ALGORYTMY SZEREGOWANIA ZADAŃ

Czesław Smutnicki

"Wzrost liczby absolwentów w Politechnice Wrocławskiej na kierunkach o kluczowym znaczeniu dla gospodarki opartej na wiedzy"

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Recenzenci:
Ewa Skubalska-Rafajłowicz

Redaktor serii:
Xxxxxxxxxxxxx

OFICYNA WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

ISBN 978-83-7493-713-9

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Spis treści

1

Przedmowa wydania pierwszego

Książka adresowana jest do studentów, doktorantów oraz absolwentów specjalizujących się w dziedzinie szeregowania zadań, projektantów systemów zarządzania, planowania i sterowania produkcją jednostkową, krótko- i średnio-seryjną, projektantów systemów zrobotyzowanych, a także informatyków zajmujących się implementacją algorytmów komputerowych.

Zagadnienia szeregowania modelują funkcjonowanie rzeczywistych systemów wytwarzania i mogą być stosowane do rozwiązywania praktycznych problemów optymalizacji i sterowania, występujących m.in. w konwencjonalnych i elastycznych systemach wytwórczych, systemach operacyjnych maszyn cyfrowych, systemach wspomagających podejmowanie decyzji, systemach zarządzania.

Książka zawiera przegląd wybranych problemów szeregowania zadań z bogactwem metod i algorytmów stosowanych do ich rozwiązywania, w dużej części projektowanych i badanych przez autora. Wskazuje alternatywne techniki i podejścia polecane dla konkretnych klas problemów praktycznych. Pokazuje przykłady zastosowań, narzędzia programowe i algorytmiczne, stosowane w praktyce oraz nowoczesne, zaskakujące swoją budową i efektywnością, metody rozwiązywania. Przedstawione podejścia mogą być po odpowiedniej modyfikacji stosowane również do analizy wielu złożonych problemów optymalizacji, w tym także dyskretnej i kombinatorycznej, trudych poprzez brak klasycznych własności analitycznych (różniczkowalność, wypukłość), wieloekstremalność czy też przekleństwo wymiarowości. Osiągnięcia teoretyczne, implikują bezpośrednio poprawę jakości komercyjnych pakietów oprogramowania, wspierających działania człowieka w wielu dziedzinach ży-

cia.

Uzupełnieniem analiz teoretycznych zawartych w książce są kody źródłowe algorytmów w języku C/C++ wraz z opisem, dostępne w witrynie internetowej autora.

2

Przedmowa wydania drugiego

Obserwowany w ostatnich latach rozwój technologii w zakresie sprzętu komputerowego skutkujący wzrostem mocy obliczeniowej PC, możliwość realizacji obliczeń rozproszonych i równoległych już na *wielordzeniowych* PC albo w *chmurach obliczeniowych*, oraz równoczesny rozwój teorii (metodologii) rozwiązywania, powodują szybkie dezaktualizowanie się podejść i algorytmów rozwiązywania uznawanych dotychcza za najkorzystniejsze. Oznacza to, że najlepszy rekomendowany algorytm (champion) dla ustalonego problemu szeregowania zmienia się średnio co 1–2 lata. W całym zakresie teorii szeregowania zadań skutkuje to także możliwościami rozwiązania problemów rozszerzonych (z pokrewnym kryterium lub bardziej złożonych modelujących dokładniej rzeczywistość) w czasie aceptowalnym przez praktyków. Stąd wersja rozszerzona książki została wzbogacona głównie o: (a) nowe technologie (metody) rozwiązywania, które pojawiły się w okresie kilku ostatnich lat, (b) nowe zagadnienia dotychczas nieznane (nieformułowane) lub niemożliwe dotychczas do rozwiązania praktycznie ze względu na znaczną złożoność problemu i/lub kłopoty numeryczne algorytmu, (c) podejścia umożliwiające efektywne wykorzystanie obliczeń równoległych lub obliczeń w chmurze, (d) modyfikacje podejść tradycyjnych sekwencyjnych zmierzających do redukcji kosztu obliczeń. Mamy jednocześnie świadomość, że rozszerzenie zakresu książki nie wyczerpuje całkowicie najbardziej interesujących zagadnień współczesnych systemów wytwórczych.

Książka aktualnie adresowana jest do studentów końcowych lat studiów pierwszego i drugiego poziomu specjalizujących się w zakresie sterowania produkcją, wytwarzaniem, magazynowaniem, montażem w systemach wy-

twórczych różnego typu, absolwentów pracujących w zakresie zarządzania i planowania produkcją, projektantów systemów zarządzania, planowania i sterowania produkcją jednostkową, krótko- i średnio-seryjną, doktorantów, a także informatyków tworzących systemy MRP, ERP, itp.

3

Wstęp

Zagadnienia szeregowania stanowią aktualnie przedmiot intensywnych studiów i badań. Wynika to z kilku różnych powodów.

Pierwszego powodu dostarcza praktyka. Według danych statystycznych, pod koniec lat dziewięćdziesiątych, tylko w USA, istniało około 40,000 firm produkcyjnych wytwarzających produkty lub części w formie "zadań produkcyjnych", zatrudniając łącznie około 2 mln pracowników i dostarczając rocznie produkcję wartości 3 mld dolarów. Zagadnienia szeregowania modelują funkcjonowanie rzeczywistych systemów wytwarzania i mogą być bezpośrednio stosowane zarówno do rozwiązywania praktycznych problemów optymalizacji występujących w dyskretnych procesach produkcyjnych, elastycznych systemach wytwarzania, systemach operacyjnych maszyn cyfrowych, systemach wspomagania podejmowania decyzji, systemach zarządzania i innych. Dodatkowo, obserwowany w ostatnich latach dynamiczny rozwój elastycznych, zintegrowanych systemów wytwarzania pozbawionych udziału człowieka (oraz czynnika niepewności z nim związanego) spowodował wzrost zainteresowania klasycznymi modelami deterministycznymi stanowiącymi podstawę teorii szeregowania zadań.

Drugim powodem prowadzenia badań są trudności ze skonstruowaniem efektywnych algorytmów rozwiązywania, możliwych do stosowania w praktyce - zagadnienia te zalicza się w literaturze do badań operacyjnych, programowania dyskretnego i kombinatorycznego. Znaczna złożoność zagadnień praktycznych implikuje wykładniczy czas obliczeń odpowiednich algorytmów komputerowych. Dość długo, jedną z najczęściej polecanych i stosowanych metod rozwiązania był schemat podziału i oszacowań. Wielu teoretyków i praktyków prezentowało i nadal prezentuje racjonalny pogląd, że podejście to jest bezużyteczne przy rozwiązywaniu zagadnień o praktycznych

rozmiarach i dużym poziomie ogólności. Stąd też, w ostatnich latach można zaobserwować tendencję do wyróżniania i grupowania zagadnień ze względu na ich własności i poziom ogólności. Umożliwia to określenie szeregu własności szczególnych, które mogą być użyte do konstrukcji efektywniejszych algorytmów specjalizowanych. Tak otrzymane algorytmy często są wykorzystywane jako pomocnicze dla zagadnień ogólniejszych i trudniejszym. Dodatkowo, badania te pozwoliły na zarówno na precyzyjne określenie granicy NP-trudności problemów optymalizacji dyskretnej jak i granic przydatności metod dokładnych, takich jak na przykład wymieniony wyżej schemat B&B. Kolejnym argumentem do prowadzenia badań jest wzrost możliwości narzędzi używanych do konstrukcji i badania algorytmów komputerowych. Mam tu na myśli zarówno burzliwy rozwój przybliżonych technik obliczeniowych opartych na metodach sztucznej inteligencji, jak i wzrost mocy obliczeniowej komputerów, umożliwiający rozwiązywania problemów uznawanych dotychczas za beznadziejnie trudne z obliczeniowego punktu widzenia.

Wymienione kłopoty z konstrukcją algorytmów rozwiązywania spowodowały powstanie, wewnątrz dziedziny szeregowania zadań, specyficznych podejść umożliwiających wykonanie kompletnej analizy problemu począwszy od budowy jego modelu matematycznego, poprzez zbadanie złożoności obliczeniowej, określenie charakterystycznych własności, aż do konstrukcji algorytmu rozwiązywania wraz z implementacją. Niektóre z tych metod, przekraczając ramy pojedynczych publikacji, stały się podstawą wielu użytecznych algorytmów i weszły na trwałe do teorii szeregowania zadań.

Niejako "przy okazji" prowadzenia niektórych badań nad modelami i algorytmami szeregowania zadań, w ostatnich latach rozwiniete zostały nowe, ogólne podejścia i techniki przybliżone, mające dalsze zastosowanie przy rozwiązywania trudnych problemów optymalizacji występujących w innych dziedzinach, np. planowanie zajęć, tras transportowych, pracy personelu, projektowanie filtrów cyfrowych, układów optycznych, konstrukcji mechanicznych, sieci komputerowych, kanałów telekomunikacyjnych, rurociągów, itp. Metody te są stosunkowo dobrze odporne na podstawowe kłopoty optymalizacji, takie jak na przykład brak klasycznych własności analitycznych (różniczkowalność, wypukłość), wieloekstremalność, przekleństwo wymiarowości, itp. Jednocześnie charakteryzują się zaskakująco wysoką efektywnością obliczeniową przy relatywnie małym stopniu skomplikowania algorytmu. Ostatecznie, dziedzina szeregowania jest obszarem badawczym, na którym ścierają się i są testowane przeróżne podejścia do rozwiązywania trudnych problemów optymalizacyjnych, prowadząc w konsekwencji do rozwoju strony algorytmicznej komercyjnych pakietów oprogramowania, wspierających działania człowieka w wielu różnych dziedzinach życia.

Prezentowana książka zawiera przegląd wybranych problemów szeregowania wraz z bogactwem podejść stosowanych do ich rozwiązywania. Nie dostarcza zbioru gotowych recept na rozwiązanie problemu, chociaż w wielu przypadkach wskazuje podejścia i algorytmy aktualnie najkorzystniejsze. O wiele jednak częściej wskazuje alternatywne techniki i podejścia polecane do rozwiązywania konkretnych klas problemów. Przedstawione metody mogą być, po odpowiedniej modyfikacji, z powodzeniem stosowane również do rozwiązywania innych problemów optymalizacji dyskretnej i kombinatorycznej. Uzupełnieniem teoretycznych analiz są źródłowe wersje odpowiednich procedur optymalizacji w języku C/C++, wraz z opisem, dostępne publicznie. Zatem monografia może być przydatna zarówno dla specjalistów zajmujących się problematyką szeregowania zadań, studentów, projektantów systemów zarządzania, planowania i sterowania produkcją jednostkową, krótko– oraz średnio– seryjną (MRP, MRP II, ERP, CIM), jak i informatyków zajmujących się implementacją algorytmów komputerowych.

Układ rozdziałów został dobrany tak by dostarczyć czytelnikowi zarówno podstaw do tworzenia modeli matematycznych, narzędzi teoretycznych potrzebnych do analizy i rozwiązywania problemów, przeglądu podstawowych problemów szeregowania i algorytmów ich rozwiązywania przydatnych w praktycznych procesach produkcyjnych, narzędzi programowych oraz przykładów zastosowań.

Ze względu na rozległość dziedziny, niektóre z tematów zostały z konieczności potraktowane przeglądowo. Lektura książki wymaga znajomości wybranych pojęć i metod z teorii grafów, algebry, teorii optymalizacji (w tym programowania liniowego, dyskretnego, dynamicznego, kombinatorycznego), struktur danych, teorii złożoności obliczeniowej, analizy algorytmów, badań operacyjnych. Pewne z tych zagadnień zostały omówione bardziej szczegółowo, dla pozostałych wskazano, w odpowiednich rozdziałach, pozycje literaturowe umożliwiające pogłębienie wiedzy czytelnika w wymienionych dziedzinach.

4

Wprowadzenie

Na początku jest problem praktyczny. Jego opis, przy użyciu pojęć teorii szeregowanaia, prowadzi do modelu matematycznego, przy czym jakość i adekwatność modelu jest konsekwencją przyjętych założeń. Dalej, model problemu jest podstawą dla procesu optymalizacji lub wyznaczania sterowania optymalnego, których wyniki są następnie stosowane w praktyce. W tym rozdziale wprowadzimy podstawowe, typowe pojęcia i definicje potrzebne przy konstrukcji różnych modeli matematycznych dla problemów szeregowania, przy badaniu własności tych modeli oraz formułowaniu algorytmów.

4.1 Rzeczywistość a modelowanie

Rzeczywiste dane liczbowe dotyczące analizowanego systemu wytwarzania mogą należeć do jednej z następujących kategorii: (1) *deterministyczne* – wszystkie wartości danych liczbowych są ustalone i znane a priori, (2) *probabilistyczne* – wartości danych liczbowych są realizacjami zmiennych losowych o znanych lub nieznanych (estymowanych) rozkładach prawodopodobieństwa bądź dane napływają w trakcie funkcjonowania sytemu, (3) *rozmyte* – wartości danych liczbowych są wielkościami rozmytymi i znamy odpowiednie funkcje przynależności lub ich parametry. Niezależnie od charakteru danych, funkcjonowanie systemu, rozumiane jako zależność pomiędzy jego wejściem, stanem i wyjściem, może być opisana formułami matematycznymi, (zbiorem takich formuł), relacją lub też zbiorem zdań pewnego języka. Przyjęcie założenia o charakterze danych i opisie implikuje zastosowanie określonych narzędzi teoretycznych do modelowania i analizy problemu, a mianowicie odpowiednio (1) *teorii szeregowania*, (2) *teorii kolejek*, *teorii procesów stochastycznych* oraz (3) *teorii zbiorów rozmytych*, ze wszystkimi korzyściami i nie-

dostatkami z tego faktu płynącymi. Ponieważ niedeterministyczne metody rozwiązywania często odwołują się do wyników pomocniczych klasycznych problemów deterministycznych, zatem teoria szeregowania jest traktowana jako fundamentalna dla wszystkich trzech wymienionych podejść. Złożoność problemów oraz kłopoty obliczeniowe już dla przypadku deterministycznego powodują, że problemy z danymi niepewnymi i rozmytymi były historycznie rzadziej formułowane i analizowane; postęp nastąpił i w tej dziedzinie tworząc pomost pomiędzy teorią a praktyką [288, 338].

Z praktyki zarządzania, powszechnie uważa się, że dane najczęściej są niepewne i nieprecyzyjne. Co więcej, zgodnie z prawem Murphy'ego, zmienią swoje wartości już w trakcie realizacji przyjętego rozwiązania niszcząc bezlitośnie jego optymalność, a czasami też dopuszczalność. Argument ten jest sztandarowym w dyskusjach pomiędzy praktykami, a ekspertami teorii szeregowania. Kolejnym argumentem jest zwykle wysoki koszt wyznaczania rozwiązania optymalnego – co w połączeniu z poprzednim implikuje wyraźny sceptycyzm w ocenie przydatności teorii. W miarę upływu czasu, zmian technologii wytwarzania oraz rozwoju nauki doszło do zbliżenia skrajnych poglądów praktyków i teoretyków. Nie ulega wątpliwości, że w większości klasycznych systemów wytwarzania podstawowym źródłem niepewności jest człowiek, bowiem to on powoduje zarówno nieprzewidywalny czas wykonywania czynności jak i losowe *załamania się* urządzeń (awarie, wypadki przy pracy); w skrajnych przypadkach system może być niesterowalny na skutek zakłóceń wprowadzanych przez człowieka. Problemem natury filozoficznej pozostaje rozstrzygnięcie czy korzystniej jest sterować niepewnym systemem w oparciu o model probabilistyczny czy też należałoby wdrożyć program naprawczy eliminujący losowe zakłócenia, aby dalej sterować już w oparciu o model deterministyczny. Zauważmy, że współczesne zrobotyzowane systemy wytwarzania, poprzez eliminację człowieka, mogą być analizowane, z dobrym przybliżeniem, w kategoriach systemów deterministycznych. Niestety, tak określone deterministyczne lub quasi-deterministyczne systemy wytwarzania mogą wciąż posiadać probabilistyczne otoczenie (losowy popyt na produkt), co stymuluje dalszy rozwój teorii szeregowania w kierunku lepszego modelowania otaczającej nas rzeczywistości. Dalej, wyraźny postęp w zakresie metod rozwiązywania oraz burzliwy rozwój sprzętu komputerowego zredukował znacznie koszt obliczeń nowo proponowanych algorytmów rozwiązywania czyniąc je przydatnymi dla problemów o praktycznym rozmiarze.

Wybór podejścia do modelowania i analizy wynika z cech sytemu, możliwości wykonania pomiarów danych, wiarygodności danych, mocy narzędzi teoretycznych, mocy dostępnych pakietów programowych, itp. Znajomość

wszystkich wymienionych elementów jest niezbędna do sprawnego rozwiązywania problemów praktycznych. Przykładowo, dane dotyczące czasu trwania czynności można przyjąć deterministyczne ustalone (normatywne), dokonać pomiaru ich cech losowych (wykonać serię pomiarów oraz zweryfikować hipotezę o typie rozkładu i jego parametrach, wartości średniej i wariancji), dokonać pomiaru w celu aproksymowania wartością deterministyczną (jeśli wariancja jest dostatecznie mała), dokonać pomiaru w celu określenia funkcji przynależności do reprezentacji rozmytej, ustalić funkcję przynależności w oparciu o opinię eksperta.

Niniejsza książka jest poświecona wyłącznie problemom deterministycznym, z pełną informacją o danych i stanie procesu, wyznaczających rozwiązania w trybie off-line. Problemy te traktowane są jako bazowe do modelowania i rozwiązywania zagadnień pokrewnych. Niektóre z omawianych podejść można zastosować do szeregowania zadań w trybie on-line i w systemach szeregowania stochastycznego.

4.2 Pojęcia podstawowe

Problemy szeregowania odwołują się do następujących elementarnych pojęć: *zadania* (zlecenia) oraz *zasoby*. Zadania polega na wykonaniu ciągu czynności zwanych *operacjami*, z których każda wymaga zaangażowania określonych zasobów. Zadaniem może być proces obróbki detalu w przemyśle maszynowym, proces montażu – w przemyśle samochodowym, realizacja inwestycji – w budownictwie, przygotowanie promu kosmicznego do wystrzelenia - w realizacji projektu naukowo-badawczego, czy też przetworzenie partii surowca – w przemyśle petrochemicznym. Zasobami są urządzenia, personel, materiały, kapitał czy też surowce energetyczne potrzebne do realizacji zadań. Zarówno zadania jak i zasoby posiadają swoje cechy charakterystyczne, dość oczywiste intuicyjnie, patrz na przykład prace [37,38,183]. W przypadku zadań wymienia się, między innymi, *termin gotowości* (termin pojawienia się zadania), *żądany termin zakończenia*, *przerywalność operacji* (dopuszczenie przerywania wykonywania), *podzielność operacji* (dopuszczenie dekompozycji operacji), *sposoby wykonywania operacji* (szczegółowe żądania zasobowe, alternatywne sposoby wykonywania). W przypadku zasobów pod uwagę bierze się trzy podstawowe ich kategorie: odnawialne (procesor, maszyna, robot), nieodnawialne (surowce, materiały podlegające zużyciu), podwójnie ograniczone (energia, kapitał). Zasoby odnawialne posiadają ograniczenie strumienia dostępności, zasoby nieodnawialne – ograniczenie globalnej ilości, zaś podwójnie ograniczone – oba rodzaje ograniczeń. Spośród wielu istot-

nych cech zasobów najczęściej wymienia się *dostępność* (czasowe przedziały dostępności), *ilość, koszt, podzielność* (w sposób dyskretny lub ciągły), *przywłaszczalność*. Należy zauważyć, że zarówno model, algorytm, jak i złożoność problemu zależy od sposobu interpretacji zasobów, te zaś dopuszczają pewną swobodę. Przykładowo, trzy osoby, Abacki, Babacki i Cabacki mogą być traktowane jako trzy różne zasoby odnawialne, niepodzielne, każdy w ilości jednej jednostki, lub jako jeden zasób (personel) odnawialny w ilości 3 jednostek, podzielny w sposób dyskretny z dokładnością do jednostki; w tym przypadku wybór sposobu interpretacji zależy od użytkownika. Niniejsza książka jest poświęcona wyłącznie systemom wykorzystującym specyficzny rodzaj zasobów odnawialnych nazywanych dalej *maszynami* (procesorami).

Oznaczmy przez $\mathcal{J} = \{1, 2, \ldots, n\}$ zbiór zadań, które mają być wykonane przy użyciu zbioru różnych typów maszyn $\mathcal{M} = \{1, 2, \ldots, m\}$. Każde zadanie $i \in \mathcal{J}$ składa się z ciągu o_i operacji $O_i = (l_{i-1} + 1, l_{i-1} + 2, \ldots, l_i)$, $l_i \stackrel{\text{def}}{=} \sum_{k=1}^{i} o_k$, $l_0 \stackrel{\text{def}}{=} 0$. Operacje w obrębie zadania muszą być wykonywane w podanej kolejności (porządku technologicznym), tzn. każda operacja j ma być wykonywana po wykonaniu operacji $j-1$, a przed wykonaniem operacji $j+1$. Dla uproszczenia zapisu zbiór operacji zadania i-tego oznaczmy podobnie przez $\mathcal{O}_i$. Dla każdej operacji $j \in \mathcal{O} \stackrel{\text{def}}{=} \sum_{i=1}^{n} \mathcal{O}_j$ określane są następujące pojęcia

M_j – ciąg zawierający m_j podzbiorów maszyn $M_j = (\mathcal{M}_{1j}, \ldots, \mathcal{M}_{m_j j})$, $\mathcal{M}_{ij} \subseteq \mathcal{M}$, określający alternatywne sposoby wykonywania operacji; operacja j potrzebuje do wykonania pewnego zestawu maszyn $\mathcal{M}_{ij}$, $1 \leqslant i \leqslant m_j$,

p_{ij} – czas wykonania operacji j sposobem i-tym $(1 \leqslant i \leqslant m_j)$,

v_j – sposób wykonywania operacji (zmienna decyzyjna), $v_j \in \{1, \ldots, m_j\}$,

S_j – termin rozpoczęcia wykonywania (zmienna decyzyjna),

C_j – termin zakończenia wykonywania, $C_j = S_j + p_{v_j j}$.

Dla każdego zadania są określane następujące pojęcia

o_i – liczba operacji w zadaniu,

r_i – najwcześniejszy możliwy czas rozpoczęcia wykonania zadania,

d_i – żądany czas zakończenia wykonywania zadania,

$\mathcal{S}_i$ – termin rozpoczęcia wykonywania zadania, $\mathcal{S}_i \stackrel{\text{def}}{=} S_{l_{i-1}+1}$,

$\mathcal{C}_i$ – termin zakończenia wykonywania zadania, $\mathcal{C}_i \stackrel{\text{def}}{=} C_{l_i}$,

$\mathcal{L}_i$ – nieterminowość zakończenia zadania, $\mathcal{L}_i \stackrel{\text{def}}{=} \mathcal{C}_i - d_i$

$\mathcal{T}_i$ – spóźnienie zakończenia zadania, $\mathcal{T}_i \stackrel{\text{def}}{=} [\mathcal{C}_i - d_i]^+$, $[x]^+ \stackrel{\text{def}}{=} \max\{0, x\}$,

$\mathcal{E}_i$ – przyśpieszenie rozpoczęcia zadania, $\mathcal{E}_i \stackrel{\text{def}}{=} [r_i - \mathcal{S}_i]^+$

$f_i(t)$ – niemalejąca funkcja reprezentująca koszt związany z zakończeniem zadania i w chwili $t \geqslant 0$,

$g_i(t)$ – nierosnąca funkcja reprezentująca koszt związany z rozpoczęciem zadania i w chwili $t \geqslant 0$,

$\mathcal{F}_i$ – czas przepływu zadania przez system, $\mathcal{F}_i \stackrel{\text{def}}{=} \mathcal{C}_i - r_i$

$\mathcal{W}_i$ – czas przestoju zadania przy przepływie przez system, $\mathcal{W}_i \stackrel{\text{def}}{=} \mathcal{F}_i - \sum_{j \in \mathcal{O}_i} p_{v_j j}$

$\mathcal{I}_k$ – czas przestoju (bezczynności) maszyny k,

$\mathcal{U}_i$ – jednostkowe spóźnienie zadania definiowane jako $\mathcal{U}_i = 1$ jeśli $\mathcal{C}_i > d_i$ oraz $\mathcal{U}_i = 0$ jeśli $\mathcal{C}_i \leqslant d_i$.

Symbol $[x]^+ = \max\{0, x\}$ będzie używany w dalszym ciągu dla skrócenia zapisu. Domyślnie przyjmuje się następujące założenia i ograniczenia: (a) wykonywanie operacji nie może być przerywane, (b) każda maszyna może wykonywać co najwyżej jedno zadanie w danej chwili czasowej. Przyjęcie innych założeń wymaga jawnego ich wyspecyfikowania.

Nie każdy problem szeregowania wymaga podania wszystkich wymienionych danych oraz wprowadzenia wszystkich wymienionych zmiennych decyzyjnych. Zwykle używamy minimalnego zbioru pojęć wystarczającego do opisu problemu. Przykładowo, w przypadku gdy dla każdego $j \in \mathcal{O}$ zachodzi $m_j = 1$, $|\mathcal{M}_{1j}| = 1$ mówimy, że problem posiada maszyny *dedykowane* bowiem zmienne decyzyjne v_j nie podlegają wyborowi; wówczas też v nie występuje w modelu problemu.

4.3 Kryteria optymalizacji

Rozwiązaniem problemu szeregowania nazywamy parę wektorów (v, S), gdzie $v = (v_1, \ldots, v_o)$ jest wektorem sposobów wykonywania operacji, zaś $S = (S_1, \ldots, S_o)$ jest wektorem terminów rozpoczęcia operacji. Rozwiązanie spełniające wszystkie ograniczenia występujące w problemie nazywamy *rozwiązaniem dopuszczalnym*. Dla uproszczenia zapisu, w problemach z maszynami dedykowanymi symbol v będziemy pomijać w oznaczeniu rozwiązania. Ponieważ interesować nas będą głównie rozwiązania dopuszczalne, zatem określenie rozwiązanie będzie się odnosić do rozwiązania dopuszczalnego chyba, że komentarz ustanowi inaczej. Zadanie optymalizacyjne polega na wyznaczeniu rozwiązania dopuszczalnego, dla którego przyjęta funkcja celu osiąga wartość optymalną. Funkcja celu reprezentuje pewną praktyczną miarę jakości funkcjonowania systemu. W zasadzie spotyka się dwie podstawowe

klasy funkcji celu

$$f_{\max} = \max_{1 \leqslant i \leqslant n} f_i(\mathcal{C}_i), \tag{4.1}$$

$$\sum f_j = \sum_{i=1}^{n} f_i(\mathcal{C}_i), \tag{4.2}$$

nazywane także kryteriami *regularnymi* oraz dwie rozszerzone klasy

$$h_{\max} = \max_{1 \leqslant i \leqslant n} \max\{g_i(\mathcal{S}_i), f_i(\mathcal{C}_i)\}, \tag{4.3}$$

$$\sum h_j = \sum_{i=1}^{n} (g_i(\mathcal{S}_i) + f_i(\mathcal{C}_i)), \tag{4.4}$$

nazywane również kryteriami *nieregularnymi*. Wszystkie kryteria należy minimalizować. Ich znaczenie praktyczne skomentujemy poniżej.

W przypadku, jeżeli $f_i(t) = t$, wtedy z wyrażenia (4.1) otrzymujemy funkcję celu reprezentującą jedno z najczęściej używanych w praktyce kryteriów, to znaczy termin zakończenia wykonania wszystkich zadań

$$C_{\max} = \max_{1 \leqslant i \leqslant n} \mathcal{C}_i. \tag{4.5}$$

Odpowiednio z wyrażenia (4.2) otrzymujemy funkcję celu związaną z sumą terminów zakończenia zadań

$$\sum C_i = \sum_{i=1}^{n} \mathcal{C}_i. \tag{4.6}$$

Bardziej znaną miarą pochodzącą od (4.6) jest średni termin zakończenia wszystkich zadań

$$\overline{C} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{C}_i. \tag{4.7}$$

Różną pilność zadań można uwzględnić na przykład poprzez przypisanie im wag $w_i \geqslant 0$ w funkcji kosztu $f_i(t) = w_i t$, co prowadzi do otrzymania z wyrażenia (4.2) funkcji celu związanej z ważoną sumą terminów zakończenia zadań

$$\sum w_i C_i = \sum_{i=1}^{n} w_i \mathcal{C}_i \tag{4.8}$$

lub analogicznej wartości średniej z miary (4.8). Afiniczne (liniowe) funkcje kosztu są wygodne również przy formułowaniu kryteriów ekonomicznych, posiadających oczywisty sens praktyczny.

W przypadku gdy zadania mają określone pożądane terminy zakończenia d_j można formułować miary wykorzystujące terminowość zakończenia zadań. Przyjmując funkcje $f_i(t) = t - d_i$, wtedy analogicznie do wyrażeń (4.5) oraz (4.7), otrzymujemy funkcje celu nazywane odpowiednio, maksymalna nieterminowość zakończenia zadań

$$L_{\max} = \max_{1 \leqslant i \leqslant n} \mathcal{L}_i = \max_{1 \leqslant i \leqslant n} (\mathcal{C}_i - d_i), \tag{4.9}$$

oraz suma nieterminowości zakończenia zadań, ważona suma nieterminowości zakończenia zadań, średnia nieterminowość zakończenia zadań,

$$\sum L_i = \sum_{i=1}^{n} \mathcal{L}_i, \quad \sum w_i L_i = \sum_{i=1}^{n} w_i \mathcal{L}_i, \quad \overline{L} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i. \tag{4.10}$$

Podobne miary można skonstruować dla spóźnień T_i, to znaczy dla $f_i(t) = \max\{0, t - d_i\}$. Otrzymujemy wtedy minimaksowe funkcje celu

$$T_{\max} = \max_{1 \leqslant i \leqslant n} \mathcal{T}_i, \tag{4.11}$$

oraz odpowiednio addytywne

$$\sum T_i = \sum_{i=1}^{n} \mathcal{T}_i, \quad \sum w_i T_i = \sum_{i=1}^{n} w_i \mathcal{T}_i, \quad \overline{T} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{T}_i. \tag{4.12}$$

Jednym z częściej wymienianych kryteriów praktycznych są miary oparte na ocenie czasu (kosztu) przepływu zadania przez system. I tak, jeżeli $f_i(t) = t - r_i$, wtedy otrzymujemy funkcje celu związane z czasem przepływu

$$F_{\max} = \max_{1 \leqslant i \leqslant n} \mathcal{F}_i, \tag{4.13}$$

$$\sum F_i = \sum_{i=1}^{n} \mathcal{F}_i, \quad \sum w_i F_i = \sum_{i=1}^{n} w_i \mathcal{F}_i, \quad \overline{F} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{F}_i. \tag{4.14}$$

Innymi użytecznymi kryteriami są miary oceniające czas oczekiwania (bezczynności) zadania w systemie. Jeżeli $f_i(t) = t - r_i - \sum_{j \in \mathcal{O}_i} p_{v_j j}$, wtedy otrzymujemy funkcje celu

$$W_{\max} = \max_{1 \leqslant i \leqslant n} \mathcal{W}_i, \tag{4.15}$$

$$\sum W_i = \sum_{i=1}^{n} \mathcal{W}_i, \quad \sum w_i W_i = \sum_{i=1}^{n} w_i \mathcal{W}_i, \quad \overline{W} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{W}_i. \tag{4.16}$$

Funkcje celu (4.5) – (4.16) są związane z oceną przepływu zadań przez system wytwórczy, co wymaga uwzględnianie zasadniczo tylko terminów zakończenia zadań $\mathcal{C}_i$. W wielu praktycznych sytuacjach istotnym problemem jest właściwe wykorzystywanie posiadanych środków zasobowych (maszyn). Do tego celu proponowana jest inna grupa miar zawierająca między innymi następujące kryteria: czas przestoju wszystkich maszyn

$$\sum I_k = \sum_{k=1}^{m} \mathcal{I}_k, \tag{4.17}$$

gdzie

$$\mathcal{I}_k \stackrel{\text{def}}{=} H_k - \sum_{j\in\mathcal{O};\, v_j=k} p_{kj}, \tag{4.18}$$

jest czasem przestoju maszyny k w planowanym przedziale pracy $[0, H_k]$ (często $H_k = \max_{1\leqslant i\leqslant n} \mathcal{C}_i$), ważona suma czasów przestoju maszyn

$$\sum w_k I_k = \sum_{k=1}^{m} w_k \mathcal{I}_k, \tag{4.19}$$

czy też średni stopień wykorzystania maszyn

$$\overline{I} = \frac{\sum_{j\in\mathcal{O}} p_{v_j j}}{\sum_{k=1}^{m} H_k}. \tag{4.20}$$

Kolejna grupa miar jest związana z procesem wykonywania zadań. Do nich należą między innymi: średnia liczba zadań wykonywanych, oczekujących na wykonanie oraz zakończonych w przedziale czasowym $(0, C_{\max})$.

Ostatnia grupa miar, pochodząca od (4.3) – (4.4), ma zastosowanie do oceny precyzji zakończenia zadań w systemach wytwarzania *dokładnie na czas* (just in time). Wśród najczęściej wymienianych są między innymi, maksymalny kara za odchylenie od żądanego przedziału wykonywania $[r_i, d_i]$

$$h_{\max} = \max_{1\leqslant i\leqslant n} \max\{g_i(\mathcal{E}_i), f_i(\mathcal{T}_i)\}, \tag{4.21}$$

bezwzględne odchylenie od wspólnego żądanego terminu zakończenia d

$$\sum |C_i - d| = \sum_{i=1}^{n} |\mathcal{C}_i - d| = \sum_{i=1}^{n} (\mathcal{E}_i + \mathcal{T}_i), \tag{4.22}$$

gdzie ostatnią równość otrzymano przy założeniu $d_i = d$, $r_i = d - \sum_{j\in\mathcal{O}_i} p_{v_j j}$, $i \in \mathcal{J}$. Inną miarą jest ważone odchylenie od pożądanego przedziału wykonywania $[r_i, d_i]$

$$\sum (z_i E_i + w_i T_i) = \sum_{i=1}^{n} (z_i \mathcal{E}_i + w_i \mathcal{T}_i) \tag{4.23}$$

które może być także interpretowane jako bezwzględne odchylenie od indywidualnych żądanych terminów zakończenia d_i przy założeniu $r_i = d_i - \sum_{j\in\mathcal{O}_i} p_{v_j j}$, $i \in \mathcal{J}$. Niektóre kryteria wprowadzają dodatkowo element negocjacji wspólnego terminu realizacji dostawy d pełniącego rolę zmiennej decyzyjnej w formie

$$\sum(z_i E_i + w_i T_i + y_i(d - d^*)^+) = \sum_{i=1}^{n}(z_i\mathcal{E}_i + w_i\mathcal{T}_i + y_i(d - d^*)^+) \tag{4.24}$$

gdzie d^* jest pewnym referencyjnym terminem dostawy. Dalsze modyfikacje kryteriów są kombinacją nowo wprowadzonych miar oraz klasycznych miar regularnych jak na przykład

$$\sum(z_i E_i + w_i T_i + y_i C_i) = \sum_{i=1}^{n}(z_i\mathcal{E}_i + w_i\mathcal{T}_i + y_i\mathcal{C}_i). \tag{4.25}$$

Oprócz liniowych miar precyzyjnego zakończenia wykonywania występują także miary nieliniowe, przykładowo

$$\sum(C_i - d)^2 = \sum_{i=1}^{n}(\mathcal{E}_i^2 + \mathcal{T}_i^2), \tag{4.26}$$

oraz miary oparte na ocenie rozproszenia pomiędzy czasami zakończenia zadań postaci

$$\sum|C_i - C_j| = \sum_{i=1}^{n}\sum_{j=1}^{n}|\mathcal{C}_i - \mathcal{C}_j|, \tag{4.27}$$

lub

$$\sum(C_i - C_j)^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}(\mathcal{C}_i - \mathcal{C}_j)^2. \tag{4.28}$$

Zagadnienia szeregowania z tego rodzaju kryteriami jednymi z trudniejszych w teorii szeregowanai zadań.

Kryteria optymalności występujące w praktyce są bądź bezpośrednio transformowalne do jednej z przedstawionych postaci, bądź też są kombinacją jednego lub kilku wyżej wymienionych kryteriów. Ze względu na znaczny postęp w zakresie metod rozwiązywania dokonany w ostatnich latach, możliwe jest skuteczne analizowanie i rozwiązywanie również problemów wielokryterialnych. Zauważmy, że traktując wykonanie poszczególnych zadań jako cele niezależne z miarami osiągnięcia $f_i(\mathcal{C}_i)$, postawiony problem z n kryteriami może być sprowadzony do problemu jedno-kryterialnego z celem

$$\sum_{i=1}^{n} w_i f_i(\mathcal{C}_i) + \epsilon \max_{1\leqslant i\leqslant n} f_i(\mathcal{C}_i) \tag{4.29}$$

czyli będącego kombinacją miar (4.1) oraz (4.2).

4.4 Związki między kryteriami

Dość licznie formułowane kryteria rodzą pytanie czy wszystkie wymienione są dostatecznie dobre oraz czy przypadkiem nie występują między nimi zależności pozwalające zredukować ich liczbę. W tym rozdziale interesować nas będą problemy równoważności kryteriów. Dwa kryteria są równoważne jeżeli optymalne rozwiązanie dla jednego z nich jest również optymalne dla drugiego kryterium. Podstawowe własności są formułowane dla problemów, w których v jest ustalone.

Równoważne są kryteria: $\min C_{\max}$, $\min \sum I_k$, $\min \sum v_k I_k$, $\max \overline{U}$. Istotnie, jeśli H_k jest ustalone to wartości kryteriów $\sum I_k$, $\sum v_k I_k$, $\overline{I}$ przy ustalonym v nie zależą od S. Przyjmując $H_k = C_{max}$, z definicji $\sum I_k$ oraz $\sum w_k I_k$ otrzymujemy

$$\sum I_k = mC_{\max} - \sum_{k=1}^{m} \sum_{j\in\mathcal{O};\ v_j=k} p_{kj} \tag{4.30}$$

$$\sum w_k I_k = (\sum_{k=1}^{m} w_k) C_{\max} - \sum_{k=1}^{m} \sum_{j\in\mathcal{O};\ v_j=k} w_k p_{kj} \tag{4.31}$$

ponieważ wartości wyrażeń $(\sum_{k=1}^{m} w_k)$ oraz $\sum_{k=1}^{m} \sum_{j\in\mathcal{O};\ v_j=k} w_k p_{kj}$ nie nie zależą od wektora S, zatem minimalizując $\sum I_k$ lub $\sum w_k I_k$ minimalizujemy $C_{\max}$ i odwrotnie.

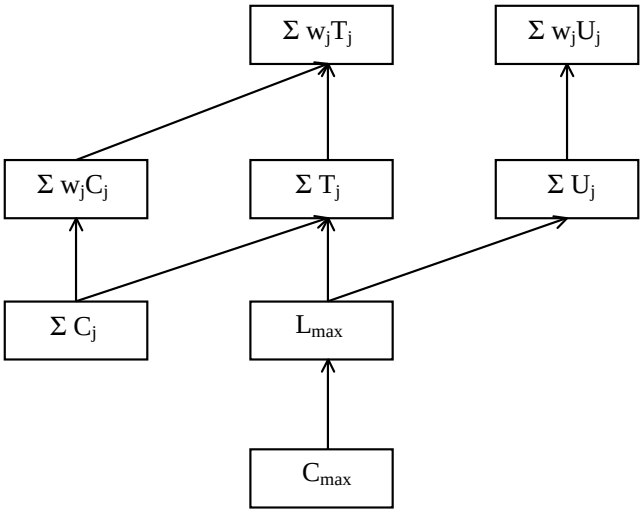
Wzajemnie równoważne są także kryteria:
$\min \sum C_i$, $\min \sum F_i$, $\min \sum W_i$, $\min \sum L_i$, $\min \overline{C}$, $\min \overline{F}$, $\min \overline{W}$, $\min \overline{L}$. Z definicji tych kryteriów otrzymujemy

$$\sum_{i=1}^{n} \mathcal{C}_i = \sum_{i=1}^{n} \mathcal{F}_i + \sum_{i=1}^{n} r_i = \sum_{i=1}^{n} \mathcal{W}_i + \sum_{i=1}^{n} r_i + \sum_{k=1}^{m} \sum_{i=1}^{n_i} p_{ik} = \sum_{i=1}^{n} \mathcal{L}_i + \sum_{i=1}^{n} d_i \tag{4.32}$$

Ponieważ wartości wyrażeń $\sum_{i=1}^{n} r_i$, $\sum_{k=1}^{m} \sum_{i=1}^{n_i} p_{ik}$, $\sum_{i=1}^{n} d_i$ są wielkościami stałymi, zatem otrzymujemy równoważność. Pozostałe równoważności możemy otrzymać w podobny sposób bezpośrednio z definicji.

Równoważne są także kryteria: $\min \sum w_i C_i$, $\min \sum w_i F_i$, $\min \sum w_i W_i$, $\min \sum w_i L_i$. Dowód tego faktu jest analogiczny do przeprowadzonych poprzednio.

Optymalna kolejność ze względu na minimalizację $L_{\max}$ jest również optymalną kolejnością ze względu na minimalizację $T_{\max}$. Istotnie, rozważmy dwie różne kolejności wykonywania zadań z wartościami opóźnień L_i



Rysunek 4.1: Zależności pomiędzy kryteriami szeregowania.

i L'_i oraz wartościami spóźnień T_i oraz T'_i. Jeżeli $L_{\max} \leqslant L'_{\max}$ to wtedy otrzymujemy

$$T_{\max} = \max_{1 \leqslant i \leqslant n} max\{0, L_i\} = \max\{0, L_{\max}\} \leqslant \max\{0, L'_{\max}\} = T'_{\max}. \quad (4.33)$$

Warto zauważyć, że kryteria $L_{\max}$ i $T_{\max}$ nie są równoważne.

Z powyższych własności wynika, że dla problemów z maszynami dedykowanymi wystarczy tylko rozważać zagadnienia z następującymi regularnymi funkcjami celu: $C_{\max}$, $\sum C_i$, $\sum w_i C_i$, $L_{\max}$, $\sum T_i$, $\sum w_i T_i$, $\sum U_i$, $\sum w_i U_i$. Zależności pomiędzy podanymi kryteriami szeregowania można przedstawić grafem na Rys. 4.1, gdzie dwa kryteria γ_1, γ_2 są w relacji $\gamma_1 \rightarrow \gamma_2$ jeśli rozwiązanie problemu z kryterium γ_2 dostarcza rozwiązania problemu z kryterium γ_1. Bardziej szczegółowe zależności pomiędzy kryteriami można znaleźć w pracy [43].

4.5 Kryteria optymalizacji a praktyka

W praktyce funkcjonowania systemów wytwarzania stosuje się pewne typowe miary oceny pracy komórek produkcyjnych. Używając pojęć charak-

terystycznych dla tych systemów, w pracy [388] wymieniono szereg wielkości, które posiadają bezpośrednią relację do wielkości wymienionych w Rozdziale 4.2 oraz kryteriów wymienionych w Rozdziale 4.3. Omówimy je poniżej.

Miary odniesione do zadań w cyklu produkcyjnym

1. *Cykl produkcyjny zadania* jest tożsamy z czasem jego przepływu $\mathcal{F}_i$.

2. *Wskaźnik wydłużenia cyklu produkcyjnego zadania* jest definiowany jako

$$\mathcal{V}_i = \frac{\mathcal{F}_i}{\mathcal{F}_i^*} \geqslant 1, \tag{4.34}$$

gdzie $\mathcal{F}_i^*$ jest daną normatywną długością cyklu dla tego zadania, przykładowo $\mathcal{F}_i^* = \sum_{j \in \mathcal{O}_i} p_{v_j j}$. Wskaźnik ten jest szczególnym przypadkiem *ważonego czasu przepływu* $w_i \mathcal{F}_i$, gdzie $w_i = 1/\mathcal{F}_i^*$.

3. *Maksymalny wskaźnik wydłużenia cykli produkcyjnych zadań* jest definiowany jako

$$V_{\max} = \max_{1 \leqslant i \leqslant n} \mathcal{V}_i \tag{4.35}$$

i jest szczególnym przypadkiem kryterium $f_{\max} = \max_{1 \leqslant i \leqslant n} f_i(\mathcal{C}_i)$ dla funkcji $f_i(t) = w_i t - w_i r_i$.

4. *Średni wskaźnik wydłużenia cykli produkcyjnych zadań* jest definiowany jako

$$\overline{V} = \frac{\sum_{i=1}^n \mathcal{F}_i}{\sum_{i=1}^n \mathcal{F}_i^*} \tag{4.36}$$

i jest szczególnym przypadkiem kryterium $\sum w_i F_i = \sum_{i=1}^n w_i \mathcal{F}_i$, z jednakowymi wagami $w_i = 1/(\sum_{i=1}^n \mathcal{F}_i^*)$. To ostatnie kryterium jest równoważne kryterium $\sum F_i$ ze względu na równość wag.

5. *Wariancja wskaźników wydłużenia cykli produkcyjnych zadań* jest definiowana jako

$$Var(V) = \frac{\sum_{i=1}^n (\mathcal{V}_i - \overline{V})^2 \mathcal{F}_i^*}{\sum_{i=1}^n \mathcal{F}_i^*}. \tag{4.37}$$

Kryterium to nie może być przedstawione w żadnej z postaci (4.1)-(4.4), jednakże jego zastosowanie jest drugorzędne. Istotnie, minimalizacja wariancji ma sens jedynie wśród rozwiązań zapewniających minimum pewnego kryterium podstawowego, jak na przykład wartości średniej $\overline{V}$, w przeciwnym przypadku będzie prowadzić do rozwiązań paradoksalnych.

6. *Cykl produkcyjny zbioru zadań* jest definiowany jako

$$CZ = \max_{1 \leqslant i \leqslant n} \mathcal{C}_i - \min_{1 \leqslant i \leqslant n} r_i, \tag{4.38}$$

jest jednoznacznie związany z kryterium $\mathcal{C}_{\max} = \max_{1 \leqslant i \leqslant n} \mathcal{C}_i$ bowiem wartość $\min_{1 \leqslant i \leqslant n} r_i$ jest stała i nie ma wpływu na minimalizację.

7. *Suma cykli produkcyjnych zbioru zadań* jest tożsama z kryterium $\sum F_i$.

8. *Średni cykl produkcyjny dla zbioru zadań* jest tożsamy z kryterium $\overline{F}$.

9. *Wariancja cykli produkcyjnych zbioru zadań* jest definiowana jako

$$Var(F) = \frac{1}{n} \sum_{i=1}^{n} (\mathcal{F}_i - \overline{F})^2. \tag{4.39}$$

Interpretacja i znaczenie tego kryterium jest podobne do opisanej wcześniej miary $Var(V)$.

10. *Wskaźnik normatywności cykli produkcyjnych zbioru zadań* jest odwrotnością wielkości $\overline{V}$ zdefiniowanej powyżej.

11. *Czas oczekiwania zadania* jest tożsamy z wartością $\mathcal{W}_i$.

12. *Średni czas oczekiwania zadań* jest tożsamy z kryterium $\overline{W}$.

13. *Kosztowy wskaźnik normatywności cykli produkcyjnych zbioru zadań* zdefiniowany jako

$$FK = \frac{\sum_{i=1}^{n} \mathcal{F}_i^* k_i}{\sum_{i=1}^{n} \mathcal{F}_i k_i} \leqslant 1 \tag{4.40}$$

gdzie k_i są normatywnymi kosztami wykonania zadań. Odwrotność tego kryterium jest szczególnym przypadkiem kryterium $\sum w_i F_i$ gdzie $w_i = k_i / (\sum_{i=1}^{n} \mathcal{F}_i^* k_i)$.

Miary dotyczące dyrektywnych terminów zakończenia zadań

1. *Opóźnienie terminu zakończenia zadania w stosunku do dyrektywnego terminu jego zakończenia* jest tożsame z wielkością $\mathcal{T}_i$.

2. *Średnie opóźnienie terminów zakończenia zadań w stosunku do dyrektywnych terminów ich zakończenia odniesione do liczby zadań wykonywanych w komórce produkcyjnej* jest tożsame z kryterium $\overline{T}$.

3. *Średnie opóźnienie terminów zakończenia zadań w stosunku do dyrektywnych terminów ich zakończenia odniesione do liczby zadań opóźnionych* jest definiowane jako

$$TU = \frac{\sum_{i=1}^{n} \mathcal{T}_i}{\sum_{i=1}^{n} \mathcal{U}_i}. \tag{4.41}$$

Kryterium to nie może być wyrażone w postaci (4.1) – (4.4), jednakże występuje w związku ze znanymi już kryteriami $TU = \overline{T}/\overline{U}$.

4. *Maksymalne opóźnienie terminu zakończenia zadań* jest tożsame z kryterium $T_{\max}$.
5. *Udział zadań opóźnionych* jest tożsamy z kryterium $\overline{U} = \frac{1}{n}\sum_{i=1}^{n} \mathcal{U}_i$.
6. *Wyprzedzenie terminu zakończenia zadania w stosunku do jego dyrektywnego terminu zakończenia* jest tożsame z wielkością $\mathcal{E}_i$ przy założeniu $r_i = d_i$.
7. *Średnie wyprzedzenie terminów zakończenia zadań w stosunku do dyrektywnych terminów ich zakończenia* jest równoważne szczególnemu przypadkowi kryterium $\overline{E} = (1/n)\sum_{i=1}^{n} \mathcal{E}_i$.
8. *Średnie odchylenie terminów zakończenia zadań od dyrektywnych terminów ich zakończenia* jest tożsame z kryterium $\overline{L}$.
9. *Koszt opóźnienia zadania* jest tożsamy z wartością $w_i\mathcal{T}_i$, gdzie w_i jest jednostkowym kosztem opóźnienia zadania.
10. *Koszty opóźnienia zadań* są tożsame z kryterium $\sum w_i T_i$.
11. *Koszty odchyleń terminów zakończenia od dyrektywnych terminów ich zakończenia* są tożsame z kryterium $\sum w_i L_i$.

Miary dotyczące prac w stanowisku

1. *Liczba operacji w kolejce przed stanowiskiem* jest funkcją $q_k(t) = |Q_k(t)|$ gdzie $Q_k(t)$ jest zbiorem operacji oczekujących w chwili t w kolejce do tego stanowiska.
2. *Liczba zadań opóźnionych w kolejce przed stanowiskiem* jest określona jako $l_k(t) = |L_k(t)|$, gdzie $L_k = \{j \in Q_k(t) : \ j \in \mathcal{O}_i, \ d_i < t\}$ jest zbiorem operacji spóźnionych oczekujących w w chwili t w kolejce do tego stanowiska.
3. *Stanowiskochłonność operacji w kolejce przed stanowiskiem* jest funkcją

$$y_k(t) = \sum_{j \in Q_k(t)} p_{kj}. \tag{4.42}$$

Miary dotyczące wykorzystania stanowiska

1. *Wskaźnik wykorzystania funduszu czasu pracy stanowiska* jest definiowany jako

$$Z_k = \frac{\sum_{j \in \mathcal{O};\, v_j = k} \frac{p_{kj}}{w_{kj}}}{H_k} \tag{4.43}$$

gdzie w^*_{kj} współczynnik wykonania normy operacji j w stanowisku k zaś H_k jest normatywnym czasem pracy stanowiska k. Dla $w_{kj} = w_k$, $v_j = k$, $j \in \mathcal{O}$, wskaźnik ten jest związany wzajemnie jednoznaczną

zależnością $Z_k = (H_k - \mathcal{I}_k)/(w_k H_k)$ z czasem przestoju maszyny $\mathcal{I}_k$ zdefiniowanym przez (4.18). Maksymalizacja Z_k pociąga minimalizację $\mathcal{I}_k$ i vice versa.

2. *Wskaźnik wykorzystania funduszu czasu pracy pracownika* jest definiowany identycznie jak Z_k, traktując pracownika jako szczególny rodzaj procesora.

3. *Suma czasów przezbrojeń stanowiska* jest definiowana jako

$$SP_k = \sum_{j=0}^{n_k} s_{\pi(j)\pi(j+1)k}, \tag{4.44}$$

gdzie permutacja π_k została otrzymana poprzez uporządkowanie zbioru operacji $\{j \in \mathcal{O};\ v_j = k\}$ o liczności n_k wykonywanych na stanowisku k według niemalejących wartości ich terminów rozpoczęcia S_j, zaś s_{ijk} jest czasem przezbrojenia stanowiska pomiędzy wykonaniem operacji i oraz j. Dodatkowo zakładamy, że $\pi(0) = 0 = \pi(n_k + 1)$ oraz s_{0jk}, s_{i0k} są czasami przygotowawczo-zakończeniowymi dla tego stanowiska.

4. *Koszty bezczynności stanowiska* są definiowane jako

$$KB_k = w_k(H_k - \sum_{j\in\mathcal{O};\ v_j=k} p_{kj}) \tag{4.45}$$

i są szczególnym przypadkiem wielkości $w_k\mathcal{I}_k$.

5. *Koszty przestoju pracownika* są definiowane identycznie jak KB_k, traktując pracownika jako szczególny rodzaj procesora.

6. *Koszty przezbrojeń stanowiska* są definiowane jako

$$KP_k = w_k \sum_{j=0}^{n_k} s_{\pi(j)\pi(j+1)k}. \tag{4.46}$$

Interpretacja i znaczenie tego kryterium jest podobne do miary SP_k opisanej wcześniej.

Miary dotyczące prac w komórce

1. *Stanowiskochłonność zadań wykonywanych w określonym przedziale czasu* (t_1, t_2) jest definiowana jako suma czasów trwania operacji wykonywanych w tym przedziale czasu

$$SZ(t_1, t_2) = \sum_{j\in\mathcal{O}} (min\{S_j + p_{v_j j}, t_2\} - max\{S_j, t_1\})^+. \tag{4.47}$$

Dla arbitralnie wybranych t_1, t_2 uzyskanie związków z innymi kryteriami jest trudne. Dla (t_1, t_2) obejmującego cały cykl wytwarzania to znaczy gdy $t_1 \leqslant min_{1\leqslant i\leqslant n} r_i$ oraz $t_2 \geqslant \max_{1\leqslant i\leqslant n} \mathcal{C}_i$ w sposób oczywisty mamy $SZ(t_1, t_2) = \sum_{j\in\mathcal{O}} p_{v_j j}$.

2. *Liczba zadań które w trakcie realizacji oczekiwały dłużej niż zadany okres czasu* W^* jest definiowana jako

$$LW = |\{i \in \mathcal{J} : \mathcal{W}_i > W^*\}|. \tag{4.48}$$

Kryterium to jest równoważne kryterium $\sum U_i$, gdzie $\mathcal{U}_i$ są definiowane w odniesieniu to żądanych terminów zakończenia $d_i = W^* + r_i + \sum_{j\in\mathcal{O}_i} p_{v_j j}$.

3. *Średnia liczba zadań w komórce produkcyjnej* jest definiowana jako

$$\overline{n}(t_1, t_2) = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} n(t)dt, \tag{4.49}$$

gdzie $n(t)$ jest liczbą zadań w komórce produkcyjnej w chwili t. Zauważmy, że zachodzi

$$\int_{t_1}^{t_2} n(t)dt = \sum_{i=1}^{n} (min\{\mathcal{C}_i, t_2\} - max\{r_i, t_1\})^+. \tag{4.50}$$

Dla arbitralnie wybranych t_1, t_2 uzyskanie związków z innymi kryteriami jest trudne. Dla (t_1, t_2) obejmującego cały cykl wytwarzania to znaczy gdy $t_1 \leqslant min_{1\leqslant i\leqslant n} r_i$ oraz $t_2 \geqslant \max_{1\leqslant i\leqslant n} \mathcal{C}_i$ w sposób oczywisty mamy $\overline{n}(t_1, t_2) = \sum_{i=1}^{n} \mathcal{F}_i/(t_2 - t_1)$. Dla ustalonych t_1, t_2 to ostatnie kryterium jest równoważne kryterium $\sum F_i$. Dla ustalonego t_1 oraz $t_2 = \max_{1\leqslant i\leqslant n} \mathcal{C}_i$, otrzymujemy związek pomiędzy kryteriami $\overline{n} = n\overline{F}/C_{\max}$.

4. *Średnia liczba zadań oczekujących w kolejkach przed stanowiskami* jest definiowana jako

$$\overline{q}(t_1, t_2) = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \sum_{k=1}^{m} q_k(t)dt. \tag{4.51}$$

Dla arbitralnie wybranych t_1, t_2 uzyskanie związków z innymi kryteriami jest trudne. Dla (t_1, t_2) obejmującego cały cykl wytwarzania, to znaczy gdy $t_1 \leqslant min_{1\leqslant i\leqslant n} r_i$ oraz $t_2 \geqslant \max_{1\leqslant i\leqslant n} \mathcal{C}_i$ w sposób oczywisty mamy $\overline{q}(t_1, t_2) = \frac{1}{t_2 - t_1} \sum_{i=1}^{n} W_i$. Dla ustalonych t_1, t_2 to ostatnie kryterium jest równoważne kryterium $\sum W_i$. Dla $t_2 = \max_{1\leqslant i\leqslant n} \mathcal{C}_i$, otrzymujemy związek pomiędzy kryteriami $\overline{q} = n\overline{W}/C_{\max}$.

5. *Średnia stanowiskochłonność operacji oczekujących w kolejkach* jest definiowana jako

$$\overline{y}(t_1, t_2) = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \sum_{k=1}^{m} y_k(t) dt. \tag{4.52}$$

Miary dotyczące wykorzystania stanowisk komórki

1. *Średni wskaźnik wykorzystania funduszy czasu stanowisk* jest definiowany jako

$$\overline{Z} = \frac{1}{m} \sum_{k=1}^{m} Z_k = \frac{1}{m} \sum_{k=1}^{m} \frac{\sum_{j \in \mathcal{O};\, v_j = k} \frac{p_{kj}}{w_{kj}}}{H_k} \tag{4.53}$$

Przy założeniu $w_{kj} = w_k$, $j \in \mathcal{O}$, $v_j = k$, wskaźnik ten jest związany jednoznaczną zależnością $m\overline{Z} = \sum_{k=1}^{m} 1/w_k - \sum_{k=1}^{m} \mathcal{I}_k/(w_k H_k)$, z kryterium typu $\sum w_k I_k$. Maksymalizacja $\overline{Z}$ pociąga minimalizację $\sum_{k=1}^{m} \mathcal{I}_k/(m w_k H_k)$ i vice versa.

2. *Wariancja wskaźników wykorzystania funduszy czasów stanowisk* jest definiowana jako

$$Var(Z) = \frac{\sum_{k=1}^{m} (Z_k - \overline{Z})^2 H_k}{\sum_{k=1}^{m} H_k}. \tag{4.54}$$

Kryterium to nie może być przedstawione w żadnej z postaci (4.1)-(4.4), jednakże jego zastosowanie jest drugorzędne. Istotnie, minimalizacja wariancji ma sens jedynie wśród rozwiązań zapewniających minimum pewnego kryterium podstawowego, jak na przykład wartości średniej $\overline{Z}$, w przeciwnym przypadku będzie prowadzić do rozwiązań paradoksalnych.

3. *Średni wskaźnik wykorzystania funduszu czasu pracowników* jest definiowany identycznie jak $\overline{Z}$, traktując pracownika jako szczególny rodzaj procesora.

4. *Suma czasów bezczynności stanowisk* jest definiowana jako

$$SB = \sum_{k=1}^{m} (H_k - \sum_{j \in \mathcal{O};\, v_j = k} p_{kj}) \tag{4.55}$$

i jest równoważna kryterium $\sum I_k$.

5. *Suma czasów przezbrojeń stanowisk* jest definiowana jako

$$SP = \sum_{k=1}^{m} \sum_{j=0}^{n_k} s_{\pi(j)\pi(j+1)k}. \tag{4.56}$$

6. *Koszty bezczynności stanowisk* są definiowane jako

$$KB = \sum_{k=1}^{m} KB_k \tag{4.57}$$

i są szczególnym przypadkiem kryterium $\sum w_k I_k$.

7. *Koszty przestoju pracowników* są definiowane identycznie jak KB, traktując pracownika jako szczególny rodzaj procesora.

8. *Koszty przezbrojeń stanowisk* są definiowane jako

$$KP = \sum_{k=1}^{m} KP_k. \tag{4.58}$$

4.6 Hierarchia złożoności obliczeniowej

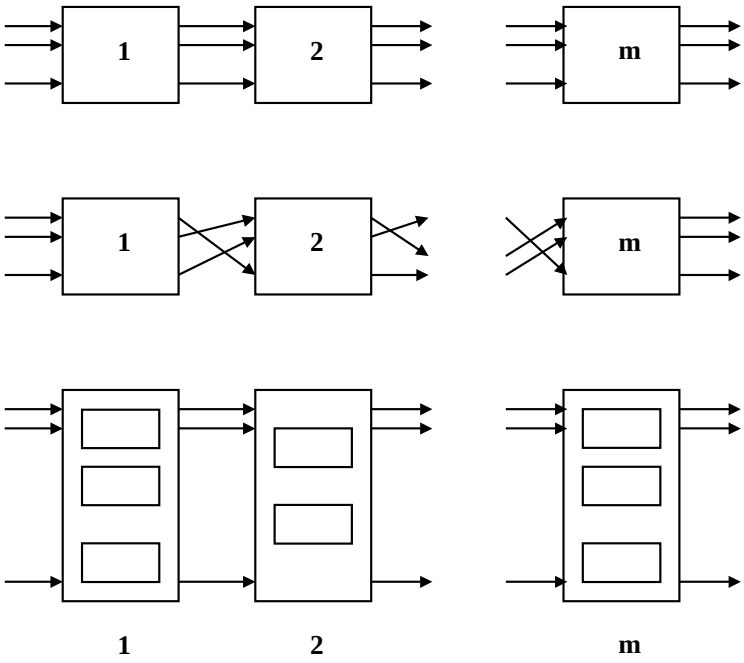
Biorąc pod uwagę, że istnieje wiele elementów składających się na określenie problemu szeregowania oraz, że dobierając te elementy otrzymamy wiele różnych problemów, w opracowano pewną uniwersalną klasyfikację zagadnień szeregowania analogiczną do tej stworzonej dla systemów masowej obsługi. Klasyfikacja ta została początkowo przedstawiona w pracy [139], a następnie dopracowana i rozszerzona w pracach [224, 308]. Ułatwia ona nie tylko zdefiniowanie problemu, ale określenie jego szczególnych własności poprzez wykorzystanie zależności hierarchicznych. W zasadzie wszystkie dotychczas rozważane zagadnienia szeregowania mogą być przedstawione przy użyciu następującego symbolicznego zapisu

$$\alpha|\beta|\gamma \tag{4.59}$$

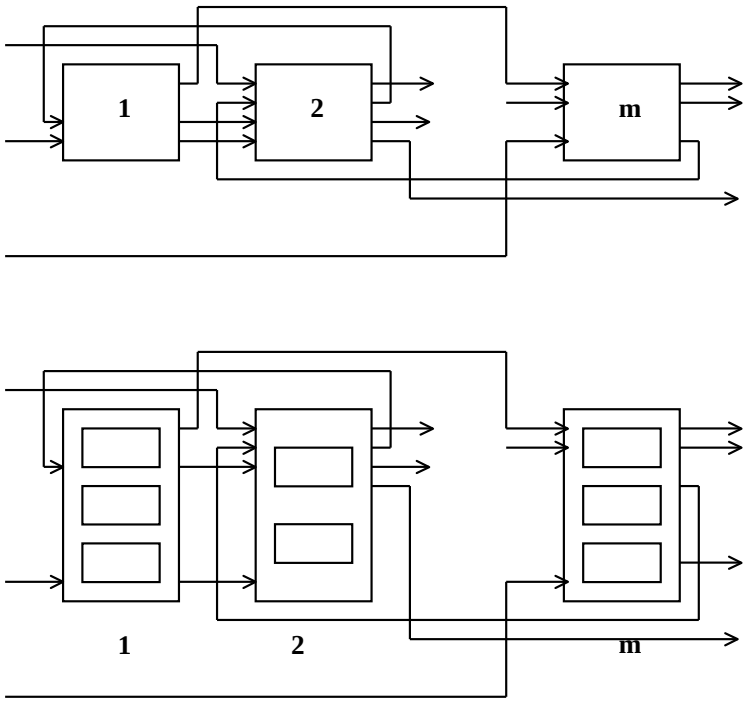
gdzie α – oznacza typ zagadnienia, β – dodatkowe specyficzne ograniczenia zagadnienia, γ – postać funkcji celu.

Znaczenia poszczególnych symboli α, β, γ mogą się nieznacznie różnić w zależności od "szkoły" naukowej, choć ogólne zasady ich wykorzystania pozostają intuicyjnie oczywiste. Podana poniżej interpretacja jest pewną próbą rozszerzenia systemu oznaczeń w celu objęcia nim także problemów o wysokim poziomie ogólności, przy jednoczwesnym zachowaniu "kompatybilności" z systemem istniejącym.

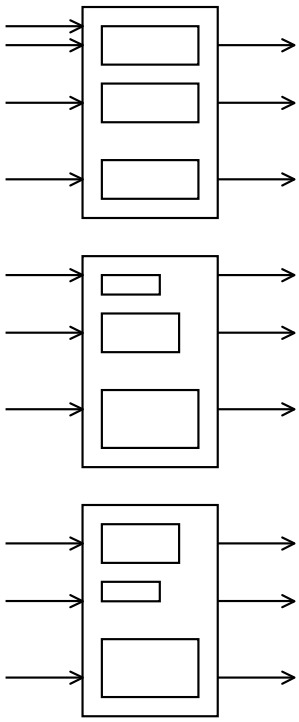
Symbol α jest złożeniem trzech symboli $\alpha_3\alpha_2\alpha_1$ mających następujące znaczenie. Symbol α_1 określa skończoną (daną) liczbę maszyn w systemie 1,2,...; jeśli liczba ta jest nieokreślona z góry to używa się symbolu pustego $\circ$ mającego sens dowolnej liczby m. Symbol α_2 określa sposób przejścia zadań przez system (inaczej – typ zagadnienia), przy czym wyróżniono następujące tradycyjne sposoby w zależności od struktury procesu wytwarzania:



Rysunek 4.2: Struktury przepływowych systemów produkcyjnych: permutacyjny (F*), niepermutacyjny (F), z maszynami równoległymi w stanowisku (PF).



Rysunek 4.3: Struktury gniazdowych systemów produkcyjnych: dedykowany (J), z maszynami równoległymi w stanowisku (PJ).



Rysunek 4.4: Struktury równoległych systemów produkcyjnych: maszyny identyczne (P), jednorodne (Q), niejednorodne (R)

F – przepływowy (flow shop), Rys. 4.2, w którym wszystkie zadania posiadają jednakową marszrutę technologiczną, wymagają obsługi na wszystkich stanowiskach, zaś każde stanowisko wymaga określenia odpowiedniej sekwencji wprowadzania zadań,

F* - przepływowy permutacyjny (permutation flow-shop), Rys. 4.2, które ma takie same założenia jak F z dodatkowym wymaganiem aby kolejność obsługi zadań na wszystkich maszynach była jednakowa (zgodna z kolejnością wprowadzania zadań do systemu),

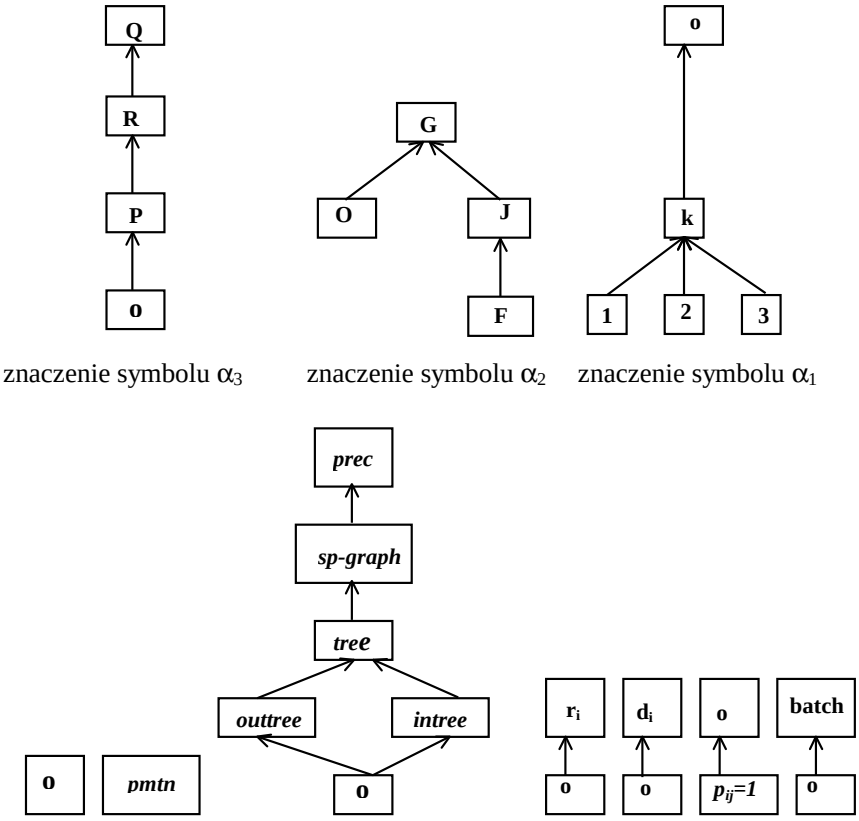
J – gniazdowy (job-shop), Rys. 4.3, w którym różne zadania mogą posiadać różne (co do liczby jak i kolejności odwiedzania stanowisk) marszruty technologiczne,

G – ogólny (general-shop), w którym każde zadanie jest pojedynczą operacją ($o_i = 1$), zaś zależności technologiczne są dane dowolnym grafem,

I - równoległy (parallel shop), Rys. 4.4, w którym każde zadanie jest pojedynczą operacją oraz wszystkie operacje są wykonywane na dokładnie jednej z kilku równoległych (tego samego typu) maszyn,

O - otwarty (open shop), w którym wszystkie operacje zadania mają być wykonane lecz kolejność technologiczna operacji w zadaniu nie jest określona.

Jeśli $\alpha_1 = 1$ to oba symbole α_2 oraz α_3 muszą być puste. Pojawiające się nowe problemy szeregowania (zwłaszcza te skojarzone z elastycznymi systemami wytwarzania) wskazują na konieczność rozszerzenie istniejącej klasyfikacji poprzez wprowadzenie symbolu α_3 określającego tryby realizacji poszczególnych operacji zadania. Jeśli α_3 jest symbolem pustym to przyjmuje się, że każda operacja ma jednoznacznie określoną (zadedykowaną) maszynę na której będzie wykonywana, to znaczy $m_j = 1$, $|\mathcal{M}_{1j}| = 1$, $j \in \mathcal{O}$. Inaczej zakłada się, że $m_j \geqslant 1$, $|\mathcal{M}_{ij}| = 1$, $i = 1, \ldots, m_j$, $j \in \mathcal{O}$ oraz operacja będzie wykonywana na jednej z identycznych równoległych maszyn (P), na jednej z maszyn jednorodnych (Q), lub jednej z maszyn niejednorodnych (R). Przykładowo, oznaczenie PF (Rys. 4.2) odpowiada systemowi F, w którym wprowadzono obsługę równoległą w stanowiskach, zaś PJ (Rys. 4.3) – odpowiedniemu wariantowi dla J. Czas trwania operacji na maszynie w trybie P jest niezależny od wyboru maszyny, w trybie R – wyraża się iloczynem pracochłonności (normatywnego czasu wykonywania) oraz szybkości pracy maszyny, zaś w trybie Q – jest specyficzny i oddzielnie określany dla każdej pary (maszyna, operacja). Jeśli $\alpha_3 \neq \circ$ oraz $\alpha_2 = \circ$ to przyjmuje się domyślnie zagadnienie typu I. W ostatnim przypadku gdy $m_j \geqslant 1$, $|\mathcal{M}_{ij}| \geqslant 1$, $i = 1, \ldots, m_j$, $j \in \mathcal{O}$ do wykonania operacji angażowany jest więcej niż jeden zasób nieodnawialny, zaś odpowiedni tryb wykonywania oznaczany jako M.



Rysunek 4.5: Redukowalność problemów w zależności od znaczenia α i β.

Symbol β określa istnienie dodatkowych założeń i ograniczeń oraz może zawierać dowolny podzbiór symboli o następującym znaczeniu:

prec – istnienieje narzucony częściowy porządek technologiczny wykonywania, zadań. Wymaganie, że zadanie i poprzedza zadanie j (co będziemy oznaczać symbolem $i \prec j$) implikuje, że w każdym dopuszczalnym rozwiązaniu mamy $\mathcal{C}_i \leqslant \mathcal{S}_j$,

tree, outree, intree, sp-graph – graf relacji *prec* posiada szczególną postać drzewa, drzewa zakorzenionego, nie zakorzenionego lub grafu szeregowo-równoległego,

r_i – zadania mają różne terminy zgłoszeń,

$C_i \leqslant d_i$ – każde zadanie musi się zakończyć przed swoim ządanym terminem zakończenia wykonywania,

no wait (bez czekania) – termin rozpoczęcia operacji następnych (w sensie porządku technologicznego) jest równy terminowi zakończenia operacji poprzednich; z tego wynika, że $\mathcal{C}_i = \mathcal{S}_i + \sum_{j=1}^{o_i} p_{v_j j}$, czyli zadanie jest wykonywane bez przerwy,

no store (bez magazynu) – brak możliwości lub zakaz składowania na składowisku międzyoperacyjnym; w tym przypadku może odbywać się składowanie jedynie na maszynach poprzez sztuczne wydłużenie czasu obróbki,

$p_{ij} = 1$ – czasy wykonania wszystkich operacji są jednakowe (i równe jedności),

pmtn (przerywać) – dopuszcza się możliwość przerywania wykonywania operacji,

M_k **– bez ograniczeń** – maszyna M_k może wykonywać jednocześnie n zadań,

$w_i = 1$ – wszystkie współczynniki wagowe zadań w kryterium są jednakowe (i równe jedności),

$o_i \leqslant o_*$ – liczba operacji w każdym zadaniu jest ograniczona stałą o_*,

$p_{ij} \leqslant p_*$ – czas wykonania wszystkich operacji jest ograniczona stałą p_*,

batch (porcjowanie) – występuje żądanie grupowania zadań,

setup (przezbrojenia) – występują czasy przezbrojenia maszyn pomiędzy wykonywaniem zadań; przezbrojenia te mogą być zależne zależne od sekwencji czynności (seq setup),

fix_j –przydział zasobów do zadań w systemie M$|\beta|\gamma$ jest ustalona.

Znaczenie symbolu β są dość często modyfikowane głównie ze względu na pojawiające się specyficzne ograniczenia wynikające z praktyki szeregowania zadań. Ostatni wymieniony parametr γ przyjmuje jedną z symbolicznych postaci funkcji celu zgodnie z oznaczeniami podanymi w poprzednim rozdziale. Przykładowo, symbol $F3|r_i|\overline{F}$ oznacza problem szeregowania zadań na trzech maszynach, każde zadanie posiada identyczną marszrutę technologiczną z różnymi czasami obsługi oraz specyficzny termin gotowości, zaś kryterium jest średni czas przepływu zadań przez system. Tak określone zagadnienia posiadają różne, wzajemnie zawierające się stopnie ogólności. Odpowiednio do tego, na Rys. 4.5 pokazano wzajemną redukowalność problemów w zależności od znaczenia parametrów α i β. W każdym z przypadków $\circ$ oznacza symbol pusty.

Jak wspomniano na wstępie, NP-trudność większości praktycznych problemów szeregowania spowodowała swoiste rozproszenie badań, koncentrując je na klasach i podklasach problemów, czy też wręcz na pojedynczych problemach szczególnych. Wprowadzona klasyfikacja pozwoliła na systematykę i uporządkowanie rezultatów badań. Posługując się wprowadzoną notacją dokonano zestawienia (zgrupowania) wyników szczególnych, takich jak złożoność obliczeniowa problemu oraz znane metody rozwiązania. Ze względu na rozległość i obszerność wyników nie będziemy powielać tabelarycznych zestawień publikowanych w innych wydaniach książkowych, odsyłając zainteresowanych do odpowiednich pozycji, patrz na przykład praca [183].

Mimi iż ostateczny rezultat normalizacji nazewnictwa można uznać za bardzo korzystny, pojawiają się pewne symptomy "niewydolności" obecnego systemu oznaczeń. Niektóre konkretne zagadnienia mogą być oznaczane niejednoznacznie za pomocą symboli np. $1|r_j|\gamma$ i $F2|M_1 - non\ bottl|\gamma$. Dalej brak jest skutecznego sposobu opisu problemów hybrydowych otrzymanych w wyniki rozszerzenia problemów klasycznych np. poprzez wprowadzenie obsługi równoległej w stanowiskach. (Zaproponowany powyżej mechanizm dwuliterowych oznaczeń dla struktur systemów jest pewnym odstępstwem od klasycznej notacji.) Wreszcie, dodatkowe ograniczenia nie wymienione powyżej, moga być uwzględniane poprzez wprowadzenie dalszych rozszerzeń listy znaczeń symbolu β. Implikuje to niekontrolowany wzrost repertuaru symboli generowanych przez poszczególnych badaczy, nie zawsze zrozumiałych bez odpowiedniego komentarza słownego.

4.7 Charakterystyka rozwiązań

W literaturze rozróżnia się kilka rodzajów uszeregowań, w zależności od posiadanych przez nie własności, patrz przegląd w pracy [368]. Każda z poniższych definicji odnosi się do uszeregowania dopuszczalnego jednie w przypadku kryterium regularnego. Kryteria nieregularne są dyskutowane w następnym rozdziale.

Uszeregowanie częściowo aktywne

Uszeregowanie jest "dosunięte w lewo" lub "częściowo aktywne" jeśli dla każdej operacji j nie można jej zakończyć wcześniej przy jednoczesnym spełnieniu następujących warunków: (1) każda operacja jest wykonywana przy użyciu nie zmienionego zbioru maszyn, (2) każda operacja inna niż j kończy się co najmniej tak samo wcześnie jak uprzednio, (3) na każdej maszynie kolejność wykonywania operacji pozostaje bez zmiany.

Uszeregowanie aktywne

Uszeregowanie jest "słabo aktywne" (aktywne) jeśli dla każdej operacji j nie można jej zakończyć wcześniej przy jednoczesnym spełnieniu następujących warunków: (1) każda operacja jest wykonywana przy użyciu nie zmienionego zbioru maszyn, (2) każda operacja inna niż j kończy się co najmniej tak samo wcześnie jak uprzednio, (3) na każdej maszynie kolejność wykonywania operacji innych niż j pozostaje bez zmiany.

Uszeregowanie silnie aktywne

Uszeregowanie jest "silnie aktywne" jeśli dla każdej operacji j nie można jej zakończyć wcześniej przy jednoczesnym spełnieniu następujących warunków: (1) każda operacja inna niż j jest wykonywana przy użyciu nie zmienionego zbioru maszyn, (2) każda operacja inna niż j kończy się co najmniej tak samo wcześnie jak uprzednio, (3) na każdej maszynie kolejność wykonywania operacji innych niż j pozostaje bez zmiany.

Uszeregowanie lewostronnie optymalne

Uszeregowanie jest "lewostronnie optymalne" jeśli dla każdej operacji j nie można jej zakończyć wcześniej przy jednoczesnym spełnieniu warunku: (1) każda operacja inna niż j kończy się co najmniej tak samo wcześnie jak uprzednio.

M_j	$j =$	1	2	3	4	5	6	7	8
{1}		2	3		2	1	2		
{2}		3			1	1		2	
{3}				1					4

Tabela 4.1: Przykład problemu

Uszeregowanie nieopóźnione

Następujące dwie klasy rozwiązań są bezpośrednią konsekwencją zasady zapobiegania bezczynności stanowisk obsługi zadań: *jeśli istnieje czynność gotowa do wykonania oraz są wolne środki zasobowe do jej wykonania, należy tą czynność niezwłocznie rozpocząć.*

Uszeregowanie (K, S) jest "słabo nieopóźnione" (nieopóźnione) jeśli nie istnieje uszeregowanie (K, S') zawierające operację j taką, że $S'_j < S_j$ oraz wszystkie maszyny ze zbioru K_j są wolne w chwili S'_j.

Uszeregowanie (K, S) jest "silnie nieopóźnione" jeśli nie istnieje uszeregowanie (K', S') zawierające operację j taką, że $S'_j < S_j$, $K'_i = K_i$, $i \neq j$ oraz wszystkie maszyny ze zbioru K'_j są wolne w chwili S'_j.

Relacje pomiędzy uszeregowaniami

Pomiędzy wymienionymi klasami uszeregowań zachodza pewne zależności. Następujące zawierania są oczywiste: każde uszeregowanie lewostronnie optymalne jest silnie aktywne, każde silnie aktywne jest słabo aktywne, każde słabo aktywne jest dosunięte w lewo. Dalsze zależności pochodzą z pracy [368] i dostarczają prostszych warunków umożliwiających identyfikację klas uszeregowań.

Uszeregowanie jest dosunięte w lewo wtedy tylko wtedy gdy nie można zakończyć żadnej operacji wcześniej przy jednoczesnym spełnieniu następujących warunków: (1) każda operacja jest wykonywana przy użyciu nie zmienionego zbioru maszyn, (2) na każdej maszynie kolejność wykonywania operacji pozostaje bez zmiany.

Uszeregowanie jest silnie aktywne wtedy tylko wtedy gdy nie można zakończyć żadnej operacji wcześniej przy jednoczesnym spełnieniu następujących warunków: (1) każda operacja jest wykonywana przy użyciu nie zmienionego zbioru maszyn, (2) każda inna operacja kończy się co najmniej tak samo wcześnie jak uprzednio.

Klasy uszeregowań słabo i silnie aktywnych są tożsame dla tych problemów dla których każda operacja może być wykonywana przy użyciu tylko

jednego ustalonego zbioru maszyn. W tym przypadku będziemy używać jednego wspólnego określenia "uszeregowanie aktywne". Każda z klas wymienionych w w pierwszych czterech definicjach zawiera co najmniej jedno rozwiązanie optymalne. W przeciwieństwie, klasy rozwiązań z dwóch ostatnich definicji nie muszą zawierać rozwiązania optymalnego! Klasy uszeregowań słabo i silnie nieopóźnionych są tożsame dla tych problemów dla których każda operacja może być wykonywana przy użyciu tylko jednego ustalonego zbioru maszyn. W tym przypadku będziemy używać jednego wspólnego określenia "uszeregowanie nieopóźnione".

Sprawdzenie czy dane uszeregowanie jest dosunięte w lewo, słabo aktywne, silnie aktywne, słabo nieopóźnione, silnie nieopóźnione może być wykonane w czasie wielomianowym. Niestety, sprawdzenie czy dane uszeregowanie nie jest lewostronnie optymalne jest problemem silnie NP-trudnym, co powoduje jego mniejszą przydatność praktyczną.

4.8 Regularność kryteriów

W przypadku kryteriów regularnych (4.1)–(4.2) rozwiązanie optymalne leży w klasie rozwiązań lewostronnie optymalnych. Jednakże ze względu na wymienioną wyżej złożoność badania tego faktu często przyjmuje się za punkt wyjścia, że leży ono w zbiorze uszeregowań silnie aktywnych. Niestety nie zawsze jest technicznie możliwe generowanie i badanie wyłącznie uszeregowań aktywnych. Niestety badanie aktywności pociąga za sobą dodatkowy nakład obliczeń, zwłaszcza w przypadku przeglądania szerszych zbiorów rozwiązań. Wtedy, głównie w celu ograniczenia kosztu obliczeń i uproszczenia algorytmu, świadomie rezygnuje się z dyskusji aktywności, rozważając najczęściej zbiór uszeregowań dosuniętych w lewo, który jest łatwiejszy do generowania i analizowania. Dlatego też większość algorytmów szeregowania dla kryteriów regularnych operuje na uszeregowanich dosuniętych w lewo.

Wymienione w rozdziale poprzednim typy nie wyczerpują całkowicie listy możliwych uszeregowań. W przypadku kryteriów nieregularnych (patrz (4.3)–(4.4)) rozwiązanie optymalne nie należy do żadnej z wymienionych klas. Nieregularność bowiem wymusza okresy celowej bezczynności maszyn spowodowane nie brakiem zadań do wykonywania lecz wartością kryterium. Powoduje to znaczne kłopoty algorytmiczne z wyznaczaniem terminów S nawet dla danej kolejności wykonywania operacji. Jak do tej pory, podklasy uszeregowań dla kryteriów nieregularnych są stosunkowo słabo zbadane głównie z powodu braku wspólnych własności.

Uszeregowanie V-kształtu

Uszeregowania V-kształtu są formułowane wyłącznie w kontekście kryteriów nieregularnych klasy (4.3)–(4.4). Istnieją w literaturze rozbieżności dotyczące tego pojęcia, głównie ze względnu na różne definicje w zależności od przypadków szczególnych funkcji kryterialnych. Pomijając wysoce specjalizowaną dyskusję różnych własności V-kształtu odwołamy się tylko do własnej, dość ogólnej definicji, pokrywającej większość wymienionych.

Uszeregowanie (v, S) posiada własność V-kształtu dla kryterium nieregularnego K jeśli $K(v, S) \leqslant K(v, S')$ dla każdego uszeregowania (v, S') posiadającego taką samą kolejność wykonywania operacji jak (v, S). Jeśli nierówność jest słaba, mówimy o słabej własności V-kształtu.

5

Struktury systemów sterowania

Systemy Planowania i Sterowania (PPC, Production Planning and Control) działają w obszarach związanych z przepływem materiałów i informacji w systemach wytwarzania. Realizują proces *planowania*, to znaczy dobór środków do realizacji wyznaczonych zadań produkcyjnych w zadanym horyzoncie czasowym i osiągnięcia postawionych celów oraz proces *sterowania*, czyli uruchamiania, nadzorowanie i zapewnianie realizacji zadań produkcyjnych. Są uważane za nadrzędne w obiegu i wymianie danych dla całego procesu wytwarzania. Współdziałają z innymi systemai informatycznymi przedsiębiorstwa takimi jak CAD (Computer Aided Design), CAP (Computer Aided Planning), CAM (Computer Aided Manufacturing), CAQ (Computer Aided Quality Control). Są trzonem systemów CIM (Computer Integrated Manufacturing). W zależności od wielkości produkcji, jej charakteru, organizacji wydziału i linii produkcyjnej, stopnia automatyzacji parku maszynowego, stosowane są strukturalnie odmienne strategie wytwarzania.

Przydatność narzędzi teorii szeregowania dla analizy odpowiednich modeli produkcyjnych zależy od wielkości i charakteru produkcji. Dla produkcji jednostkowej i krótkoseryjnej (w tym systemy ESP) odpowiednie modele szeregowania moga być stosowane bezpośrednio. Dla produkcji średnio- i wielkoseryjnej bardziej właściwe jest analizowanie zachowania się systemu w kategoriach przepływu odpowiednich strumieni materiałów niż pojedynczych elementów. W tym ostatnim przypadku cykliczność wytwarzania pozwala optymalizować narzędziami teorii szeregowania pewien fragment procesu wytwarzania odnoszący się do parametrów pojedynczego cyklu produkcyjnego, takich jak np. czas trwania cyklu, rytmiczność dostaw, synchronizację

podprocesów.

5.1 Strategia PUSH. Systemy MRP i ERP.

Strategia PUSH zakłada, że żądania wytwórcze (zamówienia na produkt końcowy) zostały przetłumaczone na żądania materiałów i półproduktów w określonych punktach wewnętrznych i na wejściach systemu dając szczegółowy bilans żądań materiałowych (MRP, Material Requirements Planning). Materiały dostarczone na wejście systemu są następnie przepychane (PUSH) za pomoca sterowań stopniowo w kierunku wyjścia systemu według ustalonego harmonogramu działań (schedule) określającego dla każdej zaplanowanej czynności termin rozpoczęcia i zakończenia oraz zbiór zasobów przydzielonych do jej realizacji. Podstawowy system MRP nie rozdziela ograniczonych zasobów (personel i maszyny) do realizacji zadań, lecz dostarcza jedynie statystyki ich dostępności i zapotrzebowań, pozostawiając podjęcie decyzji człowiekowi, który ostatecznie jest współtwórcą końcowego harmonogramu pracy systemu. Alternatywnie rozdziału takiego dostarcza system zarządzania zasobami produkcyjnymi przedsiąbiorstwa (MRP II, Manufacturing Resource Planning), którego wynikiem jest nadrzędny harmonogram pracy systemu wytwarzania (master schedule) bilansujący możliwości wykonawcze i zamówienia. Podobne cechy ma bardziej zaawansowany system dystrybuujący wszystkie zasoby przedsiębiorstwa (ERP, Enterprise Resource Planning, nazywany także czasami MRP III).

Realizacja harmonogramu nadrzędnego jest monitorowana. W razie potrzeby harmonogram jest korygowany i odpowiednie sterowania są przekazywane do systemu wytwarzania w celu uzyskania zgodności stanu systemu i wyjść z programowo założonym stanem i programową wartością wyjść. Sterowanie tego typu nosi nazwę sterowania nadążnego.

Strategie MRP, MRP II, ERP są polecane dla produkcji jednostkowej i krótkoseryjnej, przy wykorzystaniu pojedynczych maszyn w konwencjonalnych systemach wytwarzania oraz w pełni zautomatyzowanych elastycznych systemach wytwarzania (FMS, Flexible Manufacturing Systems).

Systemy sterowania oparte na strategii PUSH są systemami zcentralizowanymi (jednopoziomowymi) lub hierarchicznymi wielopoziomowymi ze zwrotnym sprzężeniem informacyjnym pomiędzy poziomami. Monitorowaniu podlegaja wszystkie stanowiska wytwarzania.

Systemy PUSH mogą być z powodzeniem modelowane i analizowane w kategoriach klasycznej teorii szeregowania zadań i rozdziału zasobów.

5.2 Strategia SQUEZEE. Systemy OPT.

Strategia SQUEZEE zakłada, że wydajność systemu wytwórczego jest ograniczona przepustowością wąskiego przekroju (nazywanego także wąskim gardłem) systemu. Przekrój ten jest zestawem stanowisk wytwórczych, przez które produkcja się przeciska (SQUEZEE) powodując spiętrzwenia i kolejki zadań. Zwiększenie wydajności systemu poprzez wzrost mocy przerobowej stanowisk (wymiana urządzeń lub rozszerzenie liczby realizatorów) jest oczywisty i nie będzie tutaj dyskutowany. Zajmiemy się wyłącznie sterowaniem przepływu zadań przez wąskie gardło. Strategia OPT ustala optymalnie harmonogram pracy stanowisk wąskiego przekroju, a następnie dostosowuje harmonogram pracy pozostałych stanowisk w celu uzyskania kompletnego rozwiązania dopuszczalnego. Celem nadrzędnym jest optymalne wykorzystanie stanowisk krytycncyh lub terminowość przejścia zadań przez cały system.

Strategia OPT nadaje się dobrze dla dla produkcji krótko- i średnioseryjnej technologicznie zorientowanej, w realizacji zleceń niewrażliwych na zmiany terminów realizacji (w tym terminu zakończenia), w przypadku niewielkiej liczby stanowisk wąskiego przekroju (jedno wąskie gardło).

Systemy sterowania oparte na strategegii SQUEZEE są systemami zcentralizowanymi (jednopoziomowymi) lub lub hierarchicznymi wielopoziomowymi ze zwrotnym sprzężeniem informacyjnym pomiędzy poziomami. W czasie pracy systemu tylko stanowiska krytyczne są monitorowane w celu korygowania bieżącego harmonogramu.

Systemy SQUEZEE moga być z powodzeniem modelowane i analizowane w kategoriach klasycznej teorii szeregowania zadań, niekiedy przy rozszerzonych klasach funkcji kryterialnych.

5.3 Strategia PULL. Systemy JIT

Filozofię wytwarzania dokładnie na czas (JIT, Just in Time) stworzyli w znacznej mierze Japończycy. Pierwsze opisane przypadki pochodziły z linii montażowej Toyoty. Strategi ta przyjmuje za podstawę produkccji zgłoszoną wielkość zapotrzebowania na określony produkt finalny, który powoduje powstanie ssania (PULL) na wyjściu systemu wytwarzania. Ssanie to jest następnie tłumaczone na ssanie materiałów i półproduktów, skierowanych pod adresem stanowaisk poprzednich i rozprzestrzenia się od wyjścia systemu przeciwprądowo w kierunku wejścia systemu. Brak ssania oznacza bezczynność systemu i stanowisk wytwarzania, zapobiegając zbędnemu wytwarzaniu

produktu na zapas. Stworzone w ten sposób informacyjne sprzężenie zwrotne pozwala na samosterowanie się systemu i jego adaptację do żądań zgłoszonych na wyjściu. Faktycznie "czysta" strategia zakłada, że nie ma żadnych zewnętrznych zmiennych sterujących, zaś sterowania lokalne są wyznaczane jednoznacznie poprzez odpowiednie sprzężenia zwrotne. Celem systemu jest możliwie szybka adaptacja aktualnego wyjścia do wyjścia zadanego, przy czym wszystkie wielkości mają charakter dynamiczny.

System JIT nie akumuluje (ZI, zero inventory) lub minimalizuje akumulację półproduktów w poszczególnych stadiach procesu poprzez dostarczanie produktów "na żądanie" i dokładnie-na-czas. W konsekwencji pozwala to ograniczyć zarówno powierzchnię produkcyjną jak i zamrożone środki kapitałowe odpowiadające produkcji w toku oraz zwiększyć płynność przepływu produkcji. Stąd często w systemach tego typu pojawiają się żądania przepływu bez czekanai (no wait), baz składowania (no store), z ograniczonym składowaniem (limited store) oraz nieregularne funkcje kryterialne żądające wykonania dostawy w ściśle określonym przedziale czasu. Jednakże w celu uzyskania pefekcyjnego funkcjonowania takiego systemu potrzeba spełnić pewne dodatkowe warunki o charakterze organizacyjnym a mianowicie: (a) niezawodne wejście do systemu materiałów i półproduktów na każde żądanie, (b) precyzyjne kontrola jakości na poszczególnych stanowiskach połączona z wysoką jakością wytwarzanego produktu, (c) niezawodność środków wytwarzania (sprzęt, personel). Tak sformułowane wymagania rodzą dalsze działania o charakterze organizacyjnym jak np. rygorystyczna i planowana konserwacja zapobiegawcza maszyn, wiarygodność poddostawców, stabilność dostaw, fachowość pracowników, szerokie wielofunkcyjne wyszkolenie pracowników, itp.

W wielu systemach JIT przyjmuje się, że przepływ zadań produkcyjnych odbywa się na zamówienie nie pojedynczo lecz małymi porcjami. Technika ta zwana Kanban powszechnie uważana jest za integralny element systemów JIT. Żądanie (ssania) na materiały lub półprodukty wewnątrz systemu ma postać karty zamówienia o określonej wielkości partii (Kanban), przesyłanej cyklicznie między kooperującymi stanowiskami. Liczba krążących kart jak i wielkość zamówienia jest ustalana arbitralnie przez nadrzędny system sterowania. W konsekwencji pomiędzy poszczególnymi stadiami procesu produkcyjnego pojawiają się ściśle kontrolowane niewielkie zapasy, które pozwalają m.in. na wygładzanie naturalnych fluktuacji procesu, zapewnienie ciągłej realizacji zamówień, szybkie reagowanie na zgłoszone zamówienia.

Strategia JIT dobrze nadaje się dla systemów wyposażonych w linie produkcyjne i gniazda wytwórcze z produkcją średnio-, wielkoseryjną i masową.

Systemy sterowania oparte na strategii PULL są stosunkowo proste i

rozproszone. Funkcje lokalnego sterowania ze sprzężeniem zwrotnym na najniższym poziomie spełniają karty Kanban. Parametry pracy tak określonego sterownika lokalnego (liczb kart, wielkość partii) mogą być ustalone przez planistyczno-sterujący człon nadrzędny, choć w praktyce po osiągnięciu stanu ustalonego procesu taka ingerencja jest rzadka.

Systemy PULL wykraczają poza ramy klsycznej teorii szeregowania zadań, ze względu na nietypowe ograniczenia i postacie funkcji kryterialnych. Dość często analizowane są poprzez modele przepływu, zaś ze względu na charakter opisu, badane narzędziami symulacyjnymi

5.4 Inne strategie CAW, CRS

Strategia CAW (Constant Average Workload) steruje zleceniami produkcyjnymi w celu zapewnienia stałęgo średniego ociążenia stanowisk. Wspomaga ona MRP II w produkcji seryjnej. Jest plecana gdy terminy dostaw są stałe, zdolności produkcyjne są niezmienne, realizacja zadań na stanowiskach jest monitorowana, dostawy materiałów są stabilne.

Strategia CRS (Continous Replenishment of Stocks) dąży do ciągłego uzupełniania bilansowanych stanó potrzeb materiałowych. Jest polecana dla produkcji, w przeważającej części seryjnej i powtarzalnej, płynnej, ze stałym zapotrzebowaniem materiałów niezależnie od długości serii, monitorowanej.

6

Aplikacje

O przydatności teorii świadczy repertuar jej możliwych zastosowań praktycznych. W przypadku teorii szeregowania są to przykłady procesów, w których algorytmy optymalizacji i sterowania można otrzymać poprzez budowę i analizę odpowiedniego modelu dyskretnego. Poniżej podamy tylko niektóre z nich.

6.1 Przemysł chemiczny

Przetwarzanie ropy naftowej oraz jej pochodnych odbywa się w wielostopniowych ciągach technologicznych złożonych z reaktorów chemicznych oraz urządzeń towarzyszących, połączonych siecią rurociągów, zbiorników składowania i pomp. Jedno zlecenie obejmuje przetworzenie pewnej partii surowca (batch). Każdy etap ciągu technologicznego jest wyposażony w jedno lub kilka jednakowych urządzeń. W ciągu technologicznym wytwarzane są produkty o różnych właściwościach co powoduje, że czas przetwarzania zlecenia na określonym stanowisku zależy od typu produktu, wielkości partii, użytego reaktora. Czasy te są znane lub zawieraja się w znanych granicach. W procesach tego typu występuja dość często nietypowe ograniczenia: żądanie natychmiastowego rozpoczęcia kolejnej operacji technologicznej ze względu na zmiany fizykochemiczne półproduktu, ograniczony czasokres oczekiwania, brak miejsc składowania półproduktu lub ograniczona ich ilość, blokowanie kilku realizatorów związane z mechanizmem przekazywania partii surowca (zbiornik-pompa-reaktor), czyszczenie reaktorów, itp. Zakładając znane zapotrzebowania na produkty końcowe, poszukiwany jest harmonogram pracy systemu, optymalny z punktu pewnego przyjętego kryterium. Lista aplikacji w procesach chemicznych i prac poświęconych tego

typu problemom jest dość długa, patrz na przykład przegląd w pracy [344].

6.2 Przemysł samochodowy

Opisany fragment systemu obejmuje dział przygotowania elementów do montażu instalacji elektrycznych wykorzystywanych następnie w linii montażowej samochodów. Skład wiązki elektrycznej wraz z elementami towarzyszącymi zależy od wyposażenia samochodu zamówionego przez klienta (typ klimatyzacji, obrotomierz, ABS, ogrzewanie, czujniki, etc.). Dla pojedynczego zamówienia punktem początkowym jest zebranie (kolekcjonowanie) odpowiedniego zestawu części składowych, których trzeba wybrać 20-50 spośród 100-500 różnych typów dostępnych. Wszystkie typy części są dostępne w kontenerach umieszczonych w pionowym regale prostokatnym (tablicy) o rozmiarze m kolumn wysokości w każda, przy czym każdy kontener zawiera wiele części określonego typu i jest uzupełniany na żądanie. Logiczna prostokatna organizacja regału może odpowiadać fizycznej budowie jedno-, dwu- lub wielo-dzielnej, której rzut na płaszczyznę podstawy jest zbliżony do kształtu liter I, U, L lub złożenia tych liter, rys. **??**. Kolekcjonowanie części dla pojedynczego zamówienia polega na transporcie pojemnika (bin) wzdłuż regału, przy czym do pojemnika pobierane są tylko niektóre części, zaś proces pobierania jest nadzorowany przez komputer. W systemach o niskim stopniu zautomatyzowania przesuwanie pojemnika i pobór części wykonuje pracownik pod dyktando informacji otrzymywanych na wyświetlaczu z komputera. Pobór części z określonej kolumny regału blokuje dostęp do części w ustalonej liczbie kolumn sąsiednich ze względu na geometrię pojemnika i pobieraka. Czas pobierania części jak czas transportu pojemnika jest w przybliżeniu stały jednak z jednej kolumny regału może być pobieranych kilka części, zaś niektóre kolumny mogą być w ogóle nie odwiedzane. Nie dopuszcza się (lub dopuszcza) wzajemnego wymijania pojemników.

Istnieje co najmnie kilka problemów teoretycznych związanych z funkcjonowaniem opisanego systemu. Posiadając określony (dany) zestaw zamówień należy wybrać kolejność ich realizacji by minimalizować pewne kryterium, np. czas realizacji wszystkich zamówień. W przypadku gdy pojemnikowi towarzyszy pracownik, kryterium może być długość trasy przez niego wykonanej. Dalej, interesować nas może sposób rozmieszczenie kontenerów w regale mający wpływ na wartość wszystkich kryteriów. Wreszcie, zakładając pewną informację aprioryczną o napływających zleceniach dziennych należy wybrać adaptacyjną strategię rozmieszczania kontenerów w regale oraz strategię obsługi zamówień, które minimalizują pewne kryterium szeregowania.

Opisany powyżej system jest przykładem ogólnej klasy systemów *kolekcjonowania* (kompletowania, zbierania) części, wkładanych do pojemnika według indywidualnego zamówienia, dość popularnych w strategii ścisłej kooperacji. W systemach o średnim i dużym stopniu zautomatyzowania, przesuw pojemników jest automatyczny, sekwencyjny, zwykle bez wymijania, podczas gdy pobieranie części jest wykonywane: (1) ręcznie przez pracownika, który musi w tym celu podejść do każdego pojemnika, (2) automatycznie przez ramię robota, który musi wykonać odpowiedni ruch ładujący. Podobnie jak poprzednio, pewne stanowiska wymagają zerowego czasu obsługi, lecz pojemniki muszą przez nie przejść. Do wykonania odpowiedniej czynności napełniającej potrzebna jest równoczesna obecność pojemnika na stanowisku i ruchomego realizatora (pracownika lub robota). Przesuw pojemników podlega ograniczeniom kolejkowania. Należy wybrać strategię przemieszczania realizatora, która minimalizuje bezwzględną długość jego trasy, czas pokonania trasy, liczbę ruchów robota lub np. czas cyklu przy obsłudze cyklicznej. Rozpatrywane są zarówno warianty deterministyczne problemu, w których można dodatkowo sterować kolejnością wprowadzania zleceń do systemu, jak i warianty obsługi na bieżąco (on-line) przy stałym napływie zgłoszeń z pewnego zbioru. Pewną, oczywistą mutacją, tego problemu jest występowanie k niezależnych pracowników. Problemy z ruchomym realizatorem występują zarówno w systemach kolekcjonowania, jak i w systemach wytwarzania (konwencjonalnych i elastycznych).

6.3 Budownictwo

Opisany system produkcyjny wytwarza bloki budowlane wykonane z betonu komórkowego. Bloki produkowane są w różnych rozmiarach i z różnych gatunków betonu, głównie na zamówienie odbiorców. Gatunek betonu ma wpływ na fizyczne parametry bloku betonowego, np. jego wytrzymałość, wagę konstrukcji budowli, izolacyjnośc, cenę, itd. Rozmiar bloku zależy od jego dalszego przeznaczenia. Bloki betonowe są wykonywane przy zastosowaniu form różnej wielkości, przy czym ani liczba ani rodzaj form nie jest ograniczony. Każda forma jest wypełniana jednym gatunkiem betonu, w wyniku czego powstaje "duży" blok, który następnie jest cięty na "mniejsze" bloki. Każde zadanie jest związane z pojedynczą formą, która musi przejść przez kolejnych osiem stanowisk w następującym porządku technologicznym : (1) przygotowanie składników, (2) mieszanie składników i wypełnianie formy, (3) wstępne utwardzanie betonu w formie, (4) cięcie bloków z uformowanego betonu, (5) kontrola jakości, (6) końcowe utwardzanie bloków w wysokiej

temperaturze i pod wysokim ciśnieniem, (7) transport bloków do magazynu, (8) transport bloków do klienta.

W rozpatrywanym systemie każde stanowisko wyposażone jest w pojedynczą maszynę bądź w kilka identycznych pracujących równolegle maszyn. Wszystkie zadania przechodzą przez wszystkie osiem stanowisk w tym samym technologicznym porządku. Zdefiniowane są także czasu przetwarzania zadania na każdym z ośmiu stanowisk. Ze względu na wymagania technologiczne, masa betonowa nie może czekać pomiędzy stanowiskami 2 i 3 oraz 3 i 4. Stosownie do specyficznego charakteru procesu produkcyjnego, bloki nie mogą być składowane pomiędzy stanowiskami 6 i 7. Czas mieszania składników oraz czas wstępnego utwardzania bloków ściśle zależy od rodzaju betonu. Czas wykonywania zadania na stanowiskach drugim i trzecim nie może być zmieniony (ani skrócony ani wydłużony), po pierwsze - ze względu na zachodzące reakcje chemiczne, po drugie - przygotowywany beton nie może być ani za miękki, ani za twardy do cięcia na stanowisku czwartym procesu technologicznego. Ponieważ operacje na stanowiskach 1, 5 oraz 8 są wykonywane z zastosowaniem dużej liczby zasobów, stanowiska te mogą być rozpatrywane jako niekrytyczne (posiadające nieograniczoną przepustowość).

Przyjmując, że dany jest zbiór zamówień, należy wyznaczyć harmonogram pracy systemu, który minimalizuje przyjete kryterium efektywności. W przypadku gdy system wytwarza wyroby cyklicznie, kryterium może odnosić się np. do długości czasu cyklu produkcyjnego.

6.4 Przemysł elektroniczny

Wytwarzanie pakietów cyfrowych w przemyśle elektronicznym, w ostatnich latach, oparte jest głównie na technologii montażu powierzchniowego (SMT, Surface Mount Technology). Proces wykonywany jest na w pełni zautomatyzowanych liniach technologicznych, począwszy od fazy wykonywania druku, poprzez montaż elementów (dużych i małych), lutowanie, testowaanie i pakowanie. Linie te mogą wykonywać różne typy pakietów oraz posiadają rożną konfigurację sprzętową w zależności technologii wytwarzania (druk jedno- i dwu-warstwowy). Przykładowo stosowane są linie jdno- i dwu-przebiegowe, z pojedynczymi lub równoległymi stanowiskami obsługi, z pojedynczym lub podwójnym przenośnikiem, z buforowaniem pośrednim lub bez. Czasy obsługi pakietu zależą od jego typu, liczby i rodzaju elementów montowanych w pakiecie, przy czym elementy małe są zwykle montowane przez robota, zaś duże zwykle ręcznie przez człowieka. Charakterystyczną

cechą procesu jest brak stanowisk składowania pomiędzy etapami technologicznymi. Przyjmując znane zapotrzebowanie na pakiety oraz dane czasy obsługi na poszczególnych stanowiskach, poszukiwany jest harmonogram pracy systemu dla wybranego kryterium optymalności. W przypadku produkcji średnio-seryjnej poszukiwane może być rozwiązanie cykliczne (cyclic schedule) minimimalizujące czas cyklu, lub rozwiązanie z grupowaniem pakietów (batch schedule). Dodatkowym ograniczeniem może być konieczność zmiany oprzyrządowania (przezbrojenie) w związku ze zmianą profilu wyrobów. Przepływowo-równoległa struktura stanowisk obsługi jest uznawana za podstawową dla systemów wytwarzania podzespołów elektronicznych.

6.5 Przemysł ciężki

Przykład zaczerpnięto z procesu hutniczego. Na wydziale prasowni huty wytwarzanymi wyrobami i podwyrobami są odkuwki lub ich partie, stanowiące podstawę do wykonania odpowiedniego elementu w wyniku dalszej obróbki mechanicznej, przykładowo: wały korbowe silników okrętowych, wirniki turbin, walce robocze dla walcowni. Wydział prasowni ma szeregowo-równoległą strukturę rozmieszczania agregatów technologicznych, a minowicie: piece grzewcze, prasy kuzienne (o różnym nacisku), środki transportu (w tym wagony do przewozu gorących wlewków oraz urządzenia dźwigowe). Proces produkcyjny polega na przepływie odkuwek przez agregaty, w celu wykonania odpowiednich czynności technologicznych. Czasy wykonywania poszczególnych operacji są znane, zaś porządek wykonywania operacji dla posczególnych odkuwek zadany jest w postaci cyklogramów. Cechą charakterystyczną procesu jest wielokrotne powtarzanie operacji podgrzewanie-kucie dla poszczególnych odkuwek . Czasy wykonywania operacji są różne dla różnych kombinacji agregat-element, przy czym ze względów technologicznych niektóre z tych kombinacji są niedopuszczalne. Celem jest ustalenie harmonogramu pracy stystemu, optymalnego z punktu widzenia pewnego przyjetego kryterium optymalności. Przyjmując, że wąskim gardłem systemu są kosztowne prasy kuzienne, optymalizacja może odnosić się do uzyskania maksymalnego stopnia wykorzystania tych agregatów.

Podobne przykłady zastosowań można znaleźć również w przemyśle hutniczym, w procesach walcowania.

7

Pakiety programowe

Korzyści, jakie może przynieść teoria praktyce można oceniać nie tylko poprzez przykłady praktyczne procesów i wdrożeń, ale także poprzez pakiety i narzędzia oprogramowania wykorzystywane w praktyce wytwarzania. Należy przy tym pamiętać, że rynek sprzetu i oprogramowania ma bardzo wysoką dynamikę rozwoju, co implikuje szybkie dezaktualizowanie się opracowań przeglądowych. Odpowiednio do modeli i narzędzi algorytmicznych, oprogramowanie podzielono na kilka klas omówionych niezależnie.

7.1 Systemy do zarządzania projektem

Projekt jest traktowany jako sieć czynności wymagających zasobów (najczęściej odnawialnych), przy czym system harmonogramowania projektu (project scheduling) nie dokonuje rozdziału zasobów lecz dostarcza jedynie bilansu lub statystyki ich żądań.

W pracy [294] przedstawiono szczegółowy przegląd oprogramowania w zakresie systemów wspomagających zarządzania projektami, dostępnych aktualnie na międzynarodowym rynku oprogramowania oraz jako share-ware sieciowy. Przegląd ten zawiera referencje do 22 systemów pracujących różnych w środowiskach m.in. DOS[1], Windows, OS/2, Unix, Sun/OS, itd. Oprócz Harvard Project Manager (historycznie najstarszego) występują tu m.in. CA-SuperProject, Microsoft Project, Time Line, Primavera Project Planner czy Artemis Schedule Publisher. Klasycznymi reprezentantami tendencji występujących w omawianej dziedzinie są m.in. systemy Project Scheduler (Scitor Corporation), CA Super Project (Computer Associates International Inc.) czy MS Project (Microsoft).

[1] Znaki handlowe wymienianych produktów są własnością ich wytwórców.

SYSTEM	DOSTAWCA
AHP-Leitstand	A.Haverman and Partner
FI-2	IDS-Prof. Scheer GmbH
FIT	Frygir Logistik Information Systems B.V.
GREEP 2.0	Frygir Logistik Information Systems B.V.
LINX	Numetrix
MOOPI	Berclain
Preactor	Incontrol
Provisa	AT&T Istel
Rhythm	Intellection
Schedulex	Numetix
Vision	Vision B.V.

Tabela 7.1: Niektóre profesjonalne systemy szeregowania

Standardem oprogramowania staje się zapewnienie użytkownikowi maksymalnej wygody pracy poprzez zastosowanie przyjaznych środowisk graficznych oraz możliwość zarządzania dużymi przedsięwzięciami (powyżej 1000 czynności i 500 zasobów) już przy wykorzystaniu małej mocy obliczeniowej PC. W większości przypadków oferowany jest jeden, dość ogólny, model problemu praktycznego oraz jedna metoda (analiza ścieżki krytycznej, PERT, CPM) jego rozwiązywania. Dlatego też systemy te umożliwiają jedynie automatyczną analizę czasową czynności pozostawiając użytkownikowi kłopoty z redystrybucją ograniczonych zasobów. Dopiero w ostatnich 5 latach pojawiły się na rynku nowe generacje systemów posiadających pewne elementy "inteligentnego" rozdziału ograniczonych zasobów. Najbardziej popularne pakiety oferują dodatkowo użytkownikowi wbudowane narzędzia programistyczne umożliwiające implementację w systemie własnego algorytmu szeregowania i rozdziału zasobów.

7.2 Systemy szeregowania zadań

Systemy szeregowania zadań wyznaczają harmonogram pracy systemu (master schedule), dla dlanego zestawu czynności przeznaczonych do wykonywania przy użyciu danego zbioru maszyn. Systemy tego typu dokonują przydziału operacji do maszyn przy założeniu pewnego kryterium szeregowania oraz w oparciu oróżne techniki konstrukcji algorytmu. Często pakiety tego typu są dedykowane dla określonych klas (struktur) systemów wytwa-

SYSTEM	UNIWERSYTET
CUISE	Columbia University
ISIS	Carnegie-Mellon University
LI	Univerität Dortmund
OPAL	Unviersité Paul Sabatier
OPIS	Carnegie-Mellon University
PRS	Cornell University
QRP	Clemson University
SCHED-STAR	Carnegie-Mellon University
SONIA	Unversité de Paris – Sud
TOSCA	University of Edinburgh
TTA	Universidad Catolika de Chile

Tabela 7.2: Akademickie prototypy systemów szeregowania

SYSTEM	WDROŻENIE
BPSS	International Paper
GATES	Transworld Airways
Jobplan	Siemens
LMS	IBM
MacMeri	Pittsburgh Plate & Glass
ReDS	Siemens

Tabela 7.3: Problemowo-zorientowane aplikacje systemów szeregowania

rzania, ze względu na specyfikę stosowanych algorytmów. Istotnie, w teorii szeregowania repertuar podejść uniwersalnych, mających zastosowanie do większości systemów, jest mocno ograniczony. Stąd krótki przegląd pakietów tego typu, podzielono na pakiety profesjonalne, prototypy akademickie oraz aplikacje problemowo-zorientowane, patrz Tab. 7.1, 7.1, 7.1.

7.3 Systemy MRP, MRP II, ERP

Przeglądu innej klasy systemów dedykowanych dla wspomagania zarządzania zadaniami i zasobami, opartych na standardzie MRP II oraz ERP, dokonano w pracy [274]. Przegląd ten zawiera referencje do 30 pakietów spełniających kryteria MRP II spośród ponad 300 pakietów przeanalizowanych. Dobrym źródłem informacji o pakietach tej klasy są także witryny internetowe, patrz np. [363]. Na podstawie dokonanych przeglądów można stwierdzić, że zdecydowanie większy nacisk jest kładziony na problemy utrzymywania rozległej bazy danych o procesie i firmie, oraz na problemy interakcji człowiek-komputer (środowiska graficzne) przy wykorzystywaniu relatywnie małej mocy obliczeniowej PC, niż na problemy optymalizacyjne. Tendencja ta odzwierciedla potrzeby rynku, na którym pierwszym krokiem do efektywnego zarądzania wytwarzaniem jest sprawne kolekcjonowanie danych i ich opracowywanie. Ponieważ światowy rynek oprogramowania typu MRP II przekroczył 1998 roku wartość 3 mln dolarów, zaś głównymi odbiorcami systemów tego typu są przemysł obronny i kosmiczny (28%), samochodowy (28%) oraz elektroniczny i konsumcyjny (25%), to właśnie oczekiwania odbiorców określają formę pakietów komercyjnych.

Strona algorytmiczna wszystkich przeglądanych systemów nieco rozczarowuje, pomimo stosowania coraz doskonalszych rozwiązań programowych. W większości przypadków oferowany jest jeden, dość ogólny, model problemu praktycznego oraz jedna metoda jego rozwiązywania. Takie systemy umożliwiają jedynie automatyczną analizę czasową czynności pozostawiając użytkownikowi kłopoty z redystrybucją ograniczonych zasobów. Dopiero w latach 1997-98 zaczęto wprowadzać do pakietów komercyjnych pewne algorytmy (głównie priorytetowe) umożliwiające automatyczny rozdział i analizę zasobów odnawialnych. W praktyce, taki system jest w większym stopniu "inteligentnym" narzędziem do zręcznej i szybkiej oceny rozwiązań proponowanych przez użytkownika, niż autentycznym "partnerem" lub "ekspertem" posiadającym umiejętność kreowania alternatywnych rozwiązań. Winą za taki stan rzeczy należałoby obarczyć zarówno teorię jak i praktykę. Przynajmniej do chwili projektowania i tworzenia istniejącego oprogramowania

SYSTEM	DOSTAWCA
Amaps	Dun&Bradstreet
ASW	IBS AB
BPCS	System Software Associates
CA-CAS	Computer Associates
CA-Masterpiece/2000	Computer Associates
CA-PRMS	Computer Associates
Chameleon 2000	Tetra
Comed	Siemens-Nixdorf
Concorde XAT	Damgaard Data A/S
Euro System	Pro-Holding
Exact	Exact Holding BVH
HP Manufacturing Management	HP
IFS Applications	IFS
Impact Award	Syspro
J.D.Edwards MRPx	J.D.Edwards Corp.
Jobshop	QMS Sage
Macola Software 6.0	Macola Inc.
MSSFS	Fourth Shift Corp.
Mapics	IBM
Max	MCS
MFG/PRO	qad. Inc.
Micro-MRP Max	Micro MRP Inc.
Movex	Intentia AB
Prodis	Software AB
Prodmax	GIPA SARL
Profit Orias BV	Orias BV
R/3	SAP AG
Renaissance CS/Promix	Ross Systems
Scala	Beshitsmodeller AB
SSPP	GTI
System 21 (d. Business 400)	JBA
TCM-EMS	EMS
Teta	Teta
Triton-Baan IV	Baan Info Systems
VPPS	Infor
WFMS	TIW Technology Inc.
Zarządzanie BPSC v.4 GL	BPSC

Tabela 7.4: Niektóre profesjonalne systemy MRP i ERP

nie były znane, lub były znane ale w zbyt małym stopniu [84], odpowiednio efektywne algorytmy rozwiązywania ogólnych problemów szeregowania i/lub rozdziału zasobów o rozmiarze występującym w przykładach praktycznych, pomimo przebadania wielu podejść i algorytmów. Z kolei praktyka odnosi się wielokroć z niechęcią do optymalizacji twierdząc, że i tak rzeczywistość odbiega od założonego pierwotnie modelu. Faktem bezspornym jest także to, iż brak jest metodologii projektowania systemów wspomagających we wspomnianym szerszym rozumieniu tego pojęcia, czego przejawem był międzynarodowy projekt badawczy koordynowany przez IIASA [343].

Obserwowany w ostatnich 2-5 latach istotny postęp w zakresie projektowania algorytmów aproksymacyjnych, rozwój techniki obliczeniowej (równoległość obliczeń, sieci neuronowe), oraz wprowadzenie technologii programowania obiektowego, otwartego, pozwalają na realizację systemu o zdecydowanie szerszych możliwościach algorytmicznych.

7.4 Pakiety symulacyjne

Trudności w modelowaniu i analizie realnie funkcjonujących systemów wytwarzania, połączone z niepewnością danych, powodują częste sięganie po narzędzia symulacyjne. Narzędzia te są dostępne w formie wyspecjalizowanych języków oprogramowania, obiektowo orientowanych bibliotek w różnych językach wspierających pracę programisty, zintegrowanych środowisk z bogatą grafiką 2D i 3D, lub kompletnych profesjonalnych pakietów łączących w/w możliwości. Chociaż są projektowane dla różnych typów procesów, zarówno ciągłych jak i dyskretnych, dalej ograniczymy się tylko do pakietów dedykowanych dla dyskretnych systemów wytwarzania. Ich działanie jest oparte na modelowaniu i analizie zdarzeń zachodzących w systemie w dyskretnych chwilach czasowych. Umożliwiaja komputerową symulację zarówno procesów przepływu materiałów jak i algorytmów obsługi i rozdziału zasobów. Pozwalają na badanie scenariuszy oraz wariantów rozwiązań (what-if). Dostarczają danych statystycznych o zachowaniu się systemu oraz narzędzi do obróbki statystycznej danych. Niektóre pakiety tego typu wymieniono w Tab. 7.4, więcej informacji można znaleźć w witrynie internetowej [335].

Korzystając z narzędzi tego typu należy pamiętać o kilku istotnych elementach. Wiarygodność symulacji zależy od wiarygodności danych wejściowych. W przypadku danych probabilistycznych lub rozmytych potrzebna jest znajomość ststystyki matematycznej przy dokonywaniu pomiaru danych rzeczywistych, w tym znajomość zagadnień estymacji, planowania i przeprowadzania eksperymentów, weryfikacji hipotez, itd. Dalej badanie scenariuszy

SYSTEM	DOSTAWCA
SIMAS II	CIMPACT Sarl
SimBax	AICOS Technologies AG Software
WITNESS	Laner Group
ARENA	Rockwell Software
Promodel	Promodel Corp.
AutoSimulations	Brooks Automation
DynaWiz	CDI

Tabela 7.5: Niektóre pakiety symulacyjne

jak i badanie cech statystycznych systemu wymaga wykonania wielu przebiegów symulacyjnych, zwykle dość kosztownych obliczeniowo. Podobnie jak porzednio, otrzymaie wiarygodnych charakterystyk probabilistycznych systemu wymaga znajmości statystyki matematycznej, zaś koszt obliczeń rośnie bardzo szybko ze wzrostem rozmiaru przykładu problemu. Wreszcie, otrzymane wyniki mają sens statystyczny i mogą nie dość dokłądnie modelować stany przejściowe systemu. Niezależnie od powyższych uwag krytycznych, pakiety symulacyjne pozostają jednym z częściej używanych narzędzi do opisu i analizy systemó dyskretnych głównie ze względu na szybkość przygotowania modelu i jego analizy.

8

Metody optymalizacji dyskretnej

Zdecydowana większość deterministycznych problemów planowania i sterowania w dyskretnych sytemach wytwarzania jest formułowana jako zagadnienia optymalizacji, w których wszystkie zmienne decyzyjne (bądź ich część) przyjmują wartości dyskretne, całkowitoliczbowe lub binarne. Zagadnienia takie, nazywane także w dalszej części problemami optymalizacji *dyskretnej* lub *dyskretno-ciągłej*, należą do klasy problemów wyjątkowo kłopotliwych z obliczeniowego punktu widzenia. W dalszym ciągu problemy te będziemy zapisywać w postaci zagadnienia minimalizacji

$$K^* \stackrel{\text{def}}{=} K(x^*) = \min_{x \in \mathcal{X}} K(x), \tag{8.1}$$

gdzie $K(x)$ jest skalarną funkcją celu, x rozwiązaniem, x^* rozwiązaniem optymalnym, zaś $\mathcal{X}$ jest zbiorem rozwiązań dopuszczalnych określonym przez zestaw warunków ograniczających. Szczególnym zainteresowaniem potraktujemy przypadki generowane przez praktykę, w których $\mathcal{X}$ jest zbiorem dyskretnym, $K(x)$ jest funkcją nieliniową, nieróżniczkowalną, zaś problem optymalizacyjny jest silnie NP-trudny.

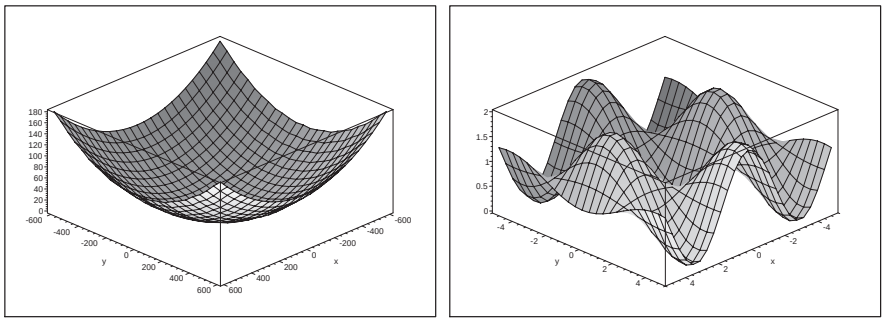
Filozofia podejść stosowanych do rozwiązywania problemów optymalizacji zmieniła się istotnie na przestrzeni ostatnich 10 – 20 lat. Problemy z unimodalnymi, wypukłymi, różniczkowalnymi, skalarnymi funkcjami celu zniknęły z laboratoriów badawczych, bowiem wiele odpowiednio efektywnych metod rozwiązywania zostało już dla nich opracowanych. Obecnie badacze skupili się on szczególnie trudnych przypadkach: wieloekstremalnych, wielokryterialnych, nieróżniczkowalnych, NP-trudnych, dyskretnych, z olbrzymim wymiarem, pochodzącym z praktyki sterowania, planowania, har-

monogramowania, transportu, projektowania, zarządzania, etc. Zagadnienia te, generowane przez przemysł, rynek i biznes, sprawiają poważne problemy w procesie poszukiwania optimum globalnego. Wiele wysiłku badaczy włożono w ostatnich latach w celu zwiększenia mocy algorytmów rozwiązywania spełniających oczekiwania praktyków. Osiągnięty postęp w rozwoju metod spowodował z kolei zwiększenie oczekiwań praktyków, zatem stale istnieje potrzeba prowadzenia badań w zakresie doskonalenia teorii i metodologii rozwiązywania wspomnianych klas zagadnień.

Główne klopoty optymalizacji przedyskutowano bardziej szczegółowo w Rozdz. 8.1. Wielokrotnie, w celu uniknięcia tych kłopotów, próbuje się zamiast rozwiązywać problem dokładnie, wyznaczyć pewne jego rozwiązanie przybliżone. Dokładność tego przybliżenia posiada tendencję przeciwstawną do czasu obliczeń, tzn. uzyskanie dokładniejszego rozwiązania wymaga dłuższego czasu pracy algorytmu przy czym ta ostatnia zależność posiada charakter silnie nieliniowy. Z tego też powodu dziedzina dyskretnych procesów wytwarzania charakteryzuje się znaczną rozmaitością zarówno modeli jak i metod rozwiązywania, zwykle dedykowanych dla wąskich klas zagadnień. Ograniczenie ogólności modeli ma na celu wykrycie tych szczególnych własności problemu, których umiejętne wykorzystanie w algorytmie zdecydowanie poprawia jego cechy numeryczne takie jak czas obliczeń, szybkość zbiegania do rozwiązania optymalnego. Często dla tego samego problemu NP-trudnego występuje w literaturze kilka, kilkanaście różnych algorytmów o istotnie różnych cechach numerycznych. Znajomość dziedziny oraz metod rozwiązywania pozwala na właściwy dobór dla każdego nowo postawionego problemu odpowiedniego algorytmu satysfakcjonującego użytkownika. Należy przy tym pamiętać, że w rozważanej dziedzinie celem *nie jest* sformułowanie jakiegokolwiek modelu i metody rozwiązania lecz celem nadrzędnym jest podanie *prostego* modelu oraz metody rozwiązywania *racjonalnej* z punktu widzenia implementowanego algorytmu komputerowego.

8.1 Kłopoty optymalizacji

Jak dotychczas zidentyfikowano kilka czynników odpowiedzialnych za kłopoty w optymalizacji powodujące wysoki koszt obliczeń i niską jakość otrzymywanych rozwiązań. Chociaż można łatwo wskazać szereg problemow praktycznych wykazujacych te własności, niemniej dla wygody czytelnika odwołamy się do powszechnie znanych instancji testowych mających bardzo dobrą ilustrację w 2D. Dodatkowo, jako ilustracje zastosowań praktycznych, będziemy wskazywać niektóre problemy szeregowania zadań.



Rysunek 8.1: Funkcja testowa Griewanka w 2D: przegląd (lewy), powiększenie (prawy)

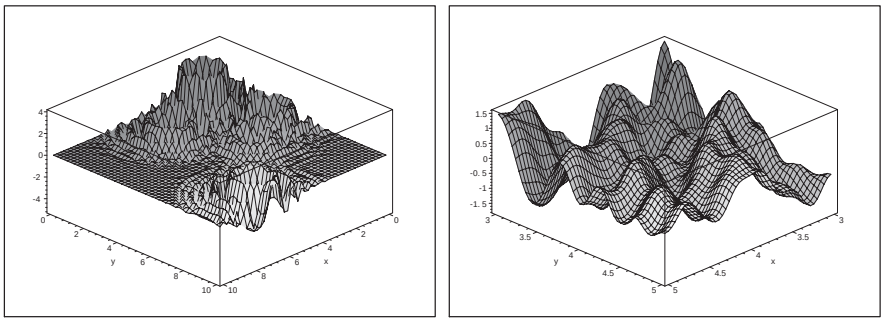
Mnogość ekstremów lokalnych

Problem ten jest dobrze ilustrowany funkcją testową Griewanka (8.2), która posiada olbrzymią liczbę ekstremów lokalnych rozmieszczonych w miarę regularnie, patrz także Rys. 8.1,

$$K(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1. \tag{8.2}$$

Minimum funkcji $f(x^*) = 0$ jest osiągalne dla $x_i^* = 0$, $i = 1, \ldots, n$. Interpretacja funkcji zmienia się wraz ze skalą obserwacji. Pobieżny przegląd sugeruje klasyczną funkcję wypukłą, unimodalną. Umiarkowana skala powiększenia sugeruje istnienie pewnej liczby ekstremów lokalnych, jednak dopiero duże powiększenie wskazuje złożoną strukturę licznych ekstremów lokalnych. Teoretycznie, uwzględniając regularność powierzchni, można łatwo zdefiniować strategiczne kierunki poszukiwań, które szybko doprowadziły by proces poszukiwań do najbardziej obiecującej części przestrzeni rozwiązań. Niestety, uwzględniając (8.2) liczba ekstremów lokalnych wzrasta wykładniczo z wymiarem przestrzeni n. Ten fakt praktycznie eliminuje metody, które badają wyłącznie małą frakcję rozwiązań lokalnych (słaba jakość) oraz zdecydowanie odrzuca metody badajace przestrzeń w sposób wyczerpujący (nieakceptowalnie duży koszt).

Dla instancji ta51 przepływowego problemu szeregowania, liczba rozwiązań x spełniających warunek $(K(x) - K^*)/K^* \leqslant 0.5\%$ (ekstrema lokalne) jest $L \approx 0.2 \cdot 10^{57}$ i stanowi nieskończenie małą frakcję $L/50! \approx 0.8 \cdot 10^{-8}$ całej przestrzeni.



Rysunek 8.2: Funkcja testowa Langermana w 2D: przegląd (lewy), powiększenie (prawy)

Rozkład ekstremów

Problem ten jest dobrze ilustrowany funkcją testową Langermanna (8.3) posiadającą liczne ekstrema lokalne nieregularnie rozłożone, patrz Rys. 8.2, w zależności od parametrów m, a_{ij}, c_i nieznanych apriori

$$K(x) = \sum_{i=1}^{m} c_i \exp[-\frac{1}{\pi}\sum_{j=1}^{n}(x_j - a_{ij})^2] \cos[\pi \sum_{j=1}^{n}(x_j - a_{ij})^2]. \tag{8.3}$$

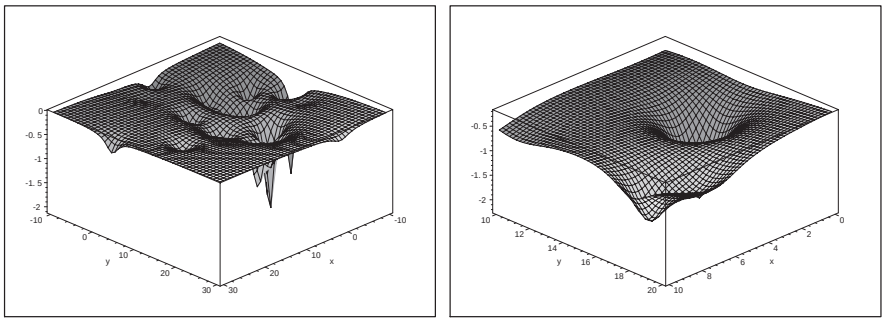
Oznacza to, że strategiczne kierunki poszukiwań w przestrzeni nie mają żadnego regularnego charakteru i muszą być ustalane w sposób adaptacyjny, w zależności od krajobrazu przestrzeni.

Hybrydowy problem przepływowy modelujący np. automatyczną linię do produkcji pakietów drukowanych objawia olbrzymią liczbę ekstremów lokalnych nieregularnie rozmieszczonych, patrz Rys. 8.4. Jest oczywiste, że skuteczna metoda poszukiwań musi badać krajobraz przestrzeni $\mathcal{X}$ w celu odpowiedniego określenia obiecujących kierunków poszukiwań.

Ekstrema zwodnicze

Ten problem jest dobrze ilustrowany funkcją testową Shakela (8.4), która objawia dość głębokie ekstrema lokalne (lisie dziury) rozłożone nieregularnie na prawie płaskiej powierzchni, patrz Rys. 8.3,

$$K(x) = -\sum_{i=1}^{m}(\sum_{j=1}^{n}[(x_j - a_{ij})^2 + c_j])^{-1}, \tag{8.4}$$



Rysunek 8.3: Funkcja testowa Shakela w 2D: przegląd (lewy), powiększenie (prawy)

gdzie $(c_i,\ i = 1, \ldots, m)$, $(a_{ij},\ j = 1, \ldots, n,\ i = 1, \ldots, m)$ są ustalonymi z góry liczbami; zaleca się przyjąć $m = 30$. Zachowanie się funkcji pomiędzy dziurami (dominujaca część powierzchni) nie dostarcza żadnej informacji o minimach spodziewanych w otoczeniu. Co więcej, iteracyjne metody poszukiwań (wykonujące krok po kroku) nie są w stanie wyjść z obserwowanych głębokich minimów lokalnych, co skutkuje przedwczesną zbieżnością i/lub stagnacją poszukiwań.

Płaskie powierzchnie z nieregularnie rozłożonymi dziurami można znaleźć, na przykład, w problemach szeregowania z całkowitą (ważoną) sumą spóźnień (tardiness). Dla tego problemu zaobserwowano w metodach lokalnych poszukiwań, że większość rozwiązań posiada tą samą wartośc funkcji celu (płaska część powierzchni). Lepsze lub gorsze rozwiązania stanowią nieznaczącą frakcję w sąsiedztwie. Bezpośrednią konsekwencją dyskutowanej własności jest rekomendacja metody rozwiązania; metody poszukiwań lokalnych nie są tutaj polecane, preferowane są metody oparte na rozproszonej populacji rozwiązań.

Przekleństwo wymiarowości

Przykłady testowe wymienione w poprzednich punktach badano typowo dla wymiaru przestrzeni $n \leqslant 20$ (rozmiar akademicki). Jak do tej pory brak jest danych na temat zachowania się odpowiednich metod rozwiązywania dla większego rozmiaru przestrzeni $n \gg 20$ (przypadki rzeczywiste). W programowaniu dyskretnym różnica pomiędzy przypadkami akademickimi, a rzeczywistymi jest dużo bardziej widoczna.

Rozpatrzmy, najstarszy historycznie rzeczywisty przykład testowy FT10

(10 zadań, 10 maszyn, 100 operacji) gniazdowego problemu szeregowania, jeden z najmniejszych rozważanych obecnie[1]. Przestrzeń rozwiązań $\mathcal{X}$ jest dyskretna, skończona, ma wymiar 90, co w porównaniu do akademickiej wartości 20 można uważać za duży wymiar (największy rozwiązywany obecnie przykład testowy tego problemu ma wymiar 1980). Jeśli do reprezentacji rozwiązania x użyjemy modelu permutacja + graf, prowadzącej do przestrzeni o najmniejszej liczności, to znajdziemy około $4 \cdot 10^{65}$ różnych rozwiązań w przestrzeni, przy czym część z nich jest niedopuszczalna. Frakcja rozwiązań dopuszczalnych do wszystkich rozwiazań zależy od danych i zmienia się od 1 do 10^{-17} (dla danych występujących w FT10). Załóżmy, że chcemy dokonać projekcji $4 \cdot 10^{48}$ ostatecznie zidentyfikowanych rozwiązań dopuszczalnych na 2D oraz wydrukować ją w formie kolorowej mapy przy użyciu drukarki 2400dpi, w której pojedynczy punkt ma rozmiar 0.01×0.01 mm. Mamy nadzieję, że mapa ta pozwoli nam wykryć nieregularnie rozmieszczone ekstrema, lub co najmniej określić strategiczne kierunki w czasie przeszukiwań sterowanych krajobrazem. W tym celu użyjemy transformacji (nieopisanej tutaj) zachowującej odległość, tj. odległe rozwiązania w przestrzeni odpowiadają odległym punktom w 2D, bliskie rozwiązania – bliskim punktom. Wynikiem wydruku jest powierzchnia wielkości $4 \cdot 10^{32}$ km^2. Dla porównania największa planeta naszego układu słonecznego Jowisz ma tylko $6.4 \cdot 10^{10}$ km^2 powierzchni. Typowa procedura poszukiwań jest w stanie sprawdzić w rozsądnym czasie co najwyżej miliard rozwiązań, co odpowiada powierzchni 0.1m^2 i może być reprezentowane nitką pajęczyny grubości 0.01 mm i długości 10 km.

Frakcja rozwiązań dopuszczalnych do wszystkich rozwiązń jest nieskończenie mała w FT10 i silnie maleje ze wzrostem wielkości przykładu. Z drugiej strony, dla tej klasy problemów rozwiązania dopuszczalne w przestrzeni dostarczają olbrzymiej liczby ekstremów lokalnych nieregularnie rozmieszczonych. Projektowany algorytm rozwiązania powinien nie tylko wbierać rozwiązania dopuszczalne, ale także identyfikować najbardziej obiecujące regiony do eksploracji.

NP-trudność

Większość problemów optymalizacji dyskretnej pochodzących z praktyki (szeregowanie, harmonogramowanie, transport, plany zajęć, itp.) jest NP-trudnych, co natychmiast implikuje wykładniczą złożoność obliczeniową algorytmu rozwiązywania. Ponieważ moc obliczeniowa procesorów wzrasta

[1]Przykład ten czekał na rozwiązanie 25 lat.

liniowo w ostatnich latach, podczas gdy koszt obliczeń wzrasta wykładniczo z rozmiarem problemu, nie ma specjalnych nadziei na rozwiązywanie optymalne rzeczywistych przykładów problemów w akceptowalnym w praktyce czasie.

Koszt obliczeń

NP-trudność implikuje nieakceptowalnie duży czas obliczeń mierzony czasem pracy procesora. Co więcej, problemy dyskretne są uważane za nadmiernie sztywne w tym sensie, że niewielkie zaburzenie wartości danych niszczy optymalność rozwiazania znalezionego w dość kosztowny sposób, zmuszając użytkownika do ponownego wykonania kosztownych obliczeń. Stąd, poszukiwanie optymalnego rozwiązania nie jest zbyt popularne w społeczności praktyków.

8.2 Krajobraz przestrzeni

Jest powszechnie wiadome, że zachowanie się algorytmu rozwiązywania musi być dostosowane do krajobrazu (landscape) przestrzeni rozwiązań. Identyfikacja tych szczególnych własności pozwala zaprojektować algorytm najlepiej dopasowany do problemu, zatem najbardziej efektywny.

Odległość

Klasyfikacja i ocena krajobrazu przestrzeni rozwiązań wymaga posługiwania się pojęciem odległości rozwiązań w przestrzeni $\mathcal{X}$. W przestrzeniach euklidesowych znane są typowe miary odległości. Zauważmy, że wynik końcowy analizy krajobrazu (zatem też wnioski) zależą od zastosowanej miary odległości. W przestrzeniach dyskretnych zagadnienie wyboru odpowiedniej miary jest dużo bardziej złożone, bowiem pod uwagę należy brać nie tylko cechy miary ale także możliwość jej policzenia i złożoność obliczeniową. Przykładowo w Tabl. 8.1 zebrano alternatywne miary pomiędzy permutacjami, podstawowe dla problemów TSP, QAP, jak również wielu problemów szeregowania.

Rozkłady

Rozkład rozwiązań oraz minimów lokalnych w dyskretnej przestrzeni rozwiązań jest zwykle nieregularny. Można to sprawdzić obserwujac *losowe*, *lokalne* lub *celowe* trajektorie przejścia przez przestrzeń po rozwiązaniach

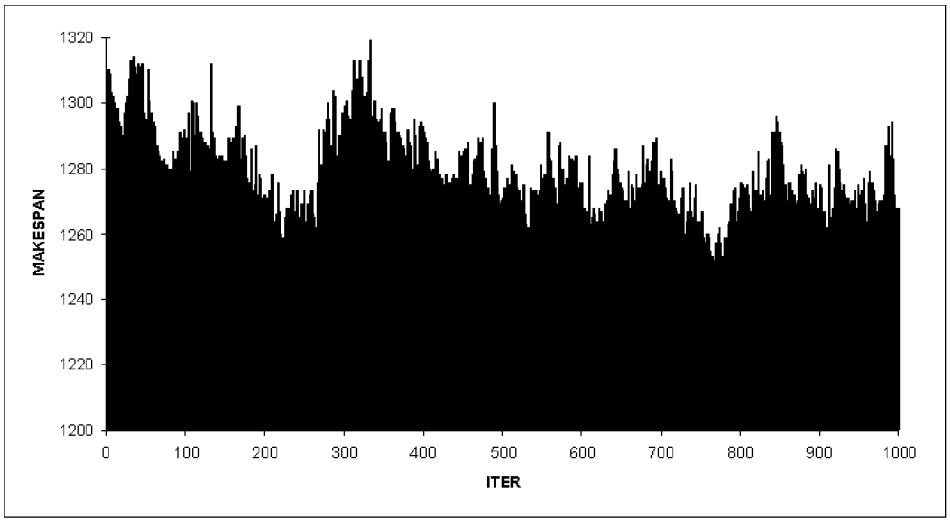
Tabela 8.1: Miary odległości $D(\alpha, \beta)$ pomiędzy permutacjami α oraz β w przestrzeni permutacji

ruch	API	NPI	INS
miara	$D^A(\alpha, \beta)$	$D^S(\alpha, \beta)$	$D^I(\alpha, \beta)$
przepis	liczba inwersji w $\alpha^{-1} \circ \beta$	n minus liczba cykli w $\alpha^{-1} \circ \beta$	n minus długość maksymalnego podciągu rosnącego w $\alpha^{-1} \circ \beta$
średnia	$n(n-1)/4$	$n - H_n$	$n - 2\sqrt{n}$ *)
wariancja	$n(n-1)(2n+5)/72$	$H_n - H_n^{(2)}$	$\theta(n^{1/3})$
złożoność	$O(n^2)$	$O(n^2)$	$O(n \log n)$

*) – asymptotycznie $H_n = \sum_{i=1}^{n} \frac{1}{i}$ $H_n^{(2)} = \sum_{i=1}^{n} \frac{1}{i^2}$

sąsiednich (odległych o jedną jednostkę w sensie miary odległości), patrz Rys. 8.4. Zwykle do testowania przestrzeni i krajobrazu używa się losowego jej próbkowania.

Rozważmy instancję ta45 gniazdowego problemu szeregowania z kryterium długości uszeregowania. Losowe próbkowanie ma co najmniej dwa cele: (1) identyfikacja regionów zawierających rozwiązania dopuszczalne, (2) identyfikacja najbardziej obiecujących obszarów w sensie kryterium $K(x)$. Odległość dowolnego rozwiązania do optymalnego rozkłada się normalnie z wartością średnią 50% średnicy przestrzeni, patrz Rys. 8.5. (Przez średnicę przestrzeni rozumie się maksymalną odległość pomiędzy dowolnymi dwoma rozwiązaniami.) Rozwiązania dopuszczalne są położone znacznie bliżej, w odległości około 20% średnicy przestrzeni. Rozkład dopuszczalnych rozwiązań w sensie kryterium $K(x)$ pokazano w Rys. 8.6. Rozkład jest także bliski normalnemu. Aproksymacja metodą najmniejszych kwadratów daje średnią 115.8 i odchylenie standardowe 17.8, co oznacza że rozwiązanie losowe jest więcej niż dwa razy gorsze od optymalnego. Interesujące, że prawdopodobieństwo znalezienia rozwiązania z $RE \leqslant 1\%$ poprzez losowe próbkowanie przestrzeni $\mathcal{X}$ jest w przybliżeniu $5 \cdot 10^{-11}$, więc praktycznie nieskończenie małe, nawet jeśli rozważymy 10^6 rozwiązań. Z drugiej strony w $\mathcal{X}$ istnieje około $2 \cdot 10^{39}$ rozwiązań z $RE \leqslant 1\%$. Nie ma sposobu na przeglądnięcie ich nawet częściowo. Jest to poważna wada metod poszukiwań losowych.



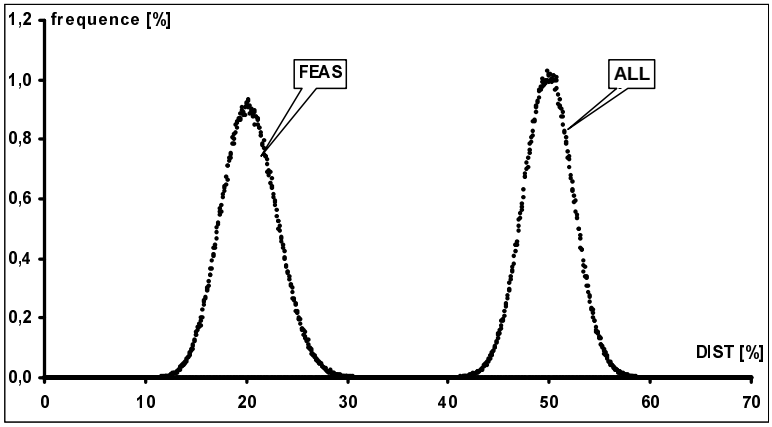
Rysunek 8.4: Wartości $K(x)$ na lokalnej trajektorii dla hybrydowego problemu szeregowania; odleglość pomiędzy kolejnymi rozwiązaniami jest równa jednej jednostce

Duża dolina

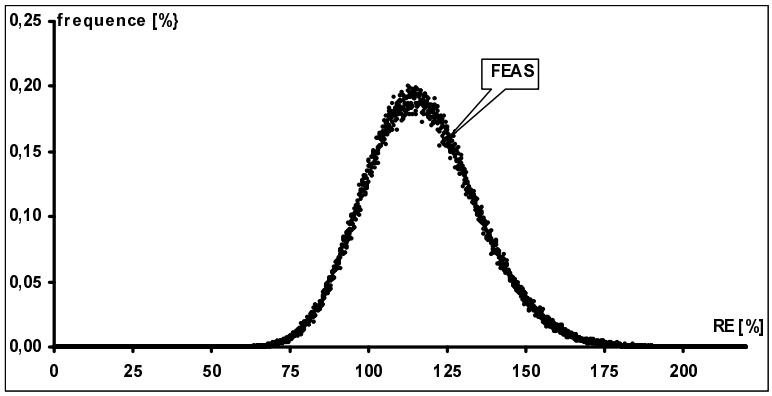
Problem jest podejrzany o posiadanie fenomenu *dużej doliny* jeśli istnieje dodatnia korelacja pomiędzy wartością funkcji celu a odległością do rozwiązania optymalnego (najlepszego znanego rozwiązania). W wielkiej dolinie występuje zagęszczenie ekstremów lokalnych, zatem proces poszukiwań powinien być tam kierowany. Duża dolina została wykryta w wielu problemach dyskretnych, by wspomnieć TSP, szeregowanie, itp. Najnowsze prace sugerują, że duża dolina może być relatywnie mała w odniesieniu do całej przestrzeni rozwiązań.

Szorstkość

Szorstkość jest miarą charakteryzującą rozrzut wartości funkcji celu pokrewnych (zwykle sąsiednich) rozwiązań. Większa szorstkość oznacza ostre, gwałtowne i nieprzewidywalne zmiany $K(x)$ dla rozwiazań sąsiednich, porównaj z Rys. 8.4. Mniejsza szorstkość oznacza płaski lub wolno-zmienny krajobraz. Dla odległości $d(x, y)$, $x, y \in \mathcal{X}$ (istnieje wiele możliwych definicji miary odległości) proponuje się miarę szorstkości jako współczynnik



Rysunek 8.5: Rozkład odległości Hamminga DIST wszystkich (ALL) i dopuszczalnych (FEAS) rozwiazań x^*; próbka 500,000 losowych rozwiązań dla instancji ta45 gniazdowego problemu szeregowania



Rysunek 8.6: Rozkład błędu względnego RE dla rozwiązań dopuszczalnych; próbka 500,000 losowych rozwiązań dla instancji ta45 gniazdowego problemu szeregowania

autokorelacji,

$$\rho(d) = 1 - \frac{AVE((K(x) - K(y))^2)_{d(x,y)=d}}{AVE((K(x) - K(y))^2)} \tag{8.5}$$

gdzie $AVE((K(x) - K(y))^2)$ jest średnia wartością różnicy $(K(x) - K(y))^2$ na zbiorze par rozwiązań (x, y), $x, y \in \mathcal{X}$, zaś $AVE((K(x) - K(y))^2)_{d(x,y)=d}$ jest analogiczną wartością średnią różnicy $(K(x) - K(y))^2$ wyznaczona dla zbioru par rozwiązań (x, y), $x, y \in \mathcal{X}$ takich, że odległość z x do y równa się dokładnie d. Wartość $\rho(d)$ definuje korelację pomiędzy rozwiązaniami $x, y \in \mathcal{X}$ odległymi wzajemnie o d od siebie. Biorąc pod uwagę specjalne właściwości algorytmów poszukiwań lokalnych, najbardziej interesujaca jest wartość $\rho(1)$, będącą korelacja pomiędzy rozwiązaniami położonymi w pojedynczym sąsiedztwie. Wartość $\rho(1)$ bliska zeru oznacza brak korelacji, co implikuje silne zróżnicowanie $K(x)$ w sąsiedztwie (krajobraz szorstki); wartość $\rho(1)$ bliska jeden oznacza silna korelację oraz płaski lub wolno-zmienny krajobraz. Ze względu na liczność $\mathcal{X}$, znalezienie odpowiednich wartości średnich jest kłopotliwe lub nawet niemożliwe. Dlatego w praktyce korzystamy z próbkowania losowego i funkcji autokorelacji zdefiniowanej nastepujaco

$$r(s) = 1 - \frac{AVE((K(x_i) - K(x_{i-s}))^2)}{2(AVE(K^2) - (AVE(K))^2)} \tag{8.6}$$

gdzie $(x_1, x_2, ..., x_k)$ jest trajektorią (ciągiem rozwiazań) generowanych przez algorytm poszukiwań losowych. Na bazie $r(1)$ definiujemy współczynnik autokorelacji ξ jako $\xi = 1/(1-r(1))$. Zgodnie z poprzednimi uwagami, większa wartość ξ oznacza płaski krajobraz.

8.3 Metody dokładne

Metoda dokładna wyznacza rozwiązanie globalnie optymalne tzn. $x^* \in \mathcal{X}$ takie, że

$$K^* \stackrel{\text{def}}{=} K(x^*) = \min_{x \in \mathcal{X}} K(x). \tag{8.7}$$

W grupie metod dokładnych, w zależoności od przynależności problemu do klasy złożoności obliczeniowej, są: (a) efektywne algorytmy dedykowane, (b) metody oparte o schemat podziału i ograniczeń (B&B), (c) metody oparte o schemat programowania dynamicznego (PD), (d) metody oparte na programowaniu liniowym całkowitoliczbowym (PLC), (e) metody oparte na programowaniu liniowym binarnym (PLB), (f) metody subgradientowe. Metody (a) są uważane za tanie obliczeniowo metody specjalizowane dla problemów

należących do P-klasy lub NP-trudnych problemów liczbowych. Metody (b)–(f) są kosztownymi obliczeniowo metodami polecanymi dla rozwiązywania problemów silnie NP-trudnych. Mimo iż dokonano znacznego postępu w rozwoju tych ostatnich metod, praktycy wciąż uważają je za nieatrakcyjne bądź też ograniczają ich zastosowaniue do bardzo wąskiego zakresu. Metody są czaso- i pamięcio-chłonne, zaś rozmiar problemów które można rozwiązać w rozsądnym czasie jest ciągle zbyt mały. Co więcej implementacja odpowiednich algorytmów wymaga dużego doświadczenia programistycznego. Poważnym problemem jest także wiarygodność danych wejściowych, które często ulegają zaburzeniu tuż po kosztownym wyznaczeniu rozwiązania optymalnego oraz tzw. nadmierna sztywność problemu. Można powiedzieć, że koszt poszukiwania rozwiązania optymalnego jest jeszcze zbyt duży w porównaniu z zyskami otrzymanymi z wdrożenie otrzymanego rozwiązania. Z drugiej strony istnieje grupa problemów dla których zastosowanie metod dokładnych jest pełni uzasadnione a nawet wskazane. Przykładowo poszukiwania minimalnego czasu cyklu dla powtarzalnego procesu produkcyjnego z niewielkim repertuarem wyrobów zwielokrotnia nawet niewielkie zyski otrzymane w jednym cyklu mnożąc je przez liczbę wykonanych cykli.

8.3.1 Efektywne algorytmy dedykowane

Dla wybranych problemów z P-klasy istnieją dedykowane algorytmy o wielomianowej złożoności obliczeniowej, może być ich kilka dla tego samego problemu. Za *algorytm optymalny* dla ustalonego problemu przyjmuje się ten o najmniejszej możliwej teoretycznie złożoności obliczeniowej.

Sam fakt istnienia algorytmu wielomianowego nie przesądza o praktycznej jego przydatności. Jeżeli stopień wielomianu złożoności obliczeniowej jest zbyt wysoki, sensownym wydaję się rozważenie zastosowania algorytmu przybliżonego w celu redukcji kosztu obliczeń.

Nie istnieje żaden związek pomiędzy licznością zbioru $\mathcal{X}$ oraz złożonością obliczeniową problemu. Problemy mające wykładniczą wielkość $|\mathcal{X}|$ moga mieć algorytm wielomianowy, odwrotnie niekiedy stwierdzenie niepustości $\mathcal{X}$ może być problemem NP-trudnym.

8.3.2 Schemat podziału i ograniczeń (B&B)

Schemat (B&B, Branch-and-Bound, B-and-B) nie określa żadnego konkretnego algorytmu lecz ogólne podejście oparte na dekompozycji i "inteligentnym" przeszukiwaniu zbioru rozwiązań dopuszczalnych problemu optymalizacyjnego. Istnieje wiele algorytmów opartych na tym schemacie. Jego

zastosowanie jest całkowicie uzasadnione tylko wtedy gdy uzyskamy pewność, że rozważany problem jest silnie NP-trudny. Schemat B&B dostarcza algorytmów wykładniczych (czas działania algorytmu jest funkcją wykładniczą od rozmiaru rozwiązywanego problemu). Może być stosowany dla dowolnego problemu dyskretnego z nieliniową badź liniową funkcją celu i takimi też ograniczeniami.

Ogólny schemat B&B

Dla dowolnego zadania optymalizacji zachodzi

$$\min_{x\in\mathcal{X}} K(x) = \min\{\min_{x\in\mathcal{X}_1} K(x), \ldots, \min_{x\in\mathcal{X}_s} K(x)\} \tag{8.8}$$

gdzie $\mathcal{X}_j$, $j \in \mathcal{S}$ jest rozbiciem zbioru $\mathcal{X}$ na podzbiory parami rozłączne i wyczerpujące, tzn.

$$\mathcal{X}_i \cap \mathcal{X}_j = \emptyset, \;\; i,j \in \mathcal{S}, \;\; i \neq j, \tag{8.9}$$

$$\bigcup_{j\in\mathcal{S}} \mathcal{X}_j = \mathcal{X}. \tag{8.10}$$

Zatem rozwiązanie problemu (8.7) można otrzymać poprzez rozwiązanie zbioru podproblemów postaci

$$\min_{x\in\mathcal{X}_j} K(x) \tag{8.11}$$

dla $j \in \mathcal{S}$, a następnie używając zależności (8.8). Pełny zbiór podproblemów jest scharakteryzowany poprzez funkcję celu $K(x)$ oraz rozbicie $\mathcal{P} = \{\mathcal{X}_j : \; j \in \mathcal{S}\}$. Odpowiednie podproblemy mogą być rozwiązywane szeregowo (system jednoprocesorowy) lub równolegle (system wieloprocesorowy). Podana dekompozycja jest racjonalna jeśli uzyskanie rozwiązania dla (8.11) jest istotnie łatwiejsze niż dla (8.7). Problem (8.11) dla wybranego ustalonego $j \in \mathcal{S}$ można rozwiązać stosując sekwencję następujących podejść: (1) relaksację, (2) bezpośrednio, (3) pośrednio, (4) dokonując podziału. Omówimy te techniki kolejno.

Relaksacja oznacza usunięcie lub osłabienie; może odnosić się do ograniczeń i/lub funkcji celu. *Relaksacja ograniczeń* to usunięcie lub złagodzenie części (lub całości) ograniczeń wyznaczających zbiór rozwiązań dopuszczalnych $\mathcal{X}_j$. W sposób oczywisty zachodzi $\mathcal{X}_j \subseteq \mathcal{X}_j^R$, gdzie $\mathcal{X}_j^R$ jest zbiorem rozwiązań dopuszczalnych problemu zrelaksowanego $\min_{x\in\mathcal{X}_j^R} K(x)$ oraz

$$\min_{x\in\mathcal{X}_j} K(x) \geqslant \min_{x\in\mathcal{X}_j^R} K(x) = K(x^{*R}) \stackrel{\text{def}}{=} LB(\mathcal{X}_j). \tag{8.12}$$

Jeżeli dla wyznaczonego rozwiązania optymalnego x^{*R} problemu zrelaksowanego zachodzi $x^{*R} \in \mathcal{X}_j$ to otrzymane rozwiązane jest także rozwiązaniem optymalnym problemu bez relaksacji (8.11). W przeciwnym przypadku problem (8.11) nie został rozwiązany, zaś wielkość $K(x^{*R})$ stanowi *dolne ograniczenie* $LB(\mathcal{X}_j)$ wartośći funkcji celu dla wszystkich rozwiązań problemu (8.11). *Relaksacja funkcji celu* to zastąpienie funkcji celu $K(X)$ funkcją $K'(x)$ taką, że $K'(x) \leqslant K(x)$ dla wszystkich $x \in \mathcal{X}$. W sposób oczywisty zachodzi

$$\min_{x \in \mathcal{X}_j} K(x) \geqslant \min_{x \in \mathcal{X}_j} K'(x) = K'(x') \stackrel{\text{def}}{=} LB'(\mathcal{X}_j). \tag{8.13}$$

Jeżeli dla wyznaczonego rozwiązania optymalnego x' problemu zrelaksowanego $\min_{x \in \mathcal{X}_j} K'(x)$ zachodzi $K(x') = K'(x')$ to otrzymane rozwiązane x' jest także rozwiązaniem optymalnym problemu bez relaksacji (8.11). W przeciwnym przypadku problem (8.11) nie został rozwiązany, zaś wielkość $K(x^{*R})$ stanowi *dolne ograniczenie* $LB'(\mathcal{X}_j)$ wartości funkcji celu dla wszystkich rozwiązań problemu (8.11). Postać relaksacji jest w zasadzie dowolna chociaż zakłada się domyślnie, że rozwiązanie problemu zrelaksowanego powinno być istotnie łatwiejsze niż problemu (8.11), np. problem zrelaksowany należy do P-klasy. W praktyce relaksacja jest silnie specyficzna dla każdego rozważanego problemu.

Bezpośrednio został rozwiązny problem jeśli obliczyliśmy explicite wartości funkcji celu $K(X)$ dla wszystkich $x \in \mathcal{X}_j$ w celu wyboru wartości minimalnej przy założeniu, że $\mathcal{X}_j$ jest niewielkiej liczności. W praktyce metoda ta jest stosowana tylko jeśli zbiór $\mathcal{X}_j$ jest jedno-elementowy z elementem podanym jawnie.

Pośrednio rozwiązany został problem (8.11) dla którego stwierdzono zachodzenie warunku

$$LB(\mathcal{X}_j) \geqslant UB, \tag{8.14}$$

gdzie UB jest pewnym *górnym ograniczeniem* wartości funkcji celu dla wszystkich rozwiązań problemu (8.7). Problem taki faktycznie nie jest rozwiązywany lecz *eliminowany* z rozważań jako nie zawierający rozwiązań o wartości funkcji celu mniejszej niż podana wartość progowa UB. Dla szeregu problemów szczególnych możliwe jest uzyskanie specyficznych *reguł eliminacji*, które mimo iż nie zawsze są przedstawiane w formie warunku (8.14) mogą być do niego sprowadzone.

Podział stosujemy do problemu, którego nie można rozwiązać używając podejść (a)–(c). Problem (8.11) odpowiadający wybranemu $\mathcal{X}_j$ jest usuwany ze zbioru $\mathcal{P}$, zaś na jego miejsce dodawane są podproblemy otrzymane przez *podział* $\mathcal{X}_j$ na pewną liczbę problemów potomnych skojarzonych ze zbiorami

rozwiązań $\mathcal{Y}_k$, $k = 1, \ldots, r$, takimi że $\mathcal{Y}_k$, $k = 1, \ldots, r$ jest podziałem zbioru $\mathcal{X}_j$. Proces ten wygodnie jest przedstawić w postaci drzewa rozwiązań, w którym węzłowi drzewa odpowiada pewien zbiór $\mathcal{X}_j$, zaś krawędziom drzewa para podzbiorów $(\mathcal{X}_i, \mathcal{X}_j)$ taka, że X_j został otrzymany poprzez podział $\mathcal{X}_i$.Korzeniowi drzewa odpowiada $\mathcal{X}$. W tak zaprojektowanym algorytmie zbiór $\mathcal{P}$ posiada charakter dynamiczny.Analiza kolejnych kolejnych węzłów odpowiadających $\mathcal{P}$ powoduje zamykanie węzłów lub dalsze rozgałęzianie drzewa.

Cały schemat B&B wygodnie jest przedstawić w formie następującego ogólnego algorytmu.

Ogólny algorytm B&B

Krok 0. (*(inicjalizacja*) Podstaw $\mathcal{P} = \{\mathcal{X}\}$ oraz $UB = \infty$.

Krok 1. (*powrót*) Jeżeli $\mathcal{P} = \emptyset$ then STOP; x^* jest rozwiązaniem optymalnym zaś $K(x^*) = UB$.

Krok 2. (*wybór*) Wybierz zbiór $\mathcal{X}_j \in \mathcal{P}$. Podstaw $\mathcal{P} := \mathcal{P} \setminus \mathcal{X}_j$.

Krok 3. (*relaksacja*) Rozwiąż problem zrelaksowany $\min_{x \in \mathcal{X}_j^R} K(x) = K(x^{*R})$. Jeżeli $x^{*R} \in \mathcal{X}_j$ to przejdź do kroku 6 inaczej podstaw $LB(\mathcal{X}_j) := K(x^{*R})$ i przejdź do kroku 5.

Krok 4. (*eliminacja*) Jeżeli $LB(\mathcal{X}_j) \geqslant UB$ przejdź do kroku 1.

Krok 5. (*podział*) Podziel $\mathcal{X}_j$ oraz podstaw $\mathcal{P} := \mathcal{P} \cup \bigcup_{k=1}^{r} \mathcal{Y}_k$ gdzie $\mathcal{Y}_k$, $k = 1, \ldots, r$ jest podziałem $\mathcal{X}_j$. Przejdź do kroku 1.

Krok 6. (*aktualizacja*) Jeżeli $K(x^{*R}) < UB$ to podstaw $UB := K(x^{*R})$ oraz $x^* := x^{*R}$. Przejdź do kroku 1.

Jeżeli $\mathcal{X}$ jest zbiorem skończonym, algorytm zatrzyma się po skończonej liczbie kroków. Każdy algorytm B&B wymaga precyzyjnego określenia następujących elementów: (a) *reguła wyboru* w kroku 2 określająca kolejnośc rozwiązywania podproblemów, (b) przyjęta relaksacja dostarczająca *dolne ograniczenie* w kroku 3, (c) *reguły eliminacji* stosowane w kroku 4, (d) *zasada podziału* w kroku 5, (e) technika dostarczająca *górne ograniczenie* w kroku 6. Algorytm B&B jest tym lepszy im więcej podproblemów z $\mathcal{P}$ jest eliminowanych. Skuteczność eliminacji zależy od wielu współgrających ze sobą czynników (postać relaksacji, dokładność dolnego/górnego ograniczenia wartości funkcji celu, reguła wyboru, itp.).

Specjalizowane warianty B&B

Specjalizowane warianty schematu B&B są zwykle bardziej efektywne dzięki wykorzystaniu szczególnych własności problemu do poprawy własności eliminacyjnych schematu. Konkretne postacie tych własności zależą od rozważanego problemu. Niekiedy stosowane są także drobne modyfikacje podanego schematu ogólnego, jak np. inicjowanie UB wartością $K(x^0)$ dla pewnego $x^0 \in \mathcal{X}$, aktualizacja (krok 6) jest wykonywana w każdej iteracji algorytmu w oparciu o wartość $K(x')$, gdzie $x' \in \mathcal{X}_j$, itp.

8.3.3 Schemat programowania dynamicznego

Schemat ten określa ogólne podejście polegające na przekształceniu zadania optymalizacji w wieloetapowy proces podejmowania decyzji, w którym stan na każdym etapie zależy od decyzji wybieranej ze zbioru decyzji dopuszczalnych. Oznaczając stany procesu na poszczególnych etapach poprzez S_i, $i = 1, 2, \ldots, N$, zaś odpowiednie decyzje porzez d_i, $i = 1, 2, \ldots, N$, to przebieg wieloetapowego procesu decyzyjnego można zapisać w postaci transformacji $S_k = T(S_{k-1}, d_{k-1})$. Z procesem podejmowania decyzji jest związana skalarna funkcja celu $F(S_1, S_2, \ldots, S_N; d_1, d_2, \ldots, d_N)$ służąca do oceny ciągu decyzji $d_1, d_2, \ldots, d_N$ którą należy minimalizować. Jeżeli S_1 jest znany, to przebieg wieloetapowego procesu decyzyjnego jest wyznaczony poprzez ciąg decyzji dopuszczalnych $d_1, d_2, \ldots, d_N$ nazywanych strategią. Strategia minimalizująca funkcję F jest nazywana strategią optymalną.

Podstawowy schemat PD

Wyznaczanie strategii optymalnej w przypadku ogólnym ma postać dość trudnego zadania optymalizacji nieliniowej

$$\min_{d_1, d_2, \ldots, d_N} F(S_1, S_2, \ldots, S_N; d_1, d_2, \ldots, d_N) \tag{8.15}$$

$$S_k = T(S_{k-1}, d_{k-1}), \quad k = 2, \ldots, N, \tag{8.16}$$

gdzie S_1 jest dane. Stąd zysk z zastosowania schematu PD do optymalizacji dyskretnej jest osiągany tylko dla pewnej podklasy problemów decyzyjnych bez pamięci. Są to problemy posiadające tzw. własność Markowa, w których przebieg procesu począwszy od stanu na etapie k-tym dalej nie zależy od jego historii, lecz tylko od S_k. Istnieje stosunkowo duża klasa problemów o znaczeniu technicznym i ekonomicznym posiadających wymienioną własność.

Dla wieloetapowych procesów decyzyjnych z własnością Markowa strategia optymalna ma tą własność, że niezależnie od podanego stanu początkowego i decyzji początkowej pozostałe decyzje muszą stworzyć strategię optymalną z punktu widzenia stanu wynikłego z pierwszej decyzji (zasada optymalności Bellmana). Pozwala to uzyskać rozwiązanie dla pewnych podklas problemów. Przykładowo rozpatrzmy zadanie minimalizacji funkcji addytywnej

$$F(S_1, S_2, \ldots, S_N; d_1, d_2, \ldots, d_N) = \sum_{i=1}^{N} f_i(S_i, d_i). \tag{8.17}$$

Zgodnie z zasadą optymalności mamy

$$\min_{d_1, d_2, \ldots, d_N} (f_1(S_1, d_1) + \ldots + f_N(S_N, d_N)) =$$
$$\min_{d_1}(f_1(S_1, d_1) + \min_{d_2, \ldots, d_N} (f_2(S_2, d_2) \ldots + f_N(S_N, d_N))) \tag{8.18}$$

Oznaczając, dla $k = 1, 2, \ldots, N$

$$q_k(S_k) = \min_{d_k, \ldots, d_N} (f_k(S_k, d_k) + \ldots + f_N(S_N, d_N)) \tag{8.19}$$

otrzymamy

$$q_N(S_N) = \min_{d_N} f_N(S_N, d_N) \tag{8.20}$$

$$q_k(S_k) = \min_{d_k}(f_k(S_k, d_k) + q_{k+1}(S_{k+1})) \tag{8.21}$$

Ostatecznie, możemy zapisać rezultat w postaci *równania rekurencyjnego Bellmana*

$$q_k(S_k) = \min_{d_k}(f_k(S_k, d_k) + q_{k+1}(T(S_k, d_k))), \quad k = N, \ldots, 1, \tag{8.22}$$

gdzie

$$q_{N+1}(S_{N+1}) = 0 \tag{8.23}$$

Decyzję optymalną wyznacza się parametrycznie.

Specjalizowane warianty PD

Schemat PD jest stosowany do rozwiązywania różnych problemów dyskretnych, począwszy od zagadnienia najdłuższej drogi w grafie, poprzez problem załadunku, problem komiwojażera (TSP), aż do problemów szeregowania, w których pełni funkcje zasadnicze (podstawa algorytmu) lub pomocnicze (pomocniczy algorytm składowy). Złożoność obliczeniowa tak otrzymanego algorytmu zależy od konkretnej aplikacji, może być zarówno wielomianowa (wspomniana droga w grafie), psedowielomianowa (problem zaladunku), jak i wykładnicza (TSP).

8.3.4 Programowanie liniowe całkowitoliczbowe

Zadanie programowania liniowego ca lkowitoliczbowego (PLC) odnosi się do problemu (8.7), w którym $K(x)$ jest funcją liniową, zaś $\mathcal{X}$ jest określony zestawem ograniczeń liniowych z dodatkowym żądaniem by wszystkie składowe wektora x przyjmowały wartości całkowite. Jeśli ostatnie wymaganie odnosi się tylko do niektórych składowych wektora x to mówimy o zadaniu PLC mieszanym (PLCM), do którego algorytmy przedstawione w końcowej części rozdziału mogą być również stosowane. Podstawę algorytmów opisanych w tym rozdziale stanowi "klasyczny" problem programowania liniowego ciągłego (PL), omówiony krótko w kolejnym podrozdziale (rozszerzenie tematu można znaleźć między innymi w książkach [102,351,375]).

Programowanie liniowe (PL).

Rozważamy zadanie programowania liniowego dane w tradycyjnej formie (Ponieważ $\min_x K(x) = -max_x(-K(x))$, możemy je zastosować do problemu (8.7))

$$\max_x \; x_0 \stackrel{\text{def}}{=} cx \tag{8.24}$$

$$Ax = b \tag{8.25}$$

$$x \geqslant 0 \tag{8.26}$$

gdzie $c_{1\times n}$, $x_{n\times 1}$, $A_{m\times n}$, $b_{m\times 1}$ są macierzami (wektorami) podanych rozmiarów. Kolumny macierzy A reprezentują wektory w przestrzeni m-wymiarowej. Wektor x nazywamy *rozwiązaniem* zadania PL, zaś rozwiązanie spełniające warunki (8.25)–(8.26) *rozwiązaniem dopuszczalnym*. Dla danego zadania PL może wystąpić jeden z następujących przypadków: (A) nie istnieje żadne rozwiązanie dopuszczalne czyli zadanie jest *sprzeczne*, (B) istnieją rozwiązania x, x' takie, że $cx' > 0$ oraz rozwiązanie $x + \alpha x'$ jest rozwiązaniem dopuszczalnym dla dowolnej liczby $\alpha \geqslant 0$; stąd cx może osiągność dowolnie dużą wartość czyli zadanie jest *nieograniczone*, (C) istnieje jedno rozwiązanie dopuszczalne x^* takie, że $\infty > cx^* \geqslant cx$ dla każdego rozwiązania dopuszczalnego x; x^* jest *rozwiązaniem optymalnym*. W dalszym ciągu będziemy się zajmować wyłącznie przypadkiem (C).

Zauważmy, że zmiana kolejności występowania zmiennych nie może mieć wpływu na rozwiązanie problemu. Dokonajmy zatem przestawienia kolejności występowania składowych wektora x, odpowiednio składowych wektora c oraz kolumn macierzy A tak, by macierz A można było zapisać w formie $A = (B, N)$, gdzie B jest pewną macierzą nieosobliwą $B_{m\times m}$ zaś $N_{m\times(n-m)}$. Macierz B jest nazywana dalej *macierzą bazową* lub *bazą wektorów* zadania

PL. Dalej, oznaczmy odpowiednio $x = (x_B, x_N)$ oraz $c = (c_B, c_N)$, gdzie x_B jest wektorem zmiennych (bazowych) odpowiadających kolumnom macierzy B, zaś x_N jest wektorem zmiennych (niebazowych) odpowiadających kolumnom macierzy N. Ograniczenie (8.25) można zapisać w formie

$$Bx_B + Nx_N = b. \tag{8.27}$$

Ponieważ B jest nieosobliwa, zatem korzystając z rachunku macierzowego otrzymujemy rozwiązanie (8.27) w postaci

$$x_B = B^{-1}b - B^{-1}Nx_N. \tag{8.28}$$

Jednym z rozwiązań zadania (8.24)–(8.26) jest *rozwiązanie bazowe* $x_B = B^{-1}b$, $x_N = 0$. Rozwiązanie bazowe, dla którego zachodzi $x_B \geqslant 0$ jest rozwiązaniem *bazowym dopuszczalnym*. Wykazano, że jeśli zadanie PL ma rozwiązanie optymalne to ma również rozwiązanie bazowe optymalne. Stąd możemy ograniczyć się tylko do badania rozwiązań bazowych. Istnieje co najwyżej $\binom{n}{m}$ różnych rozwiązań bazowych.

Na podstawie (8.24) otrzymujemy

$$x_0 = c_B x_B + c_N x_N, \tag{8.29}$$

zaś podstawiając do (8.29) zależność (8.28) dostajemy

$$x_0 = c_B B^{-1}b - (c_B B^{-1}N - c_N)x_N. \tag{8.30}$$

Zależności (8.30) oraz (8.28) zapisujemy łącznie w postaci macierzowej

$$\begin{bmatrix} x_0 \\ x_B \end{bmatrix} = \begin{bmatrix} c_B B^{-1}b \\ B^{-1}b \end{bmatrix} - \begin{bmatrix} c_B B^{-1}N - c_N \\ B^{-1}N \end{bmatrix} x_N. \tag{8.31}$$

będącej wyjściem do dalszej analizy własności problemu PL.

Oznaczmy przez $B_1, \ldots, B_m$ indeksy zmiennych bazowych, zaś przez R zbiór indeksów zmiennych niebazowych. Stąd $x_B = (x_{B_1}, \ldots, x_{B_m})$. Przyjmijmy również $x_{B_0} \stackrel{\text{def}}{=} x_0$. Oznaczmy dalej

$$y_0 \stackrel{\text{def}}{=} \begin{bmatrix} c_B B^{-1}b \\ B^{-1}b \end{bmatrix} = \begin{bmatrix} y_{00} \\ y_{10} \\ \vdots \\ y_{m0} \end{bmatrix}. \tag{8.32}$$

oraz

$$y_j \stackrel{\text{def}}{=} \begin{bmatrix} c_B B^{-1} a_j - c_j \\ B^{-1} a_j \end{bmatrix} = \begin{bmatrix} y_{0j} \\ y_{1j} \\ \vdots \\ y_{mj} \end{bmatrix}. \tag{8.33}$$

gdzie a_j, $j \in R$ oznaczają kolumny macierzy N. Wzór (8.31) można zapisać teraz w krótszej postaci

$$x_{B_i} = y_{i0} - \sum_{j \in R} y_{ij} x_j, \quad i = 0, 1, \ldots, m. \tag{8.34}$$

Niech będzie dane pewne niezdegenerowane rozwiazanie (rozwiazanie jest niezdegenerowane jeśli $x_{B_i} > 0$, $i = 1, \ldots, m$) bazowe dopuszczalne x_B. Z (8.34) mamy

$$x_0 = x_{B_0} = y_{00} - \sum_{j \in R} y_{0j} x_j. \tag{8.35}$$

Ponieważ x_0 jest funkcją liniową względem x_j, $j \in R$ to zachodzą dwa wykluczające się przypadki: (a) jeśli $y_{0j} \geqslant 0$, $j \in R$ to $x_0 = y_{00}$ jest najwyższą osiąganą wartością funkcji celu (wartością optymalną) oraz oczywiście należy przyjąć $x_j = 0$, $j \in R$, (b) istnieje $k \in R$ takie, że $y_{0k} < 0$; wówczas x_0 rośnie jeśli x_k rośnie. Zajmijmy się bardziej szczegółowo przypadkiem (b). Ponieważ z (8.34) mamy

$$x_{B_i} = y_{i0} - \sum_{j \in R \setminus \{k\}} y_{ij} x_j - y_{ik} x_k, \; i = 1, \ldots, m, \tag{8.36}$$

to wzrost x_k implikuje liniowy spadek wartości tych x_{B_i}, dla których $y_{ik} > 0$. Zatem, jeśli tylko $y_{ik} > 0$ to $x_{B_i} \geqslant 0$ tak długo jak długo $x_k \leqslant y_{i0}/y_{ik} \stackrel{\text{def}}{=} v_{ik}$. Dla wartości granicznej $x_k = v_{ik}$ mamy $x_{B_i} = 0$. Ponieważ spadek wartości może dotyczyć równocześnie kilku zmiennych bazowych x_{B_i}, to istnieje zmienna bazowa x_{B_r}, która jako "pierwsza" osiągnie wartość zero. Jest ona określona zależnością

$$v_{rk} = \min\{v_{ik} : \; y_{ik} > 0, \; i = 1, \ldots, m\}. \tag{8.37}$$

Zatem jeśli $v_{rk} > 0$ to pozostawiając wszystkie zmienne niebazowe $x_j = 0$, $j \in R \setminus \{k\}$ oraz zwiększając wartość zmiennej x_k otrzymamy nowe rozwiązanie bazowe dopuszczalne

$$x_k = v_{rk} \tag{8.38}$$

$$x_{B_i} = y_{i0} - y_{ik} v_{rk}, \; i = 1, \ldots, m, \tag{8.39}$$

dla którego $x_k > 0$, $_{B_r} = 0$, $x_0 = y_{00} - y_{0k}v_{rk}$, przy czym wartość funkcji celu otrzymana tym zabiegiem uległa zwiększeniu. Proces ten można interpretować jako "wymianę" zmiennej bazowej (wektora bazy); wektor B_r jest usuwany z bazy B zaś $k \in R$ jest wprowadzany do bazy.

Szczegółowe wzory na znalezienie nowego rozwiązania bazowego dopuszczalnego można otrzymać poprzez wyznaczenie z równania (8.34) dla $i = r$

$$x_{B_r} = y_{r0} - \sum_{j \in R \setminus \{k\}} y_{rj} x_j - y_{rk} x_k \tag{8.40}$$

wartości x_k

$$x_k = \frac{y_{r0}}{y_{rk}} - \sum_{j \in R \setminus \{k\}} \frac{y_{rj}}{y_{rk}} x_j - \frac{1}{y_{rk}} x_{B_r} \tag{8.41}$$

oraz podstawieniu jej do pozostałych równań (8.34) dla $i = 1, \ldots, m$, $i \neq k$,

$$x_{B_i} = y_{i0} - y_{ik}\frac{y_{r0}}{y_{rk}} - \sum_{j \in R \setminus \{k\}} (y_{ij} - y_{ik}\frac{y_{rj}}{y_{rk}}) x_j + \frac{y_{ik}}{y_{rk}} x_{B_r}. \tag{8.42}$$

Rozwiązanie bazowe x_B jest optymalne jeśli $y_{i0} \geqslant 0$, $i = 1, \ldots, m$ (warunek dopuszczalności) oraz $y_{0j} \geqslant 0$, $j \in R$. Opisana technika poprawiania rozwiazania bazowego jest podstawą algorytmu *sympleks*, jednej z najczęściej używanych metod dla rozwiązywania zadań PL. *Iteracją* sympleksową nazywa się opisany proces wymiany wektora bazy. Modyfikacja tablicy sympleksowej y_{ij} realizowana jest krokowo przy użyciu prostych wzorów transformacyjnych pochodzących z (8.42), bazujących na elemencie centralnym przekształcenia y_{rk}. W praktyce należy kolejno: (1) podzielić wiersz r-ty przez wielkość y_{rk} (bez elementu y_{rk}), (2) od wiersza i-tego odjać wiersz r-ty pomnożony przez y_{ik} (bez kolumny k-tej), $i \neq r$, (3) kolumnę k-tą podzielić przez $-y_{rk}$ (bez elementu y_{rk}), (4) element y_{rk} zastąpić przez $1/y_{rk}$, tzn.

$$y'_{rj} = \frac{y_{rj}}{y_{rk}}, \quad j = 0, \ldots, n, \tag{8.43}$$

$$y'_{ij} = y_{ij} - y'_{rj} y_{ik}, \quad j = 0, \ldots, n, \quad i = 0, \ldots, r-1, r+1, \ldots, m. \tag{8.44}$$

Algorytm ten przy założeniu, że wszystkie rozwiązania bazowe są niezdegenerowane, wyznacza rozwiązanie optymalne po co najwyżej $\binom{n}{m}$ iteracjach.

Opis podany powyżej nie precyzuje wszystkich szczegółów algorytmu. Jeśli więcej niż jedna zmienna niebazowa spełnia warunek $y_{0j} < 0$ (może być wprowadzona do bazy) to typowo wybieramy $y_{0k} = \min_{j \in R} y_{0j}$. Jeśli dla wyznaczonego v_{rk} istnieje kilka wartości $v_{ik} = v_{rk}$, to zmienną usuwaną z bazy wybieramy arbitralnie wśród nich. Jeśli zadanie PL ma rozwiązanie

nieograniczone to istnieją rozwiązanie bazowe dopuszczalne x_B oraz wektor y_k taki, że $y_{0k} < 0$ i $y_{ik} < 0$, $i = 1, \ldots, m$.

Do rozpoczęcia algorytmu potrzebne jest rozwiązanie bazowe dopuszczalne. Wykazano, że zadanie PL ma rozwiązanie dopuszczalne wtedy i tylko wtedy gdy ma rozwiązanie bazowe dopuszczalne. Dla niektórych klas problemów struktura macierzy A pozwala łatwo znaleźć (zgadnąć) rozwiązanie bazowe, najczęściej wtedy gdy w A można wyodrębnić podmacierz jednostkową $I_{m\times m}$ przestawiając odpowiednio kolumny. Alternatywnie, jeśli ograniczenia problemu wystąpiły w postaci $Ax \leqslant b > 0$, to w celu doprowadzenia do postaci równościowej (8.25) należy dodać tzw. zmienne osłabiające, które powodują naturalne uzupełnienie macierzy A macierzą jednostkową. W innych przypadkach stosowana jest *metoda dwóch faz*. W fazie pierwszej wprowadza się *sztuczną bazę* w formie macierzy jednostkowej $I_{m\times m}$ oraz wektor zmiennych sztucznych $s_{m\times 1}$. Wyjściowe zadanie (8.24)–(8.26) zastępuje się zadaniem postaci

$$\max_x \; z_0 \stackrel{\text{def}}{=} -\sum_{i=1}^{m} s_i \tag{8.45}$$

$$x_0 - cx = 0 \tag{8.46}$$

$$Ax + Is = b \tag{8.47}$$

$$x, s \geqslant 0 \tag{8.48}$$

Następnie eliminuje się sztuczne zmienne bazowe zastępując je zmiennymi niebazowymi z A. Faza ta pozwala także na ocenę czy zadania PL ma rozwiązanie, jeśli bowiem po zakończeniu fazy pierwszej wartość z_0 jest ujemna to zadanie PL jest sprzeczne. Każde rozwiązanie dopuszczalne zadania (8.45)–(8.46) z $z_0 = 0$ daje rozwiazanie bazowe dopuszczalne zadania (8.24)–(8.26). W praktyce rozmiar sztucznej bazy musi być co najwyżej m bowiem część wektorów bazowych może pochodzić z A. Faza ta kończy się po wyeliminowaniu wszystkich zmiennych sztucznych. W fazie drugiej realizowany jest opisany już proces minimalizacji x_0.

W metodach PLC opartych na podziale zbioru rozwiązań (patrz Algorytm Land-Doig'a opisany dalej) często zachodzi potrzeba rozwiązywania zadania PL z ograniczeniem zakresu zmiennych

$$l_j \leqslant x_j \leqslant u_j, \;\; j = 1, \ldots, n. \tag{8.49}$$

Choć ograniczenia (8.49) można włączyć do macierzy A, to jednak postępowanie takie jest nieefektywne. Lepiej dokonać modyfikacji algorytmu podstawowego w celu uwzględnienia warunku (8.49). I tak, ograniczenie $l_j \leqslant x_j$

można usunąć przez podstawienie $x'_j = x_j - l_j$ prowadzące do nowego zadania PL z ograniczeniem $0 \leqslant x'_j$. Zatem, bez straty ogólności rozważań, możemy dalej założyć $l_j = 0$. Rozwiązanie zadania PL (8.24)–(8.25), (8.49) nazywamy bazowym jeśli wszystkie zmienne niebazowe są równe 0 lub swoim u_j. Zatem niech $R^0 = \{j \in R : \ x_j = 0\}$ oraz $R^+ = \{j \in R : \ x_j = u_j\}$. Zależność (8.34) można teraz zapisać w formie

$$x_{B_i} = y_{i0} - \sum_{j \in R^0} y_{ij} x_j - \sum_{j \in R^+} y_{ij} x_j, \quad i = 0, 1, \ldots, m. \tag{8.50}$$

Oznaczmy przez $z_{i0} = x_{B_i}$ wartość tej zmiennej dla $x_j = 0$, $j \in R^0$ oraz $x_j = u_j$, $j \in R^+$. Analizując analogiczny wzór na x_0 (patrz (8.35)) możemy stwierdzić, że wzrost wartości funkcji celu możemy osiągnąć w dwóch przypadkach: (1) zmniejszając x_j dla pewnego $j \in R^+$, dla którego $y_{0j} > 0$, (2) zwiększając x_j dla pewnego $j \in R^0$, dla którego $y_{0j} < 0$. Stąd mamy warunki dostateczne optymalności w formie: (a) $0 \leqslant x_{B_i} \leqslant u_{B_i}$, $i = 1, \ldots, m$ (dopuszczalność), (b) $y_{0j} \geqslant 0$, $j \in R^0$, (c) $y_{0j} \leqslant 0$, $j \in R^+$. Wybór zmiennej k wchodzacej do bazy opiera się na warunku $y_{0k} = \min\{min_{j \in R^+}(-y_{0j}), \min_{j \in R^0} y_{0j}\}$. Wybór zmiennej opuszczającej bazę zależy od wyboru k. Rozpatrzy odpowiednie przypadki oddzielnie.

Niech $k \in R^0$ będzie zmienna wprowadzaną do bazy. Wzrostowi x_k towarzyszy zmiana wartości zmiennych bazowych przy czym w zależności od współczynnika y_{ik} niektóre z nich maleją do zera podczas gdy pozostałe rosną w kierunku swego kresu górnego. Zmienną x_{B_r}, która się pierwsza wyzeruje znajdziemy z warunku $v_{rk} = \min\{z_{i0}/y_{ik} : \ y_{ik} > 0, \ i = 1, \ldots, m\}$. Zmienną x_{B_s}, która się pierwsza nasyci otrzymamy z warunku $w_{sk} = \min\{(z_{i0} - u_{B_i})/y_{ik} : \ y_{ik} < 0 \ \ i = 1, \ldots, m\}$. Uwzględniając ograniczenie zakresu, zmienna x_k może wzrosnąć do wartości co najwyżej $\min\{v_{rk}, w_{sk}, u_k\}$. Jeśli minimum jest osiągane dla v_{rk} lub w_{sk} to odpowiednio zmienna x_{B_r} lub x_{B_s} jest usuwana z bazy. Jeśli minimum wypada na wartości u_k to baza pozostaje niezmieniona lecz zmieniają się wartości x_{B_i} na skutek zmiany x_k; dodatkowo zmienna k przechodzi z R^0 do R^+.

Odmiennie, niech $k \in R^+$ będzie zmienną, której wartość bedziemy zmniejszać. Zmianie tej towarzyszy zmiana wartości zmiennych bazowych o podobnym jak poprzednio charakterze. Zmienną x_{B_r}, która się pierwsza wyzeruje znajdziemy z warunku $\overline{v}_{rk} = \max\{z_{i0}/y_{ik} : \ y_{ik} < 0, \ i = 1, \ldots, m\}$. Zmienną x_{B_s}, która się pierwsza nasyci otrzymamy z warunku $\overline{w}_{sk} = \max\{(z_{i0} - u_{B_i})/y_{ik} : \ y_{ik} > 0\}$. Uwzględniając ograniczenie zakresu, zmienna x_k może zmaleć o co najwyżej $\max\{\overline{v}_{rk}, \overline{w}_{sk}, -u_k\}$. Jeśli maksimum jest osiągane dla $\overline{v}_{rk}$ lub $\overline{w}_{sk}$ to odpowiednio zmienna x_{B_r} lub x_{B_s}

jest usuwana z bazy. Jeśli maksimum wypada na wartości $-u_k$ to baza pozostaje niezmieniona lecz zmieniają się wartości x_{B_i} na skutek zmiany x_k; dodatkowo zmienna k przechodzi z R^+ do R^0.

Rozwiązywanie dużych zadań PL wymaga specyficznych zabiegów w celu operowania macierzą bazową o dużym rozmiarze oraz w celu zagwarantowania stabilności numerycznej obliczeń. Wykorzystuje się w tym celu między innymi *postać iloczynową macierzy odwrotnej* nadającą się szczególnie do macierzy rzadkich, *faktoryzację macierzy* trójkątną (rozkład LU) i ortogonalną (rozkład LQ), strukturalne własności macierzy ograniczeń (struktura blokowo-diagolnalna), metodę generacji kolumn (dla $m \ll n$), patrz na przykład przegląd w pracy [351].

Doświadczenia praktyczne wskazują, że zwykle liczba iteracji sympleksowych jest nie większa niż $m^2 n$, choć złożoność obliczeniowa metody jest wykładnicza. Oprócz metody sympleksów zostało podanych szereg innych podejść, bardziej efektywnych dla szczególnych postaci zadań PL, np. transportowego, oraz modyfikacji poprawiających szybkość zbieżności algorytmu. W latach 70-tych podano wielomianowy algorytm elipsoidalny dla rozwiązywania zadań PL. Algorytm ten mimo interesujących własności teoretycznych nie stał się konkurencyjny ze względu na znaczną złożoność obliczeniową; objawia on swoją przewagę nad metodą sympleks dopiero dla $m > 1,000$ oraz $n > 50,000$.

Na koniec rozpatrzmy przykład praktyczny problemu PL dany pierwotnie w formie nierównościowej

$$\max(3x_1 + 2x_2) \tag{8.51}$$

$$\begin{array}{rcrcr} 2x_1 & + & 3x_2 & \leqslant & 24 \\ 14x_1 & + & 6x_2 & \leqslant & 84 \\ x_1 & & & \leqslant & 5 \\ & & x_2 & \leqslant & 7 \end{array} \tag{8.52}$$

$$x_1, x_2 \geqslant 0. \tag{8.53}$$

W podanym przypadku 2D, ograniczenia (8.52)-(8.53) określają geometrycznie obszar wielokąta. Warstwica $3x_1 + 2x_2 = const$ przesunięta do punktu wierzchołkowego $(3\frac{3}{5}, 5\frac{3}{5})$ dostarcza rozwiązania optymalnego o wartości $x_0 = 22$. Zadanie (8.51)-(8.53) przekształcone do postaci równościowej porzez dodanie zmiennych osłabiających

$$\max(3x_1 + 2x_2) \tag{8.54}$$

B	y_0	y_1	y_2	y_3	y_4	y_5	y_6
0	0	-3	-2	0	0	0	0
3	24	2	3	1	0	0	0
4	84	14	6	0	1	0	0
5	5	1	0	0	0	1	0
6	7	0	1	0	0	0	1

Tabela 8.2: Tablica sympleksowa: iteracja pierwsza

B	y_0	y_1	y_2	y_3	y_4	y_5	y_6
0	15	3	-2	0	0	0	0
3	14	-2	3	1	0	-2	0
4	14	-14	6	0	0	-14	0
1	5	1	0	0	1	1	0
6	7	0	1	0	0	0	1

Tabela 8.3: Tablica sympleksowa: iteracja druga

$$\begin{array}{rcrcrcrcrcl} 2x_1 & + & 3x_2 & + & x_3 & & & & & = & 24 \\ 14x_1 & + & 6x_2 & + & & x_4 & & & & = & 84 \\ x_1 & + & & & & & x_5 & & & = & 5 \\ & & x_2 & + & & & & x_6 & = & & 7 \end{array} \tag{8.55}$$

$$x_1, x_2, x_3, x_4, x_5, x_6 \geqslant 0. \tag{8.56}$$

pozwala na określenie pierwszej bazy $B = (B_3, B_4, B_5, B_6)$ zawierającej wektory odpowiadające zmiennym $x_B = (x_3, x_4, x_5, x_6)$. Stąd otrzymujemy $c_B = (0, 0, 0, 0)$, $c_N = (3, 2)$. Wyjściowa tablica sympleskowa y_{ij} ma postać przedstawioną w Tabl. 8.2. Zgodnie z podanymi własnościami wektorem wprowadzanym do bazy będzie $k = 1$ ($y_{10} = \min\{-3, -2\} < 0$) zaś usuwanym z bazy będzie $r = 5$ ($\frac{y_{50}}{y_5} = \min\{\frac{24}{2}, \frac{84}{14}, \frac{5}{1}\}$). Prowadzi to do tablicy sympleksowej 8.3. Dalej wektorem wprowadzanym do bazy będzie $k = 2$ ($y_{20} = \min\{-2\} < 0$) zaś usuwanym z bazy będzie $r = 4$ ($\frac{y_{40}}{y_{42}} = \min\{\frac{14}{3}, \frac{14}{6}, \frac{7}{1}\}$). Prowadzi to do tablicy sympleksowej 8.4. Wykonanie jeszcze jednej iteracji sympleksowej prowadzi do rozwiązania optymalnego umieszczonego w tab. 8.5.

B	y_0	y_1	y_2	y_3	y_4	y_5	y_6
0	$19\frac{2}{3}$	$-1\frac{2}{3}$	$\frac{1}{3}$	0	$\frac{1}{3}$	$-1\frac{2}{3}$	0
3	7	5	$-\frac{1}{2}$	1	$-\frac{1}{2}$	5	0
2	$2\frac{1}{3}$	$-2\frac{1}{3}$	$\frac{1}{6}$	0	$\frac{1}{6}$	$2\frac{1}{3}$	0
1	5	1	0	0	0	1	0
6	$4\frac{2}{3}$	$2\frac{1}{3}$	$-\frac{1}{6}$	0	$-\frac{1}{6}$	$2\frac{1}{3}$	1

Tabela 8.4: Tablica sympleksowa: iteracja trzecia

B	y_0	y_1	y_2	y_3	y_4	y_5	y_6
0	22	0	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{3}$	0
5	$1\frac{2}{5}$	1	$-\frac{1}{10}$	$\frac{1}{5}$	$-\frac{1}{10}$	$\frac{1}{5}$	0
2	$5\frac{3}{5}$	0	$-\frac{1}{15}$	$\frac{7}{15}$	$-\frac{1}{15}$	$\frac{7}{15}$	0
1	$3\frac{3}{5}$	0	$\frac{1}{10}$	$-\frac{1}{5}$	$\frac{1}{10}$	$-\frac{1}{15}$	0
6	$1\frac{2}{5}$	0	$\frac{1}{15}$	$-\frac{7}{15}$	$\frac{1}{15}$	$-\frac{7}{15}$	1

Tabela 8.5: Tablica sympleksowa: iteracja czwarta

Dualność w PL

Przyjmując, że dane są $c_{1\times n}$, $A_{m\times n}$, $b_{m\times 1}$ można sformułować dwa zadania PL: *pierwotne* nazywane także *prymalnym* (PL-P)

$$\min_x cx, \quad Ax \geqslant b, \quad x \geqslant 0. \tag{8.57}$$

oraz w pełni "symetryczne" do niego zadanie *dualne* (PL-D)

$$\max_u ub, \quad uA \leqslant c, \quad u \geqslant 0. \tag{8.58}$$

gdzie $u_{1\times m}$. Symetria zadań oraz zależności zachodzące pomiędzy nimi pozwalają szybciej rozwiązywać niektóre klasy problemów praktycznych. W szczególności odnosi się to do wielokrotnego rozwiązywania zadań PL-P nieznacznie się różniących , takich jak na przykład otrzymanych poprzez zaburzenie wartości wektora b, lub poprzez dodanie dodatkowego ograniczenia, chrakterystycznych dla schematu odcięć oraz schematu B&B. Postać zadania PL-D zależy od typu operatora $=$, $\leqslant$, $\geqslant$ występującego w ograniczeniu zawierającym macierz A. Przykładowo, dla zadania PL-P w postaci

$$\min_x x_0 \stackrel{\text{def}}{=} cx, \quad Ax = b, \quad x \geqslant 0. \tag{8.59}$$

zadanie PL-D ma postać

$$\max_u u_0 \stackrel{\text{def}}{=} ub, \quad uA \leqslant c \tag{8.60}$$

i nie zawiera ograniczenia na nieujemność zmiennych dualnych u. W dalszym ciągu będziemy zajmować się parą zadań (8.59)-(8.60). Jeśli x oraz u są dowolnymi rozwiązaniami dopuszczalnymi odpowiednich problemów, to z nierówności $ub = uAx \leqslant cx$ oraz $x \geqslant 0$ wnioskujemy, że $ub \leqslant cx$. Dalej oznaczmy przez x^* oraz u^* odpowiednie rozwiązania optymalne. Jeśli któreś z zagadnień ma rozwiazanie nieograniczone, to drugie z nich nie ma rozwiązania dopuszczalnego. Przeciwnie, jeśli jedno z nich ma rozwiązanie skończone to ma je także drugie z nich. Istotnie dla PL-P zachodzi $x^* = B^{-1}b$ oraz $c_N - c_B B^{-1} N \geqslant 0$, na mocy dopuszczalności i optymalności. Zmienne dualne określone wzorem $u = c_B B^{-1}$ są dualnie dopuszczalne, bowiem

$$uA = u(B, N) = (c_B, c_B B^{-1} N) \leqslant (c_B, c_N) = c \tag{8.61}$$

oraz optymalne bowiem

$$ub = c_B B^{-1} b = c x_B^* \tag{8.62}$$

Stąd jeśli problem PL-P ma rozwiązanie optymalne $x_B^* = B^{-1}b$ to problem PL-D ma rozwiązanie optymalne $u^* = cB^{-1}$, które można łatwo obliczyć dysponując odwrotnością macierzy bazowej.

KOnstrukcja *dualnej metody sympleks* opiera się na warunkach koniecznych i wystarczających optymalności. Niech A_i oznacza i-tą kolumnę macierzy A. Warunki te mają postać: (1) jeśli $x_{A_i} \geqslant 0$ to $uA_i = c_i$, (2) jeśli $uA_i \leqslant c_i$ to $x = 0$. Dal takiej pary x oraz u mamy

$$(uA - c)x = 0 \tag{8.63}$$

oraz

$$ub = cx. \tag{8.64}$$

Zależność (8.63) jest wymuszona przez prymarną metodę sympleks bowiem

$$(uA - c)x = (uB - c_B)x_B + (uN - c_N)x_N = 0 \tag{8.65}$$

jednak u jest niedopuszczalne zawsze, z wyjątkiem rozwiązania optymalnego, gdzie zachodzi $uN \leqslant c_N$. Stąd i z równości $uB = c_B$ wynika, że $uA \leqslant c$.

Dualna metoda sympleks generuje w każdym kroku rozwiązanie dualnie dopuszczalne, lecz niedopuszczalne prymalnie (pewne składowe wektora x

mogą mieć ujemne składowe). Prymarna dopuszczalność zostanie osiągnięta dopiero po dojściu do dualnej optymalności. Intuicyjnie, metoda rozwiązuje zadanie prymalne podążając do prumalnej dopuszczalności i zachowując jego warunek optymalności. W kategoriach problemu dualnego metoda zmierz do optymalności zachowując jego dopuszczalność. Poniżej podano kluczowe elementy algorytmu, dalsze szczegóły można znaleźć na prezykład w [351]. Załóżmy, że dane jest rozwiązanie bazowe $x_B = B^{-1}b$ dualnie dopuszczalne to znaczy dla $u = c_B B^{-1}$ zachodzi nierówność $c_N - uN \geqslant 0$. Jeśli $x_B \geqslant 0$ to x_B jest rozwiązaniem optymalnym i algorytm kończy swoje działanie. Inaczej należy wybrać ujemną składową $x_{B_r} < 0$ (zwykle o najmniejszej wartości spośród ujemnych) oraz wykonać proces usunięcia wektora B_r z bazy. Na jego miejsce zostanie wprowadzony wektor niebazowy k spełniający warunek

$$v_{rk} = \min\{|v_{ij}| : \ \ y_{ij} < 0, \ \ j \in N\}. \tag{8.66}$$

Proces przekształcenia jest wykonywaany w oparciu o element centralny y_{rk}.

Schemat odcięć. Algorytm Gomory'ego.

Schemat odcięć jest stosowany do zadań PLC lub PLCM, patrz np. [102]. Idea schematu polega na rozwiązywaniu ciagu zrelaksowanych zadań PL zbiegających do rozwiązania optymalnego. Startowym zadaniem PL jest problem PLC zrelaksowany poprzez usunięcie warunku całkowitoliczbowości. Jeśli otrzymane rozwiazanie optymalne zadania PL jest dopuszczalne dla PLC to proces rozwiązywania kończy się. W przeciwnym przypadku do zbioru ograniczeń aktualnego zadania PL dołączane jest dodatkowe ograniczenie powodujące "odcięcie" otrzymanego optymalnego rozwiązania PL, tak by nie stracić rozwiązania optymalnego problemu PLC, po czym proces się powtarza. Istnieje wiele technik konstrukcji odcięć, zatem w ramach ogólnego schematu, konkretne algorytmy otrzymujemy precyzując sposób wprowadzania odcięcia. Poniżej przedstawiono jedną z metod prowadzącą do algorytmu znanego w literaturze jako algorytm Gomory'ego.

Niech $h \neq 0$ będzie pewną liczbą zwaną dalej *parametrem* odcięcia. Mnożąc (8.34) przez h, dla $i = 0, 1, \ldots, m$ otrzymujemy

$$hx_{B_i} + \sum_{j \in R} hy_{ij}x_j = hy_{i0}. \tag{8.67}$$

Ponieważ $x \geqslant 0$ zatem z oczywistej nierówności $[w] \leqslant w$ mamy

$$[h]x_{B_i} + \sum_{j \in R} [hy_{ij}]x_j \leqslant hy_{i0}. \tag{8.68}$$

Ponieważ x ma być całkowitoliczbowy, zatem lewa strona (8.68) musi być liczbą całkowitą i nie może przekraczać części całkowitej prawej strony, to znaczy

$$[h]x_{B_i} + \sum_{j \in R} [hy_{ij}]x_j \leqslant [hy_{i0}]. \tag{8.69}$$

Mnożąc (8.34) przez $[h]$ i odejmując następnie (8.69) otrzymujemy *podstawowe odcięcie* dla metody odcięć

$$\sum_{j \in R} ([h]y_{ij} - [hy_{ij}])x_j \geqslant [h]y_{i0} - [hy_{i0}]. \tag{8.70}$$

Konkretne algorytmy otrzymujemy podstawiając szczególne wartości parametru h.

Przyjmując $h = 1$ z warunku (8.70) mamy

$$\sum_{j \in R} (y_{ij} - [y_{ij}])x_j \geqslant y_{i0} - [y_{i0}]. \tag{8.71}$$

Ponieważ $y_{ij} = [y_{ij}] + f_{ij}$, gdzie f_{ij} jest częścią ułamkową liczby y_{ij}, warunek (8.71) może być zapisany w formie

$$\sum_{j \in R} f_{ij}x_j - s = f_{i0}, \tag{8.72}$$

gdzie $s \geqslant 0$ jest całkowitoliczbową zmienną osłabiającą.

Istnieje wiele innych sposobów wprowadzania odcięć w zależności od parametru odcięcia, bardziej szczegółową ich dyskusję można znaleźć w pracach [102,351,375]. W praktycznych zastosowaniach zwraca się uwagę na kilka innych elementów. Po wprowadzeniu odcięcia korzystniej jest rozwiązywać zadanie dualne PL-D, zamiast zadania prymalnego PL-P. Wynika to z faktu, że dołączenie dodatkowego ograniczenia w zadaniu pierwotnym nie narusza dopuszczalności zadania dualnego, a jedynie dodaje nową zmienną dualną. W konsekwencji szybciej można otrzymać rozwiązanie zadania PL-D niż PL-P. Należy także zwrócić uwagę na problem zaokrąglania i badania całkowitoliczbowości, szczególnie w kontekście dużych zadań PL i stabilności numerycznej. Niewłaściwe zinterpretowanie wartości zmiennej może być przyczyną bądź przedwczesnego zakończenia pracy algorytmu z błędem, bądź wykonania zbędnej ilości obliczeń.

Na koniec odwołajmy się do zadania (8.51)-(8.53) powiększonego o warunek x_1, x_2 - całkowite. W oparciu o rozwiązanie optymalne zadania PL można wprowadzić odcięcia $\frac{1}{3}x_5 \geqslant 0$ (wiersz zerowy), $\frac{1}{5}x_5 \geqslant \frac{2}{5}$ (wiersz pierwszy), $\frac{7}{15}x_5 \geqslant \frac{3}{5}$ (wiersz drugi), $\frac{14}{15}x_5 \geqslant \frac{3}{5}$ (wiersz trzeci), $\frac{8}{15}x_5 \geqslant \frac{2}{5}$ (wiersz

czwarty). Dodanie ograniczenia wynikającego z tego odcięcia powoduje w następnym kroku zadania PL otrzymanie rozwiązania całkowitoliczbowego optymalnego.

Algorytm Land-Doig'a

Algorytm ten rozwiazuje zadanie PLC lub PLCM w oparciu o schemat B&B. Elementy składowe podstawowej wersji algorytmu omówiono poniżej. Dolne ograniczenie LB otrzymujemy poprzez relaksację warunku całkowitoliczbowości prowadzącą do zadania PL. Zgodnie z zasadą relaksacji, jeśli rozwiązanie zadania PLspełnia warunek całkowitoliczbowości (dopuszczalne dla PLC), to jest rozwiązaniem odpowiedniego zadania PLC. Górne ograniczenie początkowo jest równe $UB = \infty$ oraz jest uaktualniane każdorazowo po otrzymaniu rozwiązania dopuszczalnego $\hat{x}$ podproblemu problemu według zależności $UB = \min\{UB, K(\hat{x})\}$. Podział zbioru rozwiązań jest binarny w oparciu o rozwiązanie x^* zadania PL; niech k będzie indeksem pierwszej zmiennej w x^*, której wartość nie spełnia warunku całkowitoliczbowości. Zbiór rozwiązań ograniczony warunkiem $\{x : Ax = b,\ l \leqslant x \leqslant u,\ x-\text{całkowite}\}$ dzieli się na dwa rozłączne podzbiory $\{x : Ax = b,\ x_k \leqslant [x_k^*],\ l \leqslant x \leqslant u,\ x-\text{całkowite}\}$ oraz $\{x : Ax = b,\ [x_k^*] + 1 \leqslant x_k,\ l \leqslant x \leqslant u,\ x-\text{całkowite}\}$. Strategia przeglądania jest w głąb, przy czym jako pierwszy jest analizowany ten spośród dwóch ostatnio otrzymanych, dla którego wartość dolnego ograniczenia jest mniejsza (odpowiada to technice stosowej obsługi listy problemów częściowych).

Jako przykład rozpatrzmy problem

$$\min(-3x_1 - 2x_2) \tag{8.73}$$

przy ograniczeniach (8.52)-(8.53) powiększonych o warunek

$$x_1, x_2 \ - \ \text{calkowite}. \tag{8.74}$$

Zbiór rozwiazań dopuszczalnych ma postać punktów o współrzędnych całkowitoliczbowych, zaznaczonych w obszarze wielokąta, oznaczmy go przez $\mathcal{X}$. Interpretacja warstwicy pozostaje bez zmiany , jednakże rozwiązanie optymalne generalnie nie leży w wierzchołku zbioru rozwiązań dopuszczalnych. Co więcej próba zaokrąglania rozwiązania ciągłego zadania PL do najbliższych wartości całkowitych nie gwarantuje nawet otrzymania rozwiązania dopuszczalnego (w omawianym przykładzie prowadzi to do otrzymania punktu (4,6)). Stosując algorytm rozwiązujemy wpierw zadania PL otrzymane przez relaksację warunku całkowitoliczbowości. Otrzymane rozwiązanie $x^{*R} = (3.6, 5.6)$ nie spełnia ograniczeń zadania PLC, zatem wartość

$x_0^R = -22$ jest tylko dolnym ograniczeniem wartości ffunkcji celu wszystkich rozwiązań w korzeniu drzewa rozwiązań, rys. **??**. Pierwszą niecałkowitą składową jest $x_1^{*R} = 3.3$, zatem dokonujemy podziału problemu $\mathcal{X}$ na dwa podproblemy: (1) problem (8.73), (8.52), (8.53), (8.74) plus ograniczenie $x_1 \leqslant 3 = [3.6]$, (2) problem (8.73), (8.52), (8.53), (8.74) plus ograniczenie $[3.6] + 1 = 4 \leqslant x_1$. Podział ten zaznaczono w rys. jako P1 oraz odpowiednio w drzewie rozwiązań w rys. **??**. Rozwiązania odpowiednich problemów zrelaksowanych dla obu podproblemów potomnych można odczytać z rys. **??** i są to (3,6) oraz $(4, 4\frac{2}{3})$. Ponieważ pierwszy podproblem został rozwiązany dostarczając rozwiązania dopuszczalnego zatem otrzymujemy $UB = -21$ Dla drugiego podproblemu otrzymujemy $LB = -21\frac{1}{3}$, co nie pozwala nam na zamkniecie węzła i zmusza do dalszego podziału względem drugiej składowej. Kolejny podział pozwla nam zamknać oba węzły potomne, pierwszy ponieważ nie rokuje znalezienia rozwiązania dopuszczalnego mniejszego niż -21, drugiego - ponieważ ma pusty zbiór rozwiązań dopuszczalnych. Optymalnym rozwiązaniem jest $x^* = (3, 6)$ dające wartość funkcji celu -21.

8.3.5 Programowanie liniowe binarne (PLB)

Zadanie PLB odnosi się do problemu PLC, w którym składowe wektora x mogą przyjować wartości binarne 0 lub 1. Mimo iż znane algorytmy PLC mogą być stosowane do tego problemu, podejscie takie nie jest zalecane głównie ze względu na błędy wynikające z zaokrągleń. Zadania PLC można sprowadzić do zadań PB korzystając z dwójkowej reprezentacji liczb całkowitych. Transformacja ta ma bardziej teoretyczne niż praktyczne znaczenie, bowiem powoduje niekorzystny wzrost rozmiaru problemu, a co za tym idzie większy koszt obliczeń.

Algorytm (addytywny) Balasa

Algorytm ten rozwiazuje zadanie PLB w oparciu o schemat B&B bez odwoływania się do rozwiązywania zadania PL. Ponieważ wykorzystuje on tylko operacje dodawania i odejmowania, otrzymane wyniki są dokładne. Elementy składowe podstawowej wersji algorytmu omówiono poniżej.

Rozpatrzmy problem wyjściowy dany w postaci

$$\min_x cx \tag{8.75}$$

$$Ax \leqslant b \tag{8.76}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n, \tag{8.77}$$

gdzie $c_{1\times n}$, $x_{n\times 1}$, $A_{m\times n}$, $b_{m\times 1}$ są macierzami (wektorami) podanych rozmiarów. Bez straty ogólności rozważań możemy przyjąć, że wszystkie $c_j \geqslant 0$. Istotnie, jeśli $c_j < 0$ to wprowadzając transformację $x'_j = 1 - x_j$ otrzymamy problem równoważny z $c_j \geqslant 0$.

Algorytm dokonuje przeglądu bezpośredniego lub pośredniego wszystkich 2^n możliwych wektorów zero-jedynkowych x. W praktyce olbrzymia ilośćrozwiązań jest przeglądana pośrednio. Proces poszukiwania jest przedstawiany za pomocą drzewa, w którym węzłowi odpowiada podzbiór rozwiązań, zawierających pewną iczbę składowych wektora x *ustalonych* porzez przypisanie im wartości zero lub jeden. Zatem podproblem w schemacie B&B jest charakteryzowany poprzez zbiór składowych ustalonych $J \subset N = \{1, 2, \ldots, n\}$ oraz ich wartości x_j, $j \in J$. Podział zbioru rozwiązań jest binarny, w oparciu o pewną jeszcze nieustaloną składową x_k, $k \in N \setminus J$; generowane są dwa problemy potomne poprzez powiększenie o warunek $x_k = 1$ albo $x_k = 0$, odpowiednio. Strategia przeglądania jest w głąb (z powrotami), przy czym z danego węzła jako pierwszy rozpatrywany jest węzeł potomny odpowiadający ustaleniu $x_k = 1$. Niech UB oznacza wartość górnego ograniczenia aktualizowaną każdorazowo po znalezieniu rozwiązania dopuszczalnego, początkowo $UB = \infty$. Dalej niech $z = \sum_{j\in J} x_j c_j$ będzie wartością funkcji celu ustalonego rozwiązania częściowego.

Dla każdego analizowanego węzła wykonywane są kolejno: (1) test poprawy funkcji celu i ograniczenia, (2)test niedopuszczalności, (3) test rozgałęzień. Omówimy je kolejno. Test poprawy diagnozuje czy dalsze rozgałęzienie węzła rokuje nadzieję na poprawę wartości funkcji celu oraz usunięcieniedopuszczalności rozwiazania częściowego x_j, $j \in J$ (jeśli jest ono niedopuszczalne). W tym celu tworzymy zbiór

$$T = \{j \in N\setminus J :\ z + c_j < UB,\ (a_{ij} < 0,\ \text{dla}\ y_i = b_i - \sum_{j\in J} a_{ij}x_j < 0)\} \quad (8.78)$$

Zbiór T zawiera numery składowych, których ustawienie na 1 nie implikuje warunku zamykania $z + c_j = LB \geqslant UB$ oraz zmierza w kierunku przywrócenia dopuszczalności ($a_{ij} < 0$) tych ograniczeń i, w których ona nie występuje ($y_i < 0$). Jeśli $T = \emptyset$ to węzeł jest eliminowany. Jeśli $T \neq \emptyset$ to stosowany jest test dopuszczalności. W teście tym bada się czy istnieje taka składowa i, że $y_i - \sum_{t\in T} \min\{0, a_{ij}\} < 0$ oraz $y_i < 0$. Wtedy węzeł podlega eliminacji bowiem i-te ograniczenie pozostanie niespełnione nawet wtedy, gdyby wszystkim zmiennym należącym do T nadać wartość 1. Nadanie składowej swobodnej wartości 1 może w sposób znaczacy wpłynać na dalszy przebieg obliczeń, ze względu na przyjetą strategię przeszukiwania, zwłaszcza na początkowym etapie obliczeń. Dlatego też wprowadzony został

iter		1	2	3	4	5
1	$J=\emptyset,\ T=\{1,3,4\},\ F=false,\ UB=\infty$					
	M_j	$\{1,2\}$	$\{1,3\}$	$\{2\}$	$\{1,2,3\}$	$\{1,3\}$
	v_j	-4	-7	-3	-5	-8
2	$J=\{3\},\ x_3=1,\ T=\{2,5\},\ F=false,\ UB=\infty$					
	M_j	$\{2\}$	$\emptyset$		$\{2\}$	$\{1,2\}$
	v_j	-5	0		-5	-2
3	$J=\{2,3\},\ x_2=1,\ x_3=1,\ T=\{2,5\},\ F=false,\ UB=17$					
4	$J=\{2,3\},\ x_2=0,\ x_3=1,\ T=\{5\},\ F=false,\ UB=17$					
5	$J=\{2,3\},\ x_2=0,\ x_3=0,\ T=\{1,4\},\ F=false,\ UB=17$					

Tabela 8.6: Iteracje algorytmu dla problemu (8.79)-(8.81)

test rozgałęzień, wybierający odpowiednią składową do poziału. W tym celu dla każdej składowej swobodnej x_j tworzymy zbiór $M_j = \{:\ y_i - a_{ij} < 0\}$ i obliczamy $v_j = \sum_{i \in M_j} |y_i - a_{ij}|$ lub przyjmujemy $v_j = 0$ jeśli $M_j = \emptyset$. Do podziału wybieramy zmienną, dla której v_j jest maksymalne. Jeśli wszystkie zbiory M_jsą puste to węzeł jest eliminowany.

Na koniec rozpatrzmy przykład praktyczny problemu PLB dany w formie

$$\min(5x_1 + 7x_2 + 10x_3 + 3x_4 + x_5) \tag{8.79}$$

$$\begin{array}{rcrcrcrcrcr} -x_1 & + & 3x_2 & - & 5x_3 & - & x_4 & + & 4x_5 & \leqslant & -2 \\ 2x_1 & - & 6x_2 & + & 3x_3 & + & 2x_4 & - & 2x_5 & \leqslant & 0 \\ & & x_2 & - & 2x_3 & + & x_4 & + & x_5 & \leqslant & -1 \end{array} \tag{8.80}$$

$$x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}. \tag{8.81}$$

Kolejne iteracje algorytmu przedstawiono w Tab. 8.6 W tablicy, F oznacza wynik testu dopuszczalności, element podziałowy wyróżniono umieszczając go w kwadracie.

8.3.6 Metody subgradientowe

Metody mogą być stosowane do zadania (8.7) przy założeniu, że $K : \mathcal{R}^n \to \mathcal{R}$ jest funkcją wypukłą zatem wszędzie ciągłą i *subróżniczkowalną*, posiadającą skończoną wartość ekstremum, zbiór $\mathcal{X}$ jest niepusty, domknięty

i wypukły, zaś zbiór $\mathcal{X}^* \stackrel{\text{def}}{=} \{x \in \mathcal{X} : \ K(x) = K^*\}$ jest niepusty i wypukły [217].

Metoda ogólna

Subróżniczką funkcji $K(x)$ w punkcie x jest zbiór

$$\partial K(x) = \{\gamma \in \mathcal{R}^n : \ K(y) \geqslant K(x) + \gamma^T(y - x), \ \forall y \in \mathcal{R}^n\}, \ x \in \mathcal{X}. \quad (8.82)$$

Subgradientem funkcji K w punkcie x jest $\gamma(x) \in \partial K(x)$. Metoda optymalizacji subgradientowej (SUB) generuje ciąg rozwiązań $\{x^s\}$ według zależności

$$x^{s+1} = P_X(x^s - \alpha^s \gamma(x^s)), \ s = 0, 1, \ldots, \quad (8.83)$$

gdzie

$$P_X(x) = \arg\min_{y \in X} \| \ y - x \ \|^2 \quad (8.84)$$

jest operatorem rzutu Eklidesowego punktu $x \in \mathcal{R}^n$ na zbiór $\mathcal{X}$, zaś α^s jest długościa kroku w s-tej iteracji. Odpowiedni dobór ciągu $\{\alpha^s\}$ gwarantuje teoretyczną zbieżność procedury do K^*. Gdy $\mathcal{X} = \mathcal{R}^n$, to rozbieżny ciąg długości kroków

$$\alpha^s > 0, \ \forall s, \ \lim_{s \to \infty} \alpha^s = 0, \ \sum_{s=0}^{\infty} \alpha^s = \infty \quad (8.85)$$

użyty we wzorze (8.83) dostarcza nieskończonego ciągu rozwiązań $\{x^s\}$ takiego, że $\{K(x^s)\} \to K^*$. W przypadku znajomości a priori wartości optymalnej K^* (lub jej dolnego ograniczenia $\underline{K}$) polecany jest także ciąg długości kroków w formie

$$\alpha^s = \lambda^s \frac{K(u^s) - K(u^*)}{\| \ \gamma(x^s) \ \|^2}, \ 0 < \epsilon_1 \leqslant \lambda^s \leqslant 2 - \epsilon_2 < 2, \ s = 0, 1, \ldots \quad (8.86)$$

gdzie λ_s jest pewnym parametrem długości kroku. Jeśli we wzorze (8.86) wielkość K^* zastąpimy przez $\underline{K}$ oraz przyjmiemy $\lambda^s = 1$, to dla każdego $\epsilon > 0$ istnieje skończony indeks $s(\epsilon)$ taki, że $K(x^{s(\epsilon)}) \leqslant 2K^* - \underline{K}^* + \epsilon$. Niestety, w obu przypadkach szybkość zbiegania SUB do rozwiązania optymalnego K^* silnie zależy od przykładu problemu (8.7). W skrajnych przypadkach może być bardzo wolna na skutek wystąpienia zjawiska "zygzakowania".

Metoda warunkowego subgradientu

Warunkową subróżniczką funkcji $K(x)$ w punkcie x jest zbiór

$$\partial^X K(x) = \{\gamma \in \mathcal{R}^n : \ K(y) \geqslant K(x) + \gamma^T(y-x), \ \forall y \in \mathcal{X}\}, \ x \in \mathcal{X}, \quad (8.87)$$

związany z definicją (8.82) zależnością

$$\partial^X K(x) = \partial K(x) \cup N_X(x), \ \forall x \in \mathcal{X}, \quad (8.88)$$

gdzie

$$N_X(x) = \begin{cases} \{\nu \in \mathcal{R}^n : \ \nu^T(y-x) \leqslant 0, \ \forall y \in \mathcal{X}\}, & x \in \mathcal{X} \\ \emptyset & x \notin \mathcal{X} \end{cases} \quad (8.89)$$

jest stożkiem normalnym do $\mathcal{X}$ w x. Odpowiednio, *subgradientem warunkowym* funkcji K w punkcie x jest $\gamma^X(x) \in \partial^X K(x)$. Metoda warunkowej optymalizacji subgradientowej (C-SUB) generuje ciąg rozwiązań $\{x^s\}$ według zależności

$$x^{s+1} = P_X(x^s - \alpha^s \gamma^X(x^s)), \ s = 0, 1, \ldots, \quad (8.90)$$

Dla kroków opisanych wzorem (8.85) przy założeniu, że $\mathcal{X}^*$ jest ograniczony oraz

$$\sup_s\{\| \gamma^X(x^s) \|\} < \infty \quad (8.91)$$

a także

$$\sum_{s=0}^{\infty} (\alpha^s)^2 < \infty \quad (8.92)$$

to $\{K(x^s)\} \to K^*$ oraz $\{x^s\}$ zbiega do pewnego elementu zbioru X^*. Taki sam charakter zbieżności występuje dla kroków opisanych analogiem wzoru (8.86)

$$\alpha^s = \lambda^s \frac{K(u^s) - K(u^*)}{\| \gamma^X(x^s) \|^2}, \ 0 < \epsilon_1 \leqslant \lambda^s \leqslant 2 - \epsilon_2 < 2, \ s = 0, 1, \ldots \quad (8.93)$$

przy założeniu (8.91).

Szczególnym przypadkiem C-SUB jest *warunkowa metoda najszybszego spadku*, w której kierunek $\gamma^X(x)$ wyznacza się poprzez rozwiązanie problemu pomocniczego

$$\min\{\| \gamma \|^2 : \ \gamma \in \partial^X K(x)\}. \quad (8.94)$$

Wyznaczony kierunek jest dopuszczalny jeśli $\mathcal{X}$ jest wielościanem.

Metoda rzutowanego subgradientu

W wielu zastosowaniach optymalizacji subgradientowej (na przykład w schemacie relaksacji Lagrange'a) tylko jeden subgradient w x jest bezpośrednio dostępny, podczas gdy zbiór rozwiązań dopuszczalnych jest opisany zestawem stosunkowo prostych warunków. Ograniczenie problemu rzutowania (8.94) do odpowiedniej różniczki cząstkowej K^X w punkcie x jest problemem *rzutowania subgradientu*

$$\gamma^X(x) = \arg\min\{\| \gamma \|^2 \colon \gamma \in \gamma(x) + N_X(x)\}. \tag{8.95}$$

Dalej symbol $\gamma^X(x)$ będzie używany do oznaczenia rzutowanego subgradientu zamiast warunkowego. Rzutowany subradient może być także otrzymany przez rozwiązanie problemu dualnego do (8.95)

$$\gamma^X(x) = \arg\min\{\| \gamma - \gamma(x) \|^2 \colon \gamma \in -T_X(x)\}, \tag{8.96}$$

gdzie $T_X(x)$ jest stożkiem tangenesowym zbioru $\mathcal{X}$ w punkcie x. Odpowiednia metoda z rzutowanym subgradientem P-SUB jest zbieżna przy tych samych założeniach jak C-SUB.

8.4 Metody przybliżone

Metoda przybliżona A wyznacza pewne rozwiązanie $x^A \in \mathcal{X}$ takie, że $K(x^A)$ jest *bliskie* $K(x^*)$. Metod przybliżonych jest zdecydowanie więcej niż dokładnych, zwykle są problemowo-zorientowane. Dla problemów optymalizacyjnych występujących w DPP dość często dla tego samego problemu istnieje wiele alternatywnych metod rozwiązywania (dokładnych i przybliżonych) o istotnie różnych cechach numerycznych. W celu ułatwienia użytkownikowi orientacji w tej rozległej dziedzinie wprowadzono oceny "dobroci" algorytmów umożliwiające ich wzajemne porównywanie. Zasadniczo, jakość metody przybliżonej jest oceniana z dwóch punktów widzenia: złożoność obliczeniowa algorytmu oraz dokładność przybliżenia. Dalsza charakterystyka bierze pod uwagę mi.in. gwarancje zbieżności do rozwiązania optymalnego, szybkość tej zbieżności. Jakość wszystkich wymienionych ocen zależy od metody, problemu oraz konkretnych danych liczbowych podanych do algorytmu.

8.4.1 Błąd przybliżenia

Zbiór danych liczbowych problemu specyfikuje *przykład konkretny* Z tego problemu. Oznaczmy przez $\mathcal{X}(Z)$ zbiór wszystkich rozwiązań problemu

dla tego przykładu zaś przez $K(x;Z)$ wartość kryterium K dla rozwiązania x w przykładzie Z. Rozwiązanie $x^* \in \mathcal{X}(Z)$ takie, że $K(x^*;Z) = \min_{x\in\mathcal{X}(Z)} K(x;Z)$ jest nazywane rozwiązaniem optymalnym dla tego przykładu. Niech $x^A \in \mathcal{X}(Z)$ oznacza rozwiązanie przybliżone generowane przez algorytm A dla przykładu Z. Za błąd przybliżenia algorytmu A możemy przyjmując jedną z następujących wielkośći

$$B^A(Z) = \left|K(x^A;Z) - K(x^*;Z)\right| \tag{8.97}$$

$$S^A(Z) = K(x^A;Z)/K(x^*;Z) \tag{8.98}$$

$$T^A(Z) = (K(x^A;Z) - K(x^*;Z))/K(x^*;Z) \tag{8.99}$$

$$U^A(Z) = (K(x^A;Z) - K(x^*;Z))/K(x^A;Z) \tag{8.100}$$

Zasadniczo sposób definicji błędu jest dowolny jednakże powinien brać on pod uwagę m.in. następujące elementy: (a) sensowność definicji, np. wyrażenia (8.98)–(8.100) nie mogą być stosowane jeśli $K(x^*) = 0$, (b) adekwatność oceny własności algorytmu, (c) możliwość wykonania analizy zachowania się błędu dla różnych Z.

8.4.2 Analiza zachowania się błędu przybliżenia

Błąd przybliżenia może być badany eksperymentalnie lub analitycznie (analiza najgorszego przypadku, probabilistyczna). Analiza eksperymentalna jest najbardziej popularną i łatwą do przeprowadzenia metodą oceny jakości algorytmu, jednakże jest także metodą subiektywną bowiem jej wynik zależy od wyboru próbki przykładów. Przeciwnie, trudniejsze do wykonania analizy najgorszego przypadku i probabilistyczna dostarczają ocen niezależnych od przykładów. Można powiedzieć, że analizy te dostarczają odmiennych, uzupełniających, czasami bardziej właściwych charakterystyk zachowania się algorytmu. Dopiero wyniki tych ostatnich analiz w połączeniu z wynikami analizy eksperymentalnej oraz analizy złożoności obliczeniowej stanowią kompletną charakterystykę algorytmu.

Analiza eksperymentalna

Jest to najbardziej popularna metoda analizy mimo jej niedoskonałości. Polega na ocenie a posteriori zachowania się algorytmu (błąd przybliżenia, czas pracy algorytmu) w oparciu o wyniki przebiegu algorytmu na ograniczonej, *reprezentatywnej próbce* przykładów konkretnych Z. Ponieważ nie zawsze do końca jest jasne co oznacza pojęcie próbki reprezentatywnej,

otrzymywane wyniki nie zawsze dostatecznie dobrze odzwierciedlają własności numeryczne algorytmu. Próbki mogą być ustalone (zbiory wspólnych przykładów testowych) lub generowana w sposób losowy. Z powodu NP-trudności rozważanych problemów, mogą wystąpić kłopoty z policzeniem rzeczywistych wartości błędów ze względu na zbyt duży koszt obliczenia $K(x^*)$. W tym przypadku re-definuje się pojęcie błędu używając wartości referencyjnej K^{Ref} w miejsce $K(x^*)$. Jako K^{Ref} można przyjąć dolne ograniczenie optymalnej wartości funkcji celu lub wartość $K(x^{Ref})$ gdzie x^{Ref} jest najlepszym znanym rozwi—zaniem, rozwiązaniem przybliżonym (otrzymanym innymi metodami) lub rozwiązaniem losowym.

Analiza najgorszego przypadku

Analiza to ocenia a priori zachowanie się wybranego błędu na *całej* populacji przykładów konkretnych Z. Najczęściej stosowana jest ona do błędu wprowadzonego przez (8.98). Odpowiednio do niego definiuje się następujące pojęcia. *Współczynnik najgorszego przypadku* algorytmu A jako

$$\eta^A = \min\{y : K(x^A; Z)/K(x^*; Z) \leqslant y,\ \forall Z\}. \tag{8.101}$$

oraz *asymptotyczny współczynnik najgorszego przypadku*

$$\eta^A_\infty = \min\{y : K(x^A; Z)/K(x^*; Z) \leqslant y,\ \forall Z \in \{W :\ K(x^*; W) \geqslant L\}\}, \tag{8.102}$$

gdzie L jest pewną liczbą. W sposób oczywisty zachodzi $\eta^A \geqslant \eta^A_\infty$.

Analiza najgorszego przypadku dostarcza ocen skrajnie pesymistycznych. Wystąpienie ekstremalnej wartości błędu, choć możliwe teoretycznie, może w praktyce się nie wydarzyć (lub zdarzyć bardzo rzadko) ze wzgledu na wysoką specyficzność danych instancji. Stąd, oceny pesymistyczne mogą odbiegać znacznie od ocen eksperymentalnych i przeciętnych. Gwarancję ograniczenia wartości błędu posiadają również schematy aproksymacyjne.

Analiza probabilistyczna

Jest to również analiza aprioryczna. Zakłada ona, że każdy przykład konkretny Z został otrzymany jako realizacja pewnej liczby n niezależnych zmiennych losowych o znanym (zwykle równomiernym) rozkładzie prawdopodobieństwa. Dla podkreślenia tego faktu będziemy używać zapisu Z_n zamiast Z. Zatem zarówno $K(x^*; Z_n)$ jak i $K(x^A; Z_n)$ są zmiennymi losowymi. Oznaczmy przez $M^A(Z_n)$ wartość błędu algorytmu A dla przykładu Z_n, np. $M^A(Z_n) = K(x^A; Z_n) - K(x^*; Z_n)$. Oczywiście $M^A(Z_n)$ jest także zmienną

losową. Analiza probabilistyczna dostarcza podstawowych informacji o zachowaniu się zmiennej losowej $M^A(Z_n)$, np. jej rozkładzie prewdopodobieństwa, momentach, etc. Jednakże najbardziej interesujące charakterystyki dotyczą typu zbieżności $M^A(K; Z_n)$ do wartości stałej m wraz ze wzrostem n oraz szybkości tej zbieżności. Wyróżniono następujące typy zbieżności: (a) *prawie na pewno*

$$Prob\{\lim_{n\to\infty} M^A(Z_n) = m\} = 1, \tag{8.103}$$

(b) *według prawdopodobieństwa*

$$\lim_{n\to\infty} Prob\{\Big|M^A(Z_n) - m\Big| > \epsilon\} = 0 \text{ dla każdego } \epsilon > 0, \tag{8.104}$$

(c) *według średniej*

$$\lim_{n\to\infty} \Big|Mean(M^A(Z_n)) - m\Big|\} = 0. \tag{8.105}$$

Zbieżność (a) pociąga (b), lecz (b) pociąga (a) tylko jeżeli dla każdego $\epsilon > 0$ zachodzi

$$\sum_{n=1}^{\infty} Prob\{\Big|M^A(Z_n) - m\Big| > \epsilon\} < \infty. \tag{8.106}$$

Podobnie, (c) pociąga (b), ale (b) pociąga (c) tylko przy spełnieniu wymienionego powyżej warunku. Najlepszym jest algorytm mający błąd zbieżny do zera prawie na pewno.

Analiza probabilistyczna dostarcza ocen uśrednionych po *całej* populacji instancji. Stąd, wyniki obserwowane w eksperymentach mogą odbiegać od ocen teoretycznych. W praktyce, większość algorytmów wykazuje w analizie eksperymentalnej zbieżność błędów względnych do zera przy wzroście rozmiaru problemu. Ponieważ analiza probabilistyczna dostarcza znaczących wyników zachowania się algorytmu, jest zwykle dość skomplikowana. Z tego powodu do tej pory niewiele algorytmów doczekało się starannego opracowania tego tematu.

8.4.3 Schematy aproksymacyjne

Ograniczenie a priori wartości błędu do podanej wielkości zapewniają także algorytmy aproksymujące rozwiązanie optymalne z zadaną z góry dokładnością. Algorytm A jest schematem aproksymacyjnym jeśli dla każdego Z oraz $\epsilon > 0$ dostarcza rozwiązania x^A takie, że $K(x^A; Z)/K(x^*; Z) \leqslant 1+\epsilon$. Algorytm A jest *wielomianowym schematem aproksymacyjnym* jeśli dla każdego ustalonego ϵ posiada on złożoność obliczeniową wielomianową. Jeżeli

dodatkowo ta złożoność jest wielomianem od $1/\epsilon$, to A jest nazywany *w pełni wielomianowy schemat aproksymacyjny.*

W praktyce stopień wielomianu złożoności obliczeniowej rośnie bardzo szybko przy ϵ dążącym do zera. Powoduje to że schematy aproksymacyjne są ciągle algorytmami mało konkurencyjnymi w porównaniu do metod z następnej sekcji.

8.4.4 Metody korzystne eksperymentalnie

W ostatnich latach, niezależnie od rozwoju metod teoretycznych, nastąpił burzliwy rozwój metod przybliżonych o dobrych i bardzo dobrych własnościach numerycznych potwierdzonych ekperymentalnie. Znaczna część tych metod czerpie swoje inspiracje z Natury i ma związek z dziedziną Sztucznej Inteligencji (AI) oraz Uczeniem Maszynowym (ML).

Tradycyjnie metody przyblizone są klasyfikowane jako *konstrukcyjne* lub *poprawiające*. Pierwsze z nich są metodami szybkimi, łatwo implementowalnymi, jednakże generują rozwiązania obarczonych względnie dużym błędem przybliżenia. Te drugie są wolniejsze, wymagają rozwiązania początkowego poprawianego następnie krokowo, oraz dostarczają rozwiązań o bardzo dobrej i doskonałej jkości. Umożliwiają także elastyczne kształtowanie kompromisu pomiędzy jakością rozwiązania a czasem obliczeń.

Teoretyczną ocenę błędu przybliżenia uzyskano do chwili obecnej dla dużej części metod konstukcyjnych i bardzo nielicznych metod poprawiających, głównie ze względu na znaczny stopień skomplikowania analizy. Dla niektórych metod poprawiających wykazano teoretyczną zbieżność do optimum globalnego, jednakże przy pewnych założeniach w praktyce trudnych do spełnienia. Ostatecznie, przydatność praktyczna poszczególnych podejść i metod jest wypadkową własności i analiz teoretycznych oraz analizy eksperymentalnej.

Metody konstrukcyjne

Algorytm konstrukcyjny (constructive method, CA) generuje *pojedyncze* rozwiązanie lub podzbiór rozwiązań o *niewielkiej* ustalonej a priori liczności, z którego następnie wybierane jest rozwiązanie najlepsze w sensie wartości funkcji celu $K(x)$. Algorytmy te są oparte głównie na następujących podejściach: (a) reguły priorytetowe, (b) adaptacja rozwiązania otrzymanego dla problemu zrelaksowanego, (c) przybliżenie rozwiązania rozwiązaniem otrzymanym z pokrewnych problemów, (d) inne. Do grupy CA zaliczane są czasami także metody które krokowo konstruują rozwiązanie (rozszerzając

tak zwane rozwiązanie częściowe), wykonując w tym celu przejście wzdłuż *jednej* ustalonej ścieżki dzrewa rozwiązań prowadzącej od korzenia (rozwiązanie puste) do liścia (kompletne rozwiązanie dopuszczalne). Decyzja o rozszerzeniu jestrozwiązania jest podejmowana na kazdym poziomie drzewa, przy wykorzystaniu pewnej reguły priorytetowej lub przeglądając w tym celu niewielki pomocniczy zbiór rozwiązań częściowych [252]. Przynależność tej ostatniej podklasy metod do grupy CA jest kontrowersyjna, bowiem są one także szczególnym przypadkiem *filtrowanego poszukiwania snopowego* (patrz omówienie dalej).

Algorytmy tej klasy mogą być używane samodzielnie lub w grupach wzajemnie rywalizjących metod. Są tanie obliczenowo, jednak błąd do rozwiązania optymalnego może być znaczny. Są używane także jako generatory rozwiązań początkowych dla algorytmów poprawiających rozwiązania, np. poszukiwań lokalnych. Kombinując pewną liczbę znanych algorytmów tego typu dla przy użyciu parametrów możemy syntetyzować nowe algorytmy dla nowych problemów. Kombinacja taka może uwzględniać zarówno całe algorytmy (który lub które algorytmy są polecana do uruchomienia?), jak i parametrów tych algorytmów (jaka kombinacja reguł priorytetowych jest polecana do zastosowania?). Pewne oryginalne podejście do tego zagadnienia zaproponowano przy wykorzystaniu tak zwanej przestrzeni heurystyk [349], która może dostarczać alternatywnego pojęcia *sąsiedztwa w przestrzeni heurystyk*.

Reguły priorytetowe

Reguły priorytetowe (dispatching rules, priorities) są najbardziej popularną i powszechnie używaną techniką szybkiego wyznaczania rozwiązania, zwłaszcza w systemach produkcyjnych i komputerowych [22,174,279]. Są proste, tanie obliczeniowo, mało wrażliwe na zaburzenia danych wejściowych, nadają się dobrze do systemów niederministycznych. Wyniki badań symulacyjnych określają zwestawy najkorzystniejszych reguł dedykowanych dla kombinacji (klasa systemu produkcyjnego) $\times$ (funkcja kryterialna). Niestety, wybór odpowiednich reguł zależy nie tylko od dwu wymienionych elementów, ale także od danych liczbowych serii konkretnych przykładów do rozwiązywania. Dlatego też praktyczna przydatność tych metod powinna być potwierdzana szczegółowymi badaniami eksperymentalnymi.

Generalnie rozróżnia się reguły statyczne (wartość priorytetu nie zmienia w trakcie oczekiwania na obsługę) oraz dynamicznie (wartość ta ulega zmianie w trakcie). W zależności od problemu oraz zastosowanej reguły, dostarczane rozwiazania mogą być odległe od 25% (przeciętnie) do 500%

(skrajnie) od rozwiązania optymalnego, w sensie miary T^A określonej wzorem (8.99).

Obciete B&B

Ograniczając zasoby komputerowe (czas pracy procesora, pamięć) używane w klasycznym schemacie B&B otrzymujemy algorytm przybliżony (curtailed B&B, truncated B&B, CB) zapewniający pewien kompromis pomiędzy czasem rozwiązywania a dokładnością przybliżenia. Ograniczenia te mogą być wprowadzone a priori do algorytmu B&B przyjmując postać następujących ustaleń: (a) arbitralnie eliminowane są wszystkie problemy częściowe odpowiadające zbiorom $\mathcal{X}_j$ położonym w drzewie rozwiązań głębiej niż pewna wartość progowa, (b) spośród problemów potomnych otrzymanych przez podział $\mathcal{X}_j$ na podzbiory $\mathcal{Y}_k$, $k = 1, \ldots, r$, problemy dla $k > s$, gdzie s jest pewną wartością progową, arbitralnie są eliminowane (s jest stałe lub maleje wraz z głębokością drzewa), (c) ograniczona jest całkowita liczba węzłów w drzewie rozwiązań, (d) ograniczony jest czas pracy algorytmu, (e) arbitralnie eliminowane są podproblemy odpowiadające tym $\mathcal{X}_j$, dla których $(1+\epsilon)LB(\mathcal{X}_j) \geqslant UB$. Mimio iż redukcja czasu obliczeń jest znaczna w porównaniu z klasycznym B&B, algorytmy CB zachowują podstawową negatywną cechę swojego przodka – objawiają znaczący wzrost ilości obliczeń (eksplozję) po przekroczeniu pewnego, względnie niedużego rozmiaru problemu. Dodatkowym mankamentem jest fakt, że dla niektórych aplikacji algorytm CB może w chwili zakończenia nie dostarczyć żadnego rozwiązania dopuszczalnego. Mimo iż określenie CB stosunkowo rzadko występuje w literaturze, pewne szczególne warianty tego podejścia są znane pod innymi nazwami, patrz poszukiwanie snopowe, poszukiwanie progowe.

Poszukiwanie adaptacyjne

Algorytmy te, oznaczane często w literaturze jako A oraz A^*, odpowiadają schematowi B&B, w którym poszzukiwane jest dowolne rozwiązanie przybliżone x^A spełniajace warunek $K(x^A)/K(x^*)) \leqslant 1 + \epsilon$ dla danej małej wartości przybliżenia $\epsilon > 0$. Pozwala to zmniejszenie nakładu obliczeń w stosunku do klasycznego B&B, zwykle za cenę nieznacznego pogorszenia wartości funkcji celu. Podejście takie pozwala kształtować elastycznie relację (czas obliczeń) $\rightarrow$ (jakość rozwiązania), dostosowując ją indywidualnie do aktualnych potrzeb, dostarczajac algorytmów o "pośrednich" własnościach numerycznych. W praktyce eliminowane są arbitralnie podproblemy częściowe odpowiadajace tym $\mathcal{X}_j$, dla których $(1+\epsilon)LB(\mathcal{X}_j) \geqslant UB$. Relacja (koszt

obliczeń) $\leftrightarrow$ (dokładność) jest silnie nieliniowa i silnie specyficzna, nie tylko dla problemów, ale nawet dla konkretnych przykładów liczbowych. Algorytmy adaptacyjne posiadają wszystkie pozytywne i negatywne cechy metod CB.

Poszukiwania lokalne

W ostatnich latach znaczną uwagę zwrócono na podejścia oparte na lokalnym przeszukiwaniu (local search, LS) zbioru rozwiązań $\mathcal{X}$. Bazą metody jest analiza wybranych lub wszystkich rozwiązań leżących w pewnym bliskim otoczeniu $\mathcal{N}(x) \subseteq \mathcal{X}$ wybranego rozwiązania x. Analiza dostarcza rozwiązań, które stają się kolejnymi źródłami lokalnych otoczeń, zastępując rozwiązania bieżące i umożliwiając tym samym powtarzanie procesu poszukiwań. Odpowiednie metody LS łączą wiele zalet: znaczną szybkość pracy, prostotę implementacji oraz mały błąd do rozwiązania optymalnego. Wielu badaczy uważe te metody za najbardziej obiecujące dla szczególnie trudnych problemów optymalizacji dyskretnej. Termin lokalne poszukiwanie odnosi się do całej grupy metod, w tym także tych najbardziej zaawansowanych "odpornych" na lokalne ekstrema. Zwykle implementacja algorytmów tego typu jest prosta, chociaż niektóre z nich są całkiem nietrywialne.

Metody popraw

Poszukiwania lokalne są często nazywane metodami popraw (improvement metods, IM) lub matodami iteracyjnymi (iterative methods), choć pierwsze z tych określeń jest trochę mylące. Nie zawsze każdy pojedynczy cykl iteracyjny LS dostarcza rozwiązania *poprawionego* w sensie wartości funkcji celu, bowiem zdarza się gorsze. Można stwierdzić, że celem nadrzednym IM jest poprawienie, w wyżej wspomnianym sensie, dostarczonego rozwiązania początkowego, to znaczy wyznaczenie, na bazie danego rozwiązania, innego lepszego rozwiazania. Takie stwierdzenie nie precyzuje żadnego konkretnego algorytmu, lecz całą grupę metod o podanej własności. Odnosząc się do pojedynczej iteracji w metodzie LS, rozróżnia się co najmniej dwie odmienne strategie poszukiwania: (a) do pierwszej poprawy, (b) do najlepszej poprawy. W przypadku (a), *pierwsze* napotkane rozwiązanie w otoczeniu spełniajace podany warunek zastępuje rozwiązanie bieżące, powodujac zamknięcie cyklu iteracyjnego. W drugim przypadku zawsze wykonywany jest proces sprawdzenia całego otoczenia i dopiero wtedy rozwiązanie *najlepsze*, to jest spełniajace podany warunek w najlepszym stopniu, zastępuje rozwiązanie bieżące. Obie strategie są zachłanne jednak w rózny

sposób. Łatwo zauważyć, że pierwsza strategia jest mniej kosztowna obliczeniowo, za to wolniej zbieżna. Co więcej, jej dobroć zależy od *kolejności przeglądania* rozwiązań w otoczeniu. Druga strategia jest bardziej kosztowna obliczeniowo, za to szybciej zbieżna.

Krajobraz i doliny

Szereg najnowszych prac z optymalizacji dyskretnej potwierdza silną zależność pomiędzy krajobrazem (landscape) przestrzeni rozwiązań, a eksperymentalnie weryfikowaną dobrocia algorytmów. Krajobraz jest tu rozumiany jako związek, często statystyczny, pomiedzy wzajemną *odległością rozwiązań*, a ich wartościami funkcji celu. Dla rozwiązań reprezentowanych obiektami kobinatorycznymi (permutacje, podziały zbioru, itp.) wymagane są do tego celu specyficzne miary odległości. Zarówno Rys. **??** jak i wyniki innych badań [306] potwierdzają zbliżony do chaotycznego rozkład wartości funkcji celu w przestrzeni, przy czym liczne, rozproszone ekstrema lokalne pełnią rolę *atraktorów*. W tych warunkach poszukiwanie ekstremum globalnego można porównać do wędrówki człowieka przez Himalaje, we mgle, w celu znalezienia punktu położonego najwyżej w bezwzględnej skali wysokości. Dla większości zagadnień można eksperymentalnie potwierdzić istnienie jednej rozległej *doliny* (big valley) zawierajacej liczne, blisko położone "dobre" ekstrema lokalne (lub niewielkiej liczby takich dolin), choć nie wszystkie klasy problemów wykazują tą właściwość. (Analogia do doliny jest dość odległa bowiem powierzchnia reprezentujaca funkcję celu nie jest gładka.) Co wiecej kształt oraz położenie doliny jest zależne od nie tylko od klasy problemu, ale także od konkretnego przykładu liczbowego problemu. Z tego powodu, nowo projektowane algorytmy wyodrębniają i kolekcjonują informację o cechach przestrzeni rozwiązań, wykorzystując ją następnie do modyfikacji kierunków poszukiwań. Ograniczenie poszukiwań do najbardziej obiecujacyh obszarów przestrzeni rozwiązń pozwala w wielu przypadkach zastosować zasadę poszukiwacza grzybów "blisko znalezionego ekstremum lokalnego można znaleźć inne lokalne ekstremum, być może lepsze".

Intensyfikacja i rozproszenie

Wykładnicza wielkość przestrzeni rozwiązań dla problemów praktycznych, połączona z NP-trudnością odpowiednich zagadnień, wyklucza możliwość przeszukania w technice LS więcej niż znikomego ułamka promila wszystkich rozwiazań. Nawet, jeśli obiecujace rozwiązania są skupione w jednej małej dolinie, to i tak bezwzględna liczba rozwiązań tam występują-

cych jest praktycznie zbyt duża, by dokonać choćby ich pobieżnego przeglądu. Stąd powstaje natychmiast pytanie, które rozwiązania należy sprawdzać i według jakiej reguły? Zostało potwierdzone w wielu badaniach, że każdy "rozsądny" proces poszukiwań powinien realizować zarówno staranne, szczegółowe przeszukiwanie niewielkich obiecujących obszarów przestrzeni (intensification), jak i rozproszone przeszukiwanie odległych obszarów przestrzeni, w celu zlokalizownia obszarów najbardziej obiecujących (diversification). Różne znane metody LS realizują oba wymienione elementy w różnym stopniu i w różny sposób. Oddzielnym nietrywialnym zagadnieniem pozostaje zapewnienie globalnej równowagi pomiędzy tymi tendencjami oraz celowe nimi sterowanie w trakcie całego procesu poszukiwań.

Poszukiwanie zstępujące

Metoda szukania zstępującego (descending search, DS) jest historycznie najstarsszą i jedną z prostszych technik poprawy danego rozwiązania $x^0 \in \mathcal{X}$. Metoda ta jest pokrewna do techniki wspinaczki (*hill climbing*) dla problemów maksymalizacyjnych. Elementarny krok tej metody wykonuje, dla danego rozwiązania $x^k \in \mathcal{X}$, przeszukiwanie całego podzbioru rozwiązań $\mathcal{N}(x^k) \subseteq \mathcal{X}$ zwanego *sąsiedztwem* (otoczeniem lokalnym rozwiązania x^k) w celu znalezienia rozwiązania $x^{k+1} \in \mathcal{N}(x^k)$ z najmniejszą wartością funkcji celu $K(x)$. Jeżeli tylko w sąsiedztwie $\mathcal{N}(x^k)$ istnieje rozwiązanie dla którego $K(x^{k+1}) < K(x^k)$, poszukiwanie się powtarza zaczynając od x^{k+1} zaś cały proces process jest kontynuowany dopóki wartość funkcji celu maleje. Trajektoria poszukiwań $x^0, x^1, \ldots$ zbiega monotonicznie w kierunku ekstremum lokalnego, gdzie metoda DS kończy swoje działanie. Startując wielokrotnie metodę DS z różnych (losowo wybranych) rozwiązań początkowych x^0 możemy eliminować w pewnym zakresie wrażliwość DS na lokalne ekstrema. Pewną mutacją metody jest poszukiwanie zstępujace z *pełzaniem* (descending search with drift, DSD), które akceptuje na trajektorii także nieodwiedzone rozwiązania z ta samą wartością kryterium, tzn. $K(x^{k+1}) = K(x^k)$. Generalnie, metoda DS jest najprostszą i najczęściej stosowaną metodą poprawy rozwiązania mimo jej nieodporności na ekstrema lokalne.

Poszukiwanie losowe

Metoda poszukiwań losowych (random search, RS), startując od danego (lub przypadkowego) rozwiązania $x_0 \in \mathcal{X}$, generuje trajektorię poszukiwań $x^0, x^1, \ldots$ w ten sposób, że kolejne rozwiązanie x^{k+1} jest wybierane losowo w sąsiedztwie $\mathcal{N}(x^k)$ (często $\mathcal{N}(x^k) = \mathcal{X}$ dla każdego k). Oczywiście, RS

zwraca jako wynik poszukiwań rozwiązanie najlepsze z trajektorii. Historycznie RS występuje w literaturze także pod nazwą metody Mone-Carlo. Chociaż metody losowe nie jest wrażliwe na ekstrema lokalne, ich zbieżność do dobrego rozwiązania jest ogólnie słaba. Pewne odmiany tej metody dopuszczają modyfikacje rozkładu prawdopodobieństwa przy wyborze sąsiada w celu skierowania poszukiwań w bardziej obiecujący obszar zbioru rozwiązań $\mathcal{X}$. Zmiana rozkładu może mieć charakter arbitralny (wynikający z teoretycznych własności problemu), eksperymentalny (wynikający z badan empirycznych) lub adaptacyjny (uczenie w trakcie szukania).

Poszukiwanie snopowe

Ogólnie poszukiwanie snopowe (beam search, BS) jest podobne do "oświetlania" wiązką (snopem) światła przestrzeni rozwiązań. Każdy "oświetlony" punkt przestrzeni rozwiązań staje się źródłem ukierunkowanej, ograniczonej wiązki świetlnej zawierającej pewną liczbę rozwiązań potomnych, zwykle najbardziej obiecujacych z punktu widzenia procesu poszukiwań. Ocena przydatności rozwiazań może być prowadzona zarówno w oparciu o wartość funkcji celu jak i w oparciu o inne funkcje definiowane przez użytkownika.

Często BS występuje w odniesieniu do schematu B&B, dokładniej CB, gdzie powoduje ograniczenie liczby bezpośrednich następników węzła w drzewie rozwiązań podlegających rozwinięciu. Liczba ta nie może być większa niż pewna ustalona a priori *szerokość wiązki*. Rozwinięciu podlegają zwykle węzły najbardziej obiecujące, np. w sensie wartości dolnego ograniczenia lub wartości pewnej innej (często heurystycznej) funkcji oceny. Wybór węzłów, nazywany filtrowaniem, dostarcza pewnej liczby węzłów wynikającej z *szerokości filtra* podlegajacej dalszej analizie. Metoda ta nosi nazwę *filtrowane poszukiwanie wiązką* (filtered beam search, FBS).

Poszukiwanie progowe

Algorytmy te (threshold algorithms, TA) realizują proces poszukiwania lokalnego, przy czym akceptowane są rozwiązania x' z otoczenia lokalnego $N(x)$, dla których $K(x') - K(x) \leqslant L$, gdzie L jest pewną wartością progową (threshold). Dla $L = 0$ otrzymujemy znany algorytm DS. Algorytmy RS, SA, SJ mogą byćinterpretowane jako TA z wartością L wybieraną losowo w każdym kroku. Klasyczny TA dopuszcza generalnie $L > 0$. Intuicyjnie, wartość L powinna być względnie duża na początku procesu poszukiwań i dążyć do zera pod koniec procesu szukania. Niestety nie ma ogólnych przeslanek do-

tyczących scenariusza zmian wartości L; dobierana jest ona do konkretnych zastosowań indywidualnie.

Poszukiwania ewolucyjne

Poszukiwanie ewolucyjne (genetic search, GS) odwołuje się do Natury zakładając, że niejawnym celem Ewolucji jest optymalizacja dopasowania osobników do środowiska, zaś przekazywanie cech pomiędzy pokoleniami odbywa się zgodnie z teorią ewolucji Darwina. GS używa podzbioru rozproszonych rozwiązań $\mathcal{W} \subset \mathcal{X}$ zwanego *populacją* do prowadzenia poszukiwań równocześnie w wielu obszarach zbioru rozwiązań $\mathcal{X}$. Dzięki temu est w stanie uniknąć ekstremów lokalnych oraz może dotrzeć do wielu odległych obszarów zbioru $\mathcal{X}$.

Każde rozwiązanie x zwane *osobnikiem* jest kodowane przez zbiór jego atrybutów zapisanych w materiale genetycznym (chromosomy, geny). Szereg technik kodowania, specyficznych dla różnych typów problemów, zostało podanych i przebadanych. Populacja jest kontrolowana zasadniczo przez cyklicznie następujące po sobie procesy *reprodukcja*, *krzyżowanie* i *mutacja* oraz *przeżycie lub selekcja*. W fazie reprodukcji, osobniki są powielane proporcjonalnie do ich miary *przystosowania do środowiska*. (Najprostszą funkcją adaptacji jest wartość $K(x)$ dla osobnika x.) Ten proces gwarantuje, że osobniki lepiej przystosowane będą miały więcej potomków w następnym pokoleniu. Osobniki wybrane na z rozszerzonej populacji tworzą *pulę rodzicielską*. Z tej puli kojarzymy pary rodziców, którzy następnie dostarczą *odnowione pokolenie*. Każdy osobnik nowego pokolenia jest nowym rozwiązaniem x' otrzymanym z dwóch rozwiązań rodzicielskich x oraz y przez zastosowanie *operatora krzyżowania genetycznego*. Wiele operatorów genetycznych, specyficznych dla problemów, zostało podanych i przebadanych, np. [59]. Mutacja jest ubezpieczeniem na wypadek utraty istotnych atrybutów rozwiązań oraz powolnym mechanizmem wprowadzania atrybutów innowacyjnych. Mutacja powoduje sporadyczne, losowe (z małym prawdopodobieństwem) zmiany w materiale genetycznym. W fazie przeżycia wybierane są osobniki (spośród starego i nowego pokolenia lub tylko owego pokolena), które wejdą w skład nowej populacji. Przeżycie (selekcja osobników) zwykle wykonywana jest metodą ruletki, dając większe szanse osobnikom lepiej przystosowanym.

Metoda GS posiada wiele punktów swobody. Wymagane jest co najmniej określenie następujących elementów: sposob kodowania atrybutów w chromosomach, postać funkcji adaptacji, schemat wyboru puli rodzicielskiej, schemat kojarzenia rodziców, operatory krzyżowania, schemat mutacji, schemat przeżywania. Chociaż funkcjonowanie wielu z w/w składników zostało

przebadane i opisane, ciągle jeszcze dla nowych problemów kluczem do sukcesu jest ich właściwa kompozycja.

Mimo iż coraz więcej złożonych mechanizmów doboru oraz operatorów genetycznych zaproponowano, pewne niedostatki algorytmów GS są wciąż obserwowane w praktyce. Wyrażają się one przedwczesną zbieżnością do lokalnego ekstremum lub słabą zbieżnością do rozwiązań bliskich optymalnemu. Podczas gdy GS zachowuje się zadawalająco dobrze dla małych przykładów, błędy przybliżenia dla przykładów o dużym rozmiarze mogą być znaczne. Wykazano, że odpowiedzialność za przedwczesną zbieżność do lokalnych ekstremów ponosi niewłaściw sterowanie dynamiką rozwoju populacji. Aby poprawić dopasowanie populacji do środowiska, do reprodukcji w każdym pokoleniu są preferowane najlepsze osobniki, co pociąga stałe zmniejszanie rozproszenia genetycznego populacji. To z kolei zmniejsza możliwości znalezienia istotnie innego lepszego rozwiązania poprzez krzyżówkę genetyczną i wstrzymuje postęp do czasu gdy dopiero mutacja (po ogromnej liczbie pokoleń) wprowadzi potrzebną zmianę w materiale genetycznym. Dlatego też celem nadrzędnym algorytmu GS jest utrzymać dopasowanie maksymalnie skrajne bez utraty rozproszenia genetycznego populacji.

Zaproponowano techniki kontrolowanie zbieżności algorytmu GS przez *strategie kojarzenia rodziców*, *wprowadzenie struktur do populacji* oraz *wzorce zachowań społecznych*. W pracy [112] wprowadzono *funkcję współdzielenia* w dopasowaniu rodziców w celu zapobieżenia zbyt bliskim podobieństwom genotypowym. Inne bezpośrenie podejścia *zapobiegania kazirodztwu* posługują się odległością Hamminga przy ocenie podobieństwa genotypowego, [74]. Tworzenie struktur w ramach populacji może być otrzymane przez podział populacji na *wyspy* (model migracyjny) lub poprzez wprowadzenie *nakładających się sąsiedztw* pokrywających całą populację (model dyfuzyjny), [74, 105, 250]. Oba modele zakładają ograniczenie bezpośredniej wymiany danych pomiędzy osobnikami z różnych obszarów zamieszkałych przez sub-populacje przez co wprowadzają naturalne *nisze ekologiczne* umożliwiające zachowania pożądanego rozproszenia genetycznego. Z kolei podejście oparte o zachowania socjalne przypisuje każdegmu osobnikowi w populacji odpowiednią postawę społeczną, np. zaspokojony, zadowolony lub rozczarowany, [236]. Wzorzec postawy wynika m.in. z wartości funkcji celu wyznaczonej dla osobnika. Ocena aktualnego zachowania ma wpływ na postawę w przyszłości, zatem osobniki mogą różnie reagować w tej samej sytuacji środowiskowej. Każda postawa społeczna wpływa na gotowość osobnika do wchodzenia w krzyżówki, mutacje lub na *uśpienie, klonowanie* (przetrwanie do kolejnego pokolenia).

Sztuczna technka GS jest wciąż rozwijana, poprzez osadzanie kolejnych procesów naśladujących Naturę, głównie w celu zapobiegania stagnacji ewo-

lucji poszukiwań. Najnowsze kierunki zmierzają do rozróżniania cech fenotypowych (obserwowana ekspresja cech osobnika - rozwiązania) i genotypowych (kod cech rozwiązania), dominacji genów oraz nadmiarowego kodowania cech za pomocą wielu genów. Można powiedzieć, że pozostało jeszcze wiele elementów Natury do wykorzystania w optymalizacji.

Do chwili obecnej algorytmy GS były i są stosowane to szeregu problemów z różnym sukcesem. Ze względu na dużą liczbę parametrów sterujących, każda implementacja metody wymaga przeprowadzenia olbrzymiej liczby czasochłonnych eksperymentów komputerowych w celu ich właściwego doboru. Tym niemniej dobre wyniki obliczeniowe otrzymane dla niektórych zastosowań pozwalają uważać GS za obiecujące narzędzie do rozwiązywania trudnych problemów optymalizacji dyskretnej.

Ewolucja różnicowa

Ewolucja różnicowa (differential evolution, DE) jest podklasą metody GS. Demokratyzm tworzenia potomków i mutacji w GS został zastąpiony w DE realizacją *ukierunkowanych* zmian [70] sondujących przestrzeń rozwiązań. DE startuje od losowej populacji rozwiązań. W każdym kroku wykorzystywane są mechanizmy mutacji i krzyżowania jednak przebiegają one odmiennie niż w GS. Każdy osobnik populacji jest rodzicem dokładnie jednego nowego *rozwiązania próbnego* otrzymanego na bazie tego osobnika oraz dwóch rozwiązań *kierunkujących* wybranych losowo z populacji. Generowanie potomka ma cechy kombinacji liniowej rozwiązań z pewnymi elementami losowości. Oddzielny mechanizm losowości zapobiega generowaniu potomka przez proste powielenie rodzica. Ważna rola przypada mutacji, która dzięki specyficznej strategii, jest samo-adaptacyjna oraz celowa co do kierunku, skali i zakresu. Jeśli rozwiązanie próbne jest lepsze w sensie wartości funkcji celu to zastępuje rodzica w populacji, inaczej jest odrzucane. Odnowienie populacji jest realizowane cyklicznie do chwili osiągnięcia zadanej liczby generacji lub stwierdzenia wystąpienia *stagnacji* poszukiwań. Metoda posiada wiele specyficznych operatorów oraz wiele punktów swobody (parametrów) dobieranych eksperymentalnie. Rekomendacje dotyczące wyboru wartości parametrów dla wybranych klas problemów zostały podane w [70].

Tworzenie rozwiązania próbnego odpowiada w pewnym sensie procesowi rekombinacji i/lub mutacji, przy czym analogia jest dość odległa. Bardziej podobne jest do losowych zaburzeń pojedynczego rozwiązania w kierunku innych rozwiązań z populacji zakładając, że populacja zawiera wyłącznie rozwiązania elitarne. Koncepcja ta jest dość skuteczna jeśli rozkład wartości funkcji celu w przestrzeni rozwiązań sugeruje występowanie dolin (va-

leys) skupionych wokół ekstremów lokalnych. Jak do tej pory nie są znane aplikacje DE dla problemów szeregowania chociaż opisano przypadki analizy problemów ze zmiennymi różnych typów. Zraportowane implementacje, m.in. dla problemów projektowania filtrów cyfrowych, projektowania konstrukcji mechanicznych, optycznych wykazują zaskakującą skuteczność DE przy wyjątkowej prostocie konstrukcji algorytmu.

Podejście immunologiczne

Sztuczny system immunologiczny (artificial immune system, AIS) naśladuje w swojej budowie i działaniu system naturalny. W Naturze system immunologiczny jest złożonym, rozproszonym, samo-regulującym, samo-adaptacyjnym systemem wykorzystującym uczenie, pamięć oraz odpowiednie wyszukiwanie informacji w celu obrony organizmu przed patogenami i toksynami powodującymi zaburzenie jego funkcjonowania. Jego podstawowym celem jest rozpoznanie i klasyfkacja komórek na własne (komórki gospodarza) oraz obce (inwazyjne). Komórki obce są następnie dalej klasyfikowane w celu uruchomienia odpowiedniego mechanizmu obronnego *wrodzonego* lub *nabytego*. Głównym typem komórek obronnych są *limfocyty B oraz T* posiadające własność rozpoznawania gospodarza, specyficzność, pamięć i rozproszenie. W systemie występują także inne komórki tzw. fagocytowe (neutrofile, eozynofile, bazofile, monocyty) pełniące funkcje pomocnicze. Pojawieniu się komórki inwazyjnej towarzyszy obecność obcych białek (antygeny) wywołujących reakcję systemu immunologicznego, której pierwotnym celem jest wytworzenie specyficznych przeciwciał blokujących receptory antygenu, a prowadzące w dalszej kolejności do zniszczenia intruza.

AIS buduje analogie do systemu naturalnego przy czym tylko niektóre funkcje, elementy składowe oraz mechanizmy mogą podlegać przeniesieniu. *Antygen* (białko inwazyjne) reprezentuje chwilowe wymagania narzucone na rozwiązanie, np. na zakres niektórych lub wszystkich składowych wektora x, nie wynikające z danych problemu. Repertuar możliwych antygenów jest bardzo duży (czasami nieskończony) zaś aktualna konfiguracja pojawiających się antygenów jest a priori nieznana. *Przeciwciało* (białko blokujące antygen) jest listą instrukcji (algorytmem) na utworzenie rozwiązania spełniającego wymagania określone antygenem. Repertuar dostępnych przeciwciał jest zwykle niewielki lecz istnieje mechanizm ich agregacji i rekombinacji w celu uzyskiwania nowych przeciwciał o odmiennych właściwościach. Wzorce przeciwciał kolekcjonowane są w bibliotece tworząc pamięć systemu. *Dopasowaniem* nazywamy dobranie przeciwciała do antygenu. *Wynik dopasowania* jest idealny jeśli przeciwciało pozwala na wygenerowanie roz-

wiązania spełniającego wymagania antygenu. W przeciwnym przypadku wynik dopasowania jest pewną miarą odchylenia otrzymanego rozwiazania od narzuconych wymagań. Złe dopasowanie zmusza system do poszukiwania nowych typów przeciwciał, zwykle w oparciu o ewolucję (GS, DE).

AIS może być używany w co najmniej dwóch kontekstach: (1) do optymalizacji, (2) do przywracania dopuszczalności i optymalności rozwiązania po zaburzeniu danych wejściowych. W pierwszym przypadku generowany jest ciąg antygenów o coraz ostrzejszych wymaganiach z punktu widzenia wartości funkcji celu. Drugi przypadek nie wymaga komentarza i jest naturalny w systemach produkcyjnych wymagających korekty przyjętych rozwiązań. Zauważmy, że pamięć systemu pozwala na jego adaptację, umożliwiając rozwiązywanie ciągu podobnych instancji problemu optymalizacji odpowiednio efektywnie.

Podejście używające AIS jest ciągle w fazie rozwoju, jego początki są datowane na koniec lat 90-tych. Z tego względu brak jest jeszcze kompleksowych wyników badań konkurencyjności tej metody. W pracy (**??**) podano także referencje do algorytmów DS wspieranych techniką AIS oraz do zastosowań w problemach szeregowania zadań.

Poszukiwania biochemiczne

Problem matematyczny jest kodowany za pomocą sekwencji DNA w formie łańcuchów (o długości 10-30 nukleotydów) z lepkimi końcami, które "pasują" do siebie w ściśle określony sposób. Rozwiązaniem jest sekwencja połączonych łańcuchów, przy czym za rozwiązanie optymalne (dopuszczalne) jest przyjmowany łańcuch o określonej długości złożony z łańcuchów elementarnych. Poszukiwanie rozwiązania polega na losowym sklejaniu się łańcuchów elementarnych i zachodzi w warunkach laboratoryjnych (probówka) w medium zawierającym odpowiednio dużą liczbę powielonych łańcuchów elementarnych. Ze względu na submikroskopową skalę pojedynczego zjawiska sklejania, zachodząca reakcja jest równoważna znacznej liczbie (np. 10^{10}) równoległych procesów poszukiwań losowych. Stwierdzenie czy zostało znalezione wymagane rozwiązanie jest wykonywane testem biochemicznym wykrywającym czasteczki DNA o określonych wielkościach.

W chwili obecnej jeszcze trudno uznać tą metodę za konkurencyjną w stosunku do innych "czysto komputerowych", ze względu na znaczny czas wykonania niezbędnych czynności laboratoryjnych towarzyszących eksperymentowi. Być może zmieni się to w przyszłości, bowiem stale prowadzone są prace w zakresie procesorów biologicznych.

Metody sztucznej inteligencji

Technika ta (artificial inteligence, AI) nie określa konkretnej metody lecz grupę metod kolekcjonujących wiedzę dotyczącą podejmowania decyzji w procesie rozwiązywania problemu, a następnie generującą na żądanie: (a) algorytm o nieznanej a priori strukturze dedykowany do rozwiązania konkretnego przykładu problemu, bądź (b) rozwiązania konkretnego przykładu problemu. Do grupy tej zalicza się także metody EM, GLS lub AMS, choć posiadają one odmienne cechy numeryczne. Do zalet metod AI należy zaliczyć: możliwość autonomicznego generowania kompletnie nowych algorytmów przybliżonych, uwzględnianie nie tylko danych bieżących ale także danych oczekiwanych, dostarczenie efektywnych narzędzi do kolekcjonowania i zarządzania wiedzą za pomocą specyficznych struktur danych. Wśród wad wymienia się nadmierną czasochłonność, nadmierne koncentrowanie się na problemie dopuszczalności rozwiązań co pociąga zbyt ubogą optymalizację, wąską specjalizację.

Metody ekspertowe

Metoda ekspertowa (expert systems, EM) wymaga zasadniczo dwóch elementów: *bazy wiedzy* oraz *modułu wnioskowania* operującego na tej bazie. Baza wiedzy zawiera sformalizowany zapis wiedzy eksperta (człowieka) w formie reguł, procedur, algorytmów lub innych typów abstrakcji. Najczęściej używane są trzy typy reprezentacji wiedzy: proceduralna, deklaratywna i meta. Wiedza proceduralna jest specyficznym, problemowo zorientowanym zestawem wiedzy o sposobach rozwiązywania problemów. Wiedza deklaratywna specyfikuje postać danych we/wy i przetwarzanych. Meta-wiedza określa sposób połaczenia elementów deklaratywnych i proceduralnych w celu rozwiązania problemu. Szereg specyficznych struktur danych jest używanych do reprezentowania wiedzy w bazie, w tym sieci semantyczne, formuły, skrypty, rachunek predykatów, reguły produkcji.

Oddzielnym problemem pozostaje zagadnienie nabywania wiedzy (uczenie) algorytmu ekspertowego. Źródłem wiedzy mogą być ludzie, dane symulacyjne, dane eksperymentalne, bazy danych lub tekst. Automatyczna ekstrakcja wiedzy z przykładów symulacyjnych i eksperymentalnych jest najkorzystniejszą i najwygodniejszą formą nauki, bowiem dla zagadnień o dużych rozmiarach ograniczona percepcja człowieka w procesie optymalizacji. Takie uczenie wykorzystuje przykłady do indukowania reguł produkcji (np. “if ... then ...”) przyjmujących postać drzew decyzyjnych, w których węzły odpowiadają atrybutom klasyfikowanych obiektów, zaś krawędzie alterna-

tywnym wartościom tych atrybutów. Proces uczenia polega na modyfikacji prawdopodobieństw skojarzonych z istniejącymi regułami, lub generacji nowych reguł produkcji [390].

Do tej pory stworzono kilka systemów ekspertowych dla problemów szeregowania (m.in. ISIS, MPECS, OPIS, patrz Rozdz. 7), co świadczy że podejście tego typu jest traktowane jako dobre narzędzie do rozwiązywania dla ogólnej klasy problemów szeregowania przy umiarkowanych żądaniach dotyczacych dokładności rozwiązań.

Metody wieloagentowe

Agent jest unikalnym procesem programowym współdziałającym asynchronicznie z innymi agentami, zwykle w systemie rozproszonym (tzw. metody z rozproszoną inteligencją). Kazdy agent jest autonomicznym, kompletnym systemem rozwiąązywania opartym o wiedzę dotyczącą pewnego podproblemu lub problemu szczególnego. Z tego punktu widzenia, może on reprezentować zarówno dedykowany algorytm, zestaw algorytmów czy też moduł ekspertowy. Zestaw agentów w systemie może być niejednorodny w odniesieniu do wiedzy długoterminowej, kryteriów oceny rozwiązań, celów, języka programowania, algorytmów, wymagań sprzętowych, itp. Zbiór agentów wybranych z "biblioteki" dostępnych, tworzy system wieloagentowy (multiagents). W celu poprawnego i skutecznego współdziałania agentów, zwykle potrzebny jest odpowiedni mechanizm koordynacji, dość specyficzny dla każdej rozważanej klasy problemów. Metoda wieloagentowa nie jest metodą autonomiczną lecz de facto kompozycją metod i algorytmów zaprojektowanych w innych technologiach. Z tego punktu widzenia, wiele zaawansowanych algorytmów szeregowania może być uważanych za metody wieloagentowe.

Metody ulosowione

Podejście to (randomized methods, RM) nie określa konkretnej metody lecz jednie ogólny schemat postępowania polegający na zastępowaniu w znanych algorytmach przybliżonych pewnych parametrów deterministycznych ich odpowiednikami losowymi. Ze względu na chaotyczny charakter przestrzeni rozwiązań, wprowadzenie niewielkich losowych zaburzeń w procesie poszukiwania zwykle poprawia jakość otrzymanego rozwiązania. Dla otrzymanych w ten sposób algorytmów zrandomizowanych można uzyskać teoretyczne, asymptotyczne oceny błędu przybliżenia.

Ścieżki przejściowe

Dla wielu problemów dyskretnych stwierdzono występowanie ekstremów lokalnych skupionych w obszarze pojednczej doliny w przestrzeni rozwiązań. Detekcja czy rozważany problem posiada tą własność, lokalizacja doliny, jak również badanie topologii przestrzeni jest możliwe poprzez śledzenie *ścieżek przechodzących* (paths relinkig, PR) przez tą przestrzeń i łączących wybrane rozwiązania. W zależności od charakteru przestrzeni ścieżki mogą być one generowane poprzez kombinację (liniową) rozwiazań, lub kierunkowane za pomocą specyficznej *miary* odległości [110,306]. W tym ostatnim przypadku, wychodzac od jednego rozwiązania, generuje się trajektorię zawierającą rozwiązania zbiegające się (lub oddalające się) w sensie miary odległości do (od) drugiego rozwiązania Aby dokonać w miarę pełnego "pokrycia" przestrzeni ścieżkami przejściowymi, algorytm powinien operować pewnym podzbiorem rozproszonych rozwiązań, na bazie których dobierana jest kombinacja przejść (najczęściej są to wszystkie pary punktów). Z tego powodu technika PR jest rekomendowana dla metod populacyjnych takich jak GS czy SS.

Podejście to nie określa żadnego konkretnego algorytmu lecz dostarcza pewnego mechanizmu, który może być wbudowany w LS, TS, GS lub inne metody poszukiwań, zwiększając istotnie ich efektywność.

Celowe ścieżki śledzące

Mianem tym (goal-oriented tracing paths, GOTP) określa się ścieżki przejściowe z ustalonym punktem docelowym, zarówno deterministyczne jak i losowe, wykonywane w technice LS. Podane referencje odnoszą się do operatora genetycznego MSFX (multi-step fusion) wykonujacego wielo-krokową fuzję chromosomów rodziców [306] w GS, który realizuje GOTP z elementami losowości podobnymi do SA. Jego zastosowanie w problemach szeregowania przyniosło wyraźne korzyści dla jakości generowanych rozwiązań, niestety czas pracy algorymu nie zwykle dośc długi.

Sterowane poszukiwania lokalne

Metoda sterowanych poszukiwań lokalnych (guided local search, GLS) kieruje trajektorię poszukiwań w najbardziej obiecujące regiony zbioru rozwiązań wykorzystując deterministyczny scenariusz formułowany oddzielnie dla każdego specyficznego problemu, np. [5,29]. Regiony są wybierane a priori w oparciu o szczególne własności rozwiązywanego problemu. Wariant metody ze wspomnianych prac selekcjonuje regiony przez zastosowanie techniki

BS połączonenej z LS, co pociąga za sobą negatywne skutki eksplozji obliczeń. Dodatkowo wariant ten jest dość wyrafinowany implementacyjnie i nie może być rozszerzany w łatwy sposób na inne problemy.

Symulowane wyżarzanie

Metoda symulowanego wyżarzania (simulated annealing, SA) zaproponowana oryginalnie w pracy [1] używa analogii to termodynamicznego procesu studzenia w celu wyprowadzenia trajektorii poszukiwań z ekstremum lokalnego lub uniknięcia takiego ekstremum. Stany ciała są postrzegane jako analogiczne to rozwiązań, zaś energia ciała – analogiczna do wartości funkcji celu. W czasie fizycznego procesu odprężania temperatura jest redukowana powoli w celu dojścia w każdej temperaturze do równowagi energetycznej przy zachowaniu maksymalnej entropii. Stąd, trajektoria poszukiwań jest prowadzona poprzez zbiór $\mathcal{X}$ w "statystycznie właściwym" kierunku.

SA generuje trajektorię poszukiwań $x^0, x^1, \ldots$ przechodzącą przez zbiór $\mathcal{X}$. Kolejne rozwiązanie x^{k+1} na tej trajektorii jest wybrane wśród tych z $\mathcal{N}(x^k)$ w następujący sposób. Wpierw *rozwiązanie zaburzone* $x' \in \mathcal{N}(x^k)$, is wybierane w sposób uporządkowany lub losowo z równomiernym rozkładem prawdopodobieństwa. To rozwiązanie może dostarczyć albo $K(x') \leqslant K(x^k)$ albo $K(x') > K(x^k)$. W pierwszym przypadku x' jest akceptowane natychmiast jako nowe rozwiązanie do następnej iteracji, tzn. $x^{k+1} := x'$. W drugim przypadku x' jest akceptowane jako nowe rozwiązanie z prawdopodobieństwem równym $\min\{1, e^{-\Delta/T_i}\}$, gdzie $\Delta = K(x') - K(x^k)$ zaś T_i jest parametrem zwanym *temperaturą* w iteracji i. (W praktyce x' jest akceptowane jeśli dla wygenerowanej liczby losowej $R[0,1]$ zachodzi nierówność $R < e^{-\Delta/T_i}$.) Jeśli x' nie zostało zaakceptowane (ani przez pierwszy ani przez drugi warunek), ostatecznie podstawiamy $x^{k+1} := x^k$.

Temperatura jest zmieniana w czasie iteracji według *schematu studzenia*. W każdej ustalonej temperaturze wykonywana jest pewna liczba, oznaczmy ją m, iteracji. (Chociaż studzenie powinno przebiegać bardzo powoli, często stosowana jest wartość $m = 1$ w celu uniknięcia długiego czasu działania algorytmu.) Powszechnie są stosowane dwa schematy zmian temperatury: geometryczny $T_{i+1} = \lambda_i T_i$, i logarytmiczny $T_{i+1} = T_i/(1 + \lambda_i T_i)$, $i = 0, \ldots, N-1$, gdzie N jest całkowitą liczbą iteracji, λ_i jest pewnym parametrem liczbowym, zaś T_0 jest temperaturą początkową. Powinno zachodzić $T_N < T_0$ oeraz T_N jest bliskie zeru.

Metoda SA posiada wiele parametrów, które należy dobrać eksperymentalnie. Parametr λ_i jest często wybierany jako stały. Zakładając znane (estymowane) wartości T_0, T_N i N, można wyznaczyć wartość parametru λ_i, wg

zależności $\lambda_i = (T_0 - T_N)/(NT_0T_N)$ dla schematu logarytmicznego, [276]. W pracy [1] dostarczono także wiele cennych rad dotyczących automatycznego wyboru x^0, T_0, λ_i, m oraz kryterium stopu. Zaproponowali oni wybrać x^0 losowo, co wspiera losowość poszukiwań i usuwa wpływ rozwiązania początkowego na proces obliczeń. Temperatura początkowa jest ustalana na poziomie 10 krotnie większym niż maksymalna wartość Δ pomiędzy dowolnymi dwoma kolejnymi rozwiązaniami przy założeniu, że oba są zaakceptowane. W praktyce, rozpoczynając z x^0, generowana jest pewna liczba k kolejnych rozwiązań próbnych $x^0, x^1, \ldots, x^{k-1}$ z których każde jest akceptowane. Następnie wyznaczana jest wartość $\Delta_{\max} = \max_{1 \leqslant i \leqslant k} K(x^{i+1}) - K(x^i)$. Następnie przyjmujemy $T_0 = -\Delta_{max} n(p)$, gdzie $p \approx 0.9$. W pracy [1] podano także metodę wyznaczania λ_i dla logarytmicznego schematu studzenia, tj. $\lambda_i = ln(1+\delta)/(3\sigma_i)$, gdzie δ jest parametrem dokładności osiągnięcia stanu równowagi (0.1 – 10.0) zaś σ_i jest odchyleniem satndardowym wariości funkcji celu $K(x)$ dla wszystkich rozwiązań x generowanych w temperaturze T_i. Wartość m zależy od natury użytego zaburzenia losowego i jest zwykle równa liczbie (lub jej frakcji) różnych rozwiązań, które mogą być osiągnięte z danego rozwiązania przez wprowadzenie pojedynczego zaburzenia, tzn. m jest rzędu $O(\left|\mathcal{N}(x^k)\right|)$. Kryterium stopu algorytmu może być oparte na ograniczeniu czasu przebiegu, liczby wykonywanych iteracji, dostatecznej bliskości temperatury do zera, dostatecznej bliskości pochodnej wygładzonej średniej wartości $K(x)$ w T_i do zera, [73].

Schemat studzenia ma silny wpływ na zachowanie się metody. Jeżeli studzenie jest zbyt szybkie, SA zachowuje się podobnie do metody DS i zwykle szybko kończy swoje działanie w lokalnym ekstremum słabej jakości. Jeżeli studzenie jest zbyt powolne, czas przebiegu algorytmu staje się nieakceptowalnie długi. W praktyce, właściwy dobór wszystkich parametrów sterujących metody wymaga wykonania znacznej liczby testów komputerowych.

Istnieje szereg modyfikacji podstawowego schematu metody, mających na celu poprawę jej wlasności numerycznych. Pierwsza z nich zakłada, że w każdej iteracji jest wybierana pewna próbka $\mathcal{M}^k \subset \mathcal{N}(x^k)$ (o ustalonej liczności) rozwiązań zaburzonych. Wszystkie rozwiązania z próbki są porządkowane zgodnie z wartością funkcji celu. Jeżeli w tej próbce istnieje rozwiązanie x' takie, że $K(x') \leqslant K(x^k)$, to rozwiązanie jest wybierane do następnej iteracji. Jeżeli nie – pierwsze rozwiązanie z próbki jest akceptowane z prawdopodobieństwem $\min\{1, e^{-\Delta/T_i}\}$. W pozostałych przypadku pozostawiamy stare rozwiązanie, tzn. $x^{k+1} = x^k$. Druga modyfikacja zmienia prawdopodobieństwo akceptacji rozwiązania zaburzonego, $e^{-\Delta/T_i}/(1 + e^{-\Delta/T_i})$, w celu akceptacji z jednakowym prawdopodobieństwem zarówno zaburzeń

korzystnych ($\Delta \leqslant 0$) jak i niekorzystnych ($\Delta > 0$) [108]. Kolejna modyfikacja zmienia podstawowy schemat studzenia przez dopuszczenie okresowego wzrostu (skoku) temperatury do jej wartości początkowej T_0.

Pokazano, że przy spelnieniu pewnych założeń metoda SA zbiega do rozwiązania globalnie optymalnego z prawdopodobieństwem jeden [1]. Szczegółowa analiza została także przeprowadzona dla asymptotycznej zbieżności oraz szybkości zbieżności. Ponieważ niektóre z tych założeń są w praktyce niespełnialne, otrzymane wyniki mają bardziej teoretyczne niż praktyczne znaczenie. Do chwili obecnej, SA było stosowane wielu typów problemów z różnym stopniem sukcesu. Niemniej, dobre wyniki otrzymane dla niektórych aplikacji pozwalają uważać SA za jedno z mocniejszych narzędzi do rozwiązywania trudnych problemów optymalizacji dyskretnej.

Symulowane wstrząsanie

Metoda symulowanego wstrząsania (simulated jumping, SJ) także używa termodynamicznej analogii do sterowania poszukiwaniami oraz wymuszania ucieczki z ekstremum lokalnego [13]. Tak zwany proces *wstrząsania* polega na stosowanym naprzemiennie gwałtownym oziębianiu i nagrzewaniu ciała, powodującym stałe "potrząsanie" jego stanem energetycznym. Proces ten jest kontynuowany dopóki pożądany stan końcowy o niskiej energii nie zostanie osiągnięty. Wstrząsanie pozwala nie tylko uzyskąć dostęp do wielu obszarów zbioru rozwiązań ale także zapobiega zatrzymaniu procesu poszukiwań w ekstremum lokalnym i zwiąksza prawdopodobieństwo osiągnięcia minimum globalnego.

W praktyce SJ wprowadza do SA dodatkowo schemat podgrzewania. Podgrzewanie jest stosowane gdy zaburzone rozwiązanie x' nie zostało zaakceptowane, tzn. gdy musimy podstawić $x^{k+1} = x^k$. Schemat ten może przyjąć postać $T_{i+1} = T_i + T_0 * r/i$, gdzie r jest liczbą losową o rozkładzie równomiernym z przedziału [0.0,0.15]. Oczywiście, SJ posiada więcej niż SA elementów wybieranych eksperymentalnie w celu zapewnienia odpowiednio dobrych własności numerycznych algorytmu. Opisane zastosowania SJ pokazują, że jest to obiecujące podejście, jednakże szybkość zbieżności silnie zależy od parametrów sterujących.

Tunelowanie stochastyczne

Metoda ta (stochastic tunneling, ST) próbuje przezwyciężyć słabości SA obserwowane dla silnie szorstkich funkcji celu, poprzez modyfikację funkcji kryterium $K(x)$ w kierunku jej wzrostu $K'(x) = 1 - \exp(-\gamma(K(x) - K^*))$,

gdzie γ jest pewnym parametrem, zaś K^* jest wartością funkcji celu najlepszego dotychczas znalezionego rozwiązania. Taka transformacja zachowuje położenia minimów, ale równocześnie zmniejsza wielkość wahań funkcji celu (zmniejsza szorstkość).

Poszukiwanie z zakazami

Poszukiwanie z zakazami (tabu search, TS) jest metodą zaproponowaną pierwotnie przez Glover [110] a następnie rozwiniętą przez innych autorów. TS powiela naturalny proces poszukiwania rozwiązania wykonywany przez człowieka. Podstawowa wersja TS rozpoczyna działanie od pewnego rozwiązania początkowego $x^0 \in \mathcal{X}$. (Niektóre bardziej zaawansowane warianty TS mogą startować również z rozwiąznia niedopuszczalnego.) Elementarny krok tej metody wykonuje, dla podanego rozwiązania x^k, przeszukiwanie całego sąsiedztwa $\mathcal{N}(x^k)$ of x^k. Sąsiedztwo $\mathcal{N}(x^k)$ jest definiowane poprzez ruchy (przejścia) które można wykonać z x^k. (Ruch transformuje rozwiązanie jedno rozwiązanie w inne rozwiązanie.) Celem tego poszukiwania jest znalezienie w $\mathcal{N}(x^k)$ rozwiązania x^{k+1} z najmniejszą wartością funkcji celu $K(x)$ lub wartością innej niekosztownie obliczanej funkcji oceny. Następnie proces poszukiwania jest powtarzany od najlepszego znalezionego rozwiązania dostarczając trajektorii poszukiwań $x^0, x^1, \ldots$. TS zwraca najlepsze rozwiązanie z trajektorii.

Aby zapobiec cyklicznemu powtarzaniu się rozwiązań na trajektorii, zatrzymaniu w ekstremum lokalnym, oraz aby prowadzić trajektorię poszukiwań w obiecujące obszary zbioru rozwiązań wprowadzono pamięć historii poszukiwań. Wśród wielu klas pamięci używanych dla TS, [110], najczęściej wymieniana jest pamięć krótkoterminowa zwana listą zabronień. Lista ta przechowuje przez ograniczony okres czasu najświeższe rozwiązania z trajektorii poszukiwań, wybrane atrybuty tych rozwiązań, ruchy prowadzące do tych rozwiązań lub atrybuty ruchów, traktując je wszystkie jako formę zabronienia dla ruchów wykonywanych w przyszłości. Wykonanie ruchu posiadającego zabronione atrybuty jest zabronione (odpowiednie rozwiązanie musi zostać pominięte w poszukiwaniach). Zabronienie to może być anulowane jeśli funkcja aspiracji uzna ten ruch za wystarczająco korzystny. Poszukiwanie zatrzymuję się jeśli została wykonana pewna liczba iteracji bez poprawy wartości funkcji celu, wykonano założoną z góry całkowitą liczbę iteracji, przekroczono czas obliczeń, znaleziono rozwiązanie satysfakcjonujące użytkownika. Możliwość znalezienia rozwiązania optymalnego (bez gwarancji znalezienia go) zapewnia tak zwana *własność styczności* (connectivity property) dowodzona dla oddzielnych problemów i sąsiedztw. Własność ta

określa warunki przy których dla dowolnego rozwiązania początkowego x^0, istnieje trajektoria poszukiwań $x^0, x^1, \ldots, x^r$ taka, że $x^{k+1} \in \mathcal{N}(x^k)$ zbieżna do rozwiązania optymalnego $x^r = x^*$. Własność styczności jest także analizowana dla metod SA i SJ. Niezależnie od powyższej własności, dla niektórych wariantów metody TS wykazano teoretyczną zbieżność do rozwiązania optymalnego.

Poszukiwanie z pamięcią adaptacyjną

Poszukiwanie z pamięcią adaptacyjną (adaptive memory search, AMS) jest stosowane do określenia bardziej zaawansowanych schematów TS, które wychodzą poza zbiór składników i narzędzi opisanych w rozdziale poprzednim. Zostało sprwawdzone eksperymentalnie, że proces poszukiwania powinny posiadać swoistą równowagę pomiędzy uszczegółowieniem (staranne przeszukiwanie niewielkich obszarów zbioru $\mathcal{X}$) a rozproszeniem (przeszukiwanie rozproszonych obszarów zbioru $\mathcal{X}$). Tak więc AMS wprowadza szerszy schemat dodając bardziej wyrafinowane techniki pamięciowe (np. *pamięć długoterminowa*, *pamięć częstotliwościowa*, *pamięć atrybutowa*, *haszowa lista zabronień*), nowe elementy strategiczne (np.*strategia listy kandydatów*, *oscylacje strategiczne*, *przełączanie ścieżek*) oraz zachowanie adaptacyjne z uczeniem (np. *zabronienia reagujące* [30,386]). Więcej szczegółów technik AMS można znaleźć w pracy [110]. Chociaż uproszczone wersje TS czasami są zaskakująco skuteczne, metoda AMS jest znacznie bardziej efektywna niż TS.

Metoda geometryczna

Podstawą metody (geometric search) jest pewne geometryczne twierdzenie dotyczące uporządkowania wektorów [32,92,329]. Współrzędne wektorów reprezentują, w tym przypadku, dane problemu. Na bazie w/w własności zbudowane zostało szreg konkretnych algorytmów oraz schematów aproksymacyjnych dla wybranych problemów szeregowania. Mimo, iż otrzymane w ten sposób algorytmy posiadają precyzyjnie określoną teoretyczną zależność pomiędzy kosztem obliczeń a jakością dostarczanych rozwiązań, otrzymane wyniki mają w chwili obecnej bardzie teoretyczne niż praktyczne znaczenie, głównie ze względu na: niską efektywność, kontrowersyjne wyniki w analizie eksperymentalnej, znaczne skomplikowanie implementacji komputerowej algorytmu.

Poszukiwanie mrówkowe

Poszukiwanie mrówkowe (ant colony optimization, ACO) odwołuje się do "inteligentnego" zachowania kolonii "nieinteligentnych" lecz współpracujących osobników, mającej analogię do społeczności mrówek [70, 83]. Metoda ta polecana pierwotnie do problemów optymalizacji stochastycznej znalazła zastosowanie w rozwiązywaniu problemów optymalizacji dyskretnej. Działania poszukiwania są rozproszone pomiędzy osobniki z bardzo prostymi umiejętnościami, które przypominają w swych zachowaniach prawdziwe mrówki. Metoda ta jest inspirowana odkryciem znaczenia *ścieżki feromonowej* w poszukiwaniu przez mrówkę najkrótszej ścieżki prowadzącej z mrowiska do źródła pożywienia i z powrotem. Podczas gdy izolowana mrówka porusza się losowo, trafiając na pozostawioną na powierzchni gruntu ścieżkę feromonową porusza się wzdłuż tej ścieżki z prawdopodobieństwem proporcjonalnym do intensywności śladu feromonowego. Każda mrówka w czasie ruchu znaczy ścieżkę swoim własnym feromonem, zwiększając intensywność śladu. Równocześnie feromon paruje ze ścieżek aby zapobiec eksplozji procesu. Zakładając pewne dopuszczalne ścieżki dla mrówek (zależnie od rozważanego problemu), pewien model zachowania mrówki (mrówka ma pewną pamięć krótkoterminową, może używać wzroku w ograniczonym zakresie, posiada czujnik śladu ferromonowego), początkowy rozkład mrówek na ograniczonym obszarze oraz początkową intensywność ścieżek feromonowych, rozpoczyna się proces symulacji z dyskretnym czasem. Jeżeli tylko nie wystąpi zjawisko stagnacji poszukiwań, algorytm zbiega do pewnego dobrego rozwiązania przybliżonego.

Algorytm mrówkowy dla problemu optymalizacji dyskretnej traktuje rozwiązanie dopuszczalne problemu jako analogiczne do trasy mrówki od mrowiska do źródła pożywienia (dla niektórych zagadnień także z powrotem), zaś wartość funkcji celu jako analogiczną do długości tej trasy. Każda mrówka przechodząc po trasie generuje pojedyncze rozwiązanie, przy czy mrówki przechodzące po trasach krótszych szybciej osiągają punkt docelowy. To z kolei implikuje czestsze krążenie mrówek na trasach najkrótszych nasycając te trasy intensywniej ferromonem. Ze względu na problemy implementacyjne spotykane są dwa odmienne schematy symualcji: (a) przejście mrówek na trasie mrowisko-pożywienie odbywa się w jednym cyklu, zaś mrówki są "premiowane" ilością ferromonu odwrotnie proporcjonalną do długości wykonanej trasy, (b) przejście mrówek na trasie mrowisko pożywienie jest procesem dynamicznym, prędkość mrówki jak i ilość pozostawianego ferromonu są stałe w czasie.

Wyniki otrzymane dla niktórych aplikacji są bardzo obiecujące, jednakże

metoda AS będąc ciągle w fazie rozwoju nie jest jeszcze konkurencyjna do SA czy TS biorąc pod uwagę czas obliczeń i błąd przybliżenia.

Poszukiwania rozproszone

Poszukiwanie rozprszone (scatter search, SS) operuje na zbiorze rozwiązań zwanych *rozwiązaniami referencyjnymi*, które stanowią dobre rozwiązania otrzymane w dotychczasowym procesie obliczeniowym [110]. Metoda SS generuje systematycznie nowe rozwiązania jako liniową kombinację rozwiązań referencyjnych. W wielu przypadkach do otrzymania rozwiązań w cyklu następnym używany jest zestaw procedur heurystycznych. Dzięki temu podejściu, kombinacja liniowa dostarcza dobrych rozwiązań rozproszonych w aktualnym obszarze oszukiwań.

Istnieją pewne podobieństwa i różnice pomiędzy SS a GS. Obie metody są procedurami bazującymi na populacjach rozwiązań, które startują z pewnego podzbioru rozwiązań modyfikowanego w procesie poszukiwań. Kolejne nowe rozwiązania są otrzymywane jako pewna *kombinacja* istniejących. GS dostarcza nowego pokolenia poprzez losowy wybór rodziców oraz losowy wybór punktów krzyżowania chromosomów – w tym sensie realizuje najbardziej demokratyczną politykę zezwalającą na kombinowanie wszystkich rozwiązań. Z kolei SS skupia się na wyborze dobrych rozwiązań jako podstawy do tworzenia nowych kombinacji bez utraty zdolności produkowania rozwiązań rozproszonych. Dodatkowo, w SS i GS zastosowane są odmienne mechanizmy tworzenia kombinacji.

Zlepianie

Zlepianie (chunking, C) nie jest autonomiczną metoda poszukiwania rozwiązania lecz pewnym dodatkowym narzędziem wspierającym skuteczność innych metod szukania. Oznacza ono grupowanie podstawowych informacji w celu utworzenia zredukowanej liczby jednostek informacyjnych wyższego poziomu. Jest uważane za krytyczny aspekt ludzkiej inteligencji [387]. Człowiek, któremu przychodzi rozwiązać trudny problem często rozpoczyna od zgrupowania własności postrzeganych jako pokrewne i przeniesienia ich w takiej formie do procesu rozwiązywania. Daje to możliwość organizacji danych oraz metody rozwiązywania w sposób hierarchiczny, pozwalający zdekomponować problem na podproblemy i tak go rozwiązać. Kiedy problem nie może być zdekomponowany lub metoda dekompozycji jest nieznana, powszechną strategią jest grupowanie atrybutów rozwiązań w celu redukcji wielkości zbioru rozwiązań c może doprowadzić to odkrycia i wykorzystania zależności

wyższego poziomu. Formy *zlepek* są specyficzne dla każdego rozwiązywanego problemu, mogą być predefiniowane przez człowieka lub wykryte w czasie procesu poszukiwań. Zlepianie jest pokrewne technice Uczenia Maszynowego i może być stosowane w wielu metodach lokalnych poszukiwań, np. GS, SA, SJ, TS, AMS.

Spełnienie ograniczeń

W tej metodzie (constraint satisfaction, CS) problem optymalizacyjny jest zastępowany problemem spełnialności tzn. znalezienia rozwiązania x i wartości funkcji $K(x)$ spełniającego zmodyfikowany zestaw ograniczeń [61]. Ogłnie, metoda ta posługuje się *zmiennymi decyzyjnymi* V (np. $(x, K(x))$ w naszym przypadku), *dziedziną* $\mathcal{D}$ tych zmiennych oraz *ograniczeniami* $\mathcal{C}$ nałożonymi na dwie lub więcej zmiennych z V. Podstawowy krok tej metody polega na konstrukcji w uporządkowany sposób rozwiązań poprzez rozszerzanie metodą pierwszy-wgłąb bieżącego rozwiązania częściowego. Każde takie rozszerzenie definiuje nowy CS problem posiadający zmodyfikowane $\mathcal{D}$ i $\mathcal{C}$. Nowa postać $\mathcal{D}$ jest otrzymywana poprzez *propagację ograniczeń* z $\mathcal{C}$. Jeżeli otrzymany CS problem jest niezgodny (niespełnialny), poszukiwania są wycofywane z tego stanu, zaś proces jest kontynuowany z innego zgodnego stanu. Algorytmy CS różnią się typem i poziomem wymuszania zgodności, mechanizmami powrotów ze stanów niezgodnych (np. *skoki powrotne*, *powroty kierunkowo-zależne*, *powroty dynamiczne*) i heurystykami wykorzystywanymi do tworzenia rozwiązań rozszerzonych. Metoda CS posiada pewne podobieństwa do metod B&B i CB, zatem przejawia zarówno pozytywne jak i negatywne cechy tych dwóch podejść.

Poszukiwania ścieżkowe

Metoda ta (path search, PS) wykonuje lokalne poszukiwania używając specyficznego scenariusza [380]. Pewna liczba trajektorii poszukiwań zwanych *ścieżkami poszukiwań* jest generowana z pewnego rozwiązania początkowego. Każda trajektoria $x^0, \ldots, x^k$ zawiera rozwiązania takie, że $x^{k+1} = \min_{x \in \mathcal{N}(x^k)} F(x)$, gdzie $\mathcal{N}(x^k)$ jest sąsiedztwem zaś $F(x)$ pewną *funkcją oceny*, np. $F(x) = K(x)$. Trajektoria jest kończona jeśli dana a priori liczba ostatnio generowanych rozwiązań z tej trajektorii nie poprawia najlepszego znanego rozwiązania. Po zakończeniu nie-perspektywicznej trajektorii, nowa trajektoria jest generowana zaczynając z wybranego (nieodwiedzonego) rozwiązania wśród rozwiązań leżących na trajektoriach już wygenerowanych. Podejście to ma pewne podobieństwa do TS i SA.

Algorytmy memetyczne

Mianem tym określa się klasę algorytmów (memetic algorithms, MA) ewolucyjnych opartych na teoriach ewolucji Lamarcka lub Baldwina. Teoria ta zakłada, odmiennie niż u Darwina, że informacją dziedziczoną pomiedzy pokoleniami jest *mem*, bedący sumą informacji o budowie biologicznej osobnika (*gen*) oraz wiedzy nabytą w czasie życia osobnika. W najprostszym przypadku algorytm memetyczny jest algorytmem genetycznym wzbogaconym o proces uczenia każdego osobnika. Uczenie zwykle ma postać metody opartej na technikach LS.

Do cech pozytywnych metod MA należy zaliczyć lepszą szybkość zbieżności od GS, lepszą jakość generowanych rozwiązań. Wśród wad należy wymienić długi czas działania, przedwczesna stagnacja przez zmniejszenie się rozproszenia genetycznego populacji.

Algorytmy kulturowe

Algorytmy kulturowe (cultural algorithms, CA) sąszczególną klasą algorytmów ewolucyjnych, które wykorzystuja wiedzę o specyficznych własnościach problemu zgromadzoną w formie *kultury* społecznej do poprawy szybkości działania algorytmów, a przede wszystkim do poprawy ich dokładności. Osobniki w populacji reprezentują metody rozwiązywania problemu, kompletne algorytmy lub ich składniki. Wpierw, osobniki są *oceniane* przy pomocy pewnej funkcji oceny w zakresie nabytego lub posiadanego poziomu *doświadczenia* dotyczącego umiejetności rozwiązania konkretnego problemu lub klasy przykładów problemu. *Funkcja adaptacji* określa, który osobnik w bieżącej populacji jest w stanie rozwiązać przedstawiony problem, lub *głosować* w celu ustalenia stanowiska wspólnego dla całej *grupy decyzyjnej* (elita społeczna, rząd). Doświadczenia tych pojedynczych osobników są używane do *określenia przekonań* całej grupy decyzyjnej, która ma następnie *wpływ* na kierunki i zakres ewolucji populacji, zmierzajac do samo-adaptacji w oparciu o przestrzeń przekonań (belief space). Łatwo zauważyć podobieństwo tej klasy algorytmów do tecchnikLS z przestrzenią heurystyk.

Sieci neuronowe

Chociaż sieci neuronowe (neural networks, NN) są dziedziną intensywnie rozwijaną, ich zastosowanie do rozwiązywania problemów dyskretnych wydaje się wciąż dalekie od oczekiwanego. Generalnie spotyka się dwa odmienne podejścia.

Pierwszy z nich używa deterministycznego deterministycznego, analogowego modelu sieci Hopfielda i Tanka [393] dla reprezentacji ograniczeń i funkcji celu. Model sieci buduje się indywidualnie dla każdego rozważanego problemu. Poszczególne skladowe rozwiązania mają swoje analogie do napięć w sieci, ograniczenia są przekładane na interakcje neuronów, zaś funkcja celu ma analogie do energii sieci. Do minimalizacji wykorzystuje się własność sieci polegającą na samorzutnym zbieganiu od pewnego stanu początkowego do stanu minimalnego energetycznie. Obliczenia wykonywane są poprzez wielokrotne starty z różnych losowych stanów początkowych. Sieci te są często krytykowane za poważne braki, jak na przykład niemożność znalezienia rozwiązania globalnie optymalnego, problemy ze skalowaniem, niemożność wyznaczenia nawet rozwiązania dopuszczalnego.

Podejście drugie odwołuje się do sieci Kohonena lub dość odległych analogii do sieci Hecht-Nielsena (counter propagation) z rywalizacją i zwykle wymaga dwóch faz, patrz przegląd w pracy [354]. W fazie *uczenia* wagi synaptyczne są modyfikowane na podstawie prezentacji *ciągu uczącego* instancji Z oraz odpowiadających im rozwiązań optymalinych (quasi-optymalnych). Jako nauczyciela wybiera się algorytm o dużej dokładności generowanych rozwiązań, zwykle wykonany w technice B&B lub GS, TS, SA. Uczenie przebiega w okresie bezczynności sieci i jest prowadzone w oparciu o serie instancji losowych lub pochodzących z praktycznego procesu, w którym sieć jest zainstalowana. Ten drugi typ uczenia jest zwykle korzystniejszy bowiem pozwala "zaadoptować" ją do środowiska przyszłej pracy. W fazie *rozpoznawania* rozwiązanie, dla podanej instancji, jest generowane przez sieć. Poważnym problemem jest kombinatoryczny charakter rozwiązań niektórych problemów dyskretnych (takich jak na przykład permutacje), trudnych do reprezentowania przez stany i wyjścia sieci. Tak rozumiana kombinatoryczność wymaga stosowania problemowo-zorientowanego *translatora*, tłumaczącego wartości wyjścia na legalne (dopuszczalne) rozwiązanie. Alterntywną techniką jest dopuszczenie by wyjścia reprezentowały bezpośrednio rozwiązanie, natomiast jego niedopuszczalność jest wykrywana przez problemowo-zorientowany *detektor* i korygowana w sposób iteracyjny poprzez czasową modyfikację waag synaptycznych, zmierzającą w kierunku prowadzącym do uzyskania rozwiązania dopuszczalnego.

Poszukiwanie rojem cząstek

Metoda ta (Particle Swarm Optimization, PSO) jest inspirowana ruchem roju (swarm) osobników mających analogię do stada ptaków, ryb, zwierząt, owadów, etc. w Naturze. PSO używa podzbioru rozwiązań $\mathcal{W} \subset \mathcal{X}$ nazy-

wanego rojem cząstek (swarm of particles) dla rozproszonego przeszukiwania przyległych obszarów przestrzeni rozwiązań. Rozwiązanie $x \in \mathcal{W}$ odpowiada *położeniu* cząstki. Cząstki poruszają się w przestrzeni po trajektorii $x_1, x_2, \ldots$ określonej przez sekwencję kolejnych ich położeń

$$x_{k+1} = x_k + v_{k+1} \tag{8.107}$$

gdzie v_{k+1} jest *prędkością* cząstki, określoną równaniem

$$v_{k+1} = w \cdot v_k + c_1 r_1 (pb - x_k) + c_2 r_2 (gb - x_k). \tag{8.108}$$

Pre-definiowane współczynniki w (inercja), c_1, c_2 są ustalane na drodze eksperymentalnej. Wielkości r_1, r_2 są wybierane losowo w każdym kroku. Wartość gb jest położeniem aktualnie najlepszego globalnie rozwiązania (lider roju), zaś pb jest najlepszym rozwiązaniem w lokalnym otoczeniu x.

Poszukiwanie nietoperza

Metoda ta (Bat Algorithm, BA) jest jedną z metod opartą na inteligencji roju, inspirowaną zachowaniem echolokacyjnym nietoperzy. BA stosuje dostrajanie częstotliwości oraz automatykę równowagi pomiędzy intensyfikacją a dywersyfikacją poprzez sterowanie głośnością oraz poziomem impulsu echolokacyjnego.

Poszukiwanie harmoniczne

harmony search

Poszukiwanie ze zmiennym sąsiedztwem

Metoda ta (Variable Neighborhood Search, VNS) jest metaheurystyką wykonującą stale i systematycznie zmiany sąsiedztwa, w celu zbiegnięcia do lokalnego minimum bądź ucieczki z dolin zawierających te mninima. Poszukiwanie w obrębie pojedynczego sąsiedztwa jest wykonywane techniką DS.

Chciwe poszukiwanie losowe

Chciwa ulosowiona adaptacyjne metoda poszukiwania (Greedy Randomized Adaptive Search Procedure, GRASP) jest metaheurystyką iteracyjną, poprawiającą. Iteracja metody składa się z fazy konstrukcji rozwiązania początkowego *chciwego* ulosowionego oraz z fazy poprawy tego rozwiązania przy pomocy poszukiwań lokalnych (np. DS). Każde chciwe rozwiązanie

ulosowione jest generowane algorytmem listowym, poprzez krokowe rozszerzanie rozwiązania częściowego w oparciu o listę rankingową (ograniczoną listę kandydatów według ich udziału w najlepszych dotychczas znalezionych rozwiązaniach) z wykorzystaniem elementów losowości.

Poszukiwanie rojem pszczół

Sztuczny rój pszczół (Artificial Bee Swarm, ABS) jest populacyjną metodą przeszukiwania sąsiedztwa połączoną z poszukiwaniem losowym wykorzystującymi współpracę (uczenie), funkcjonującymi na podobieństwo roju pszczół, ?. Rój pszczół zbiera miód do ula. Każda pszczoła przemieszcza się losowo do obszaru przestrzeni poszukiwań gdzie zbiera nektar (sprawdza rozwiązania). Jej efektywność jest oceniana po powrocie w oparciu o zebrany nektar. Najwyżej ocenione osobniki są wybierane do realizacji *tańca pszczół* (waggle dance) w celu poinformowania pozostałych osobników roju o najbardziej obiecujących regionach poszukiwań (kierunek, odległość, jakość). Kolejnego dnia pszczoły wylatują w kierunkach zmodyfikowanych w oparciu o otrzymaną informację. Alalogia do problemu optymalizacyjnego jest oczywista. Wartość kryterium jest odwrotnie proporcjonalna do ilości zebranego nektaru. Lokalizacja kwiatów będacych celem pszczół odpowiada rozwiązaniom. Taniec pszczół jest ekspresją cech statystycznych poszczególnych regionów przestrzeni rozwiązań.

Metody hybrydowe

Pewne braki metod przybliżonych, takie jak zbyt wolna zbieżność, przedwczesna zbieżność do ekstremum lokalnego słabej jakośći, stagnacja poszukiwań, obserwowane w badaniach eksperymentalnych, są motorem tworzenia konstrukcji hybrydowych (hybrid algorithm, HA) łączących kilka wymienionych powyżej podejść, np. SA+TS, TS+NN, itp. Przykładowo, dość popularne jest wsspomaganie techniki GA przez LS, skutkujące podejściem MA. Otrzymywane wyniki są silnie specyficzne dla kombinacji problem/algorytm i zwykle nieco lepsze od tych osiąganych dla podejść "czystych". Niezależnie od powyższego, metody przybliżone mogą być wbudowane jako algorytmy pomocnicze dla metod dokładnych. I tak w schemacie B&B mogą one wspomagać proces aktualizacji górnego ograniczenia przyspieszając jego zbieżność, mogą być podstawą reguły podziału, a także reguł eliminacji.

Biorąc pod uwagę znaczną różnorodność metod przybliżonych, repertuar możliwych do wygennerowania algorytmów hybrydowych jest dość znaczny, i trudny do ogarnięęcia. Stąd stwierdzenie, które kombinacje metod wnoszą

nowe, znaczące poprawy istniejących algorytmów, a które są wyłącznie widowiskowymi pomysłami, wymagają zwykle rozleglych, wiarygodnych badań eksperymentalnych.

Metody równoległe

W ostatnich latach wzrost mocy obliczeniowej PC realizowany jest głównie poprzez wprowadzenie architektur równoległych (wielordzeniowość, wieloprocesorowość). Ponieważ wzrost liczby procesorów lub rdzeni w komputerze jest wciąż zbyt wolny w porównaniu do wzrostu wielkości przestrzeni rozwiązań przy zwiększaniu rozmiaru problemu, póki co nie ma nadziei na usunięcie bariery NP-trudności. Z drugiej strony, równoległość obliczeń pozwala na istotną poprawę cech numerycznych metod przybliżonych. Stąd równoległe metaheurystyki stały się obecnie najbardziej interesującą i rozwijaną klasą algorytmów nowej generacji, ?. Metody równoległe odwołują się do kilku fundamentalnych pojęć: (1) teoretyczne modele obliczeń równoległych (SISD, SIMD, MISD, MIMD), (2) teoretyczne modele dostępu do pamięci współdzielonej (EREW, CREW, CRCW), (3) praktyczne środowiska obliczeń równoległych (sprzęt, oprogramowanie, GPGPU), (4) programowanie z pomięcią współdzieloną (Pthreads w C, wątki Java, Open MP w FORTRAN-ie, C, C++), (5) programowanie z pamięcią rozproszoną, przesyłanie komunikatów, (6) obliczenia w Internecie (PVM, MPI, Sockets, Java RMI, CORBA, Globus, Condor), (6) miary jakości algorytmów równoległych (czas przebiegu, szybkość, efektywność, koszt), (7) pojedyncze/wielokrotne wątki, (8) oszacowanie ziarnistości, (9) wątki niezależne/kooperujące, (10) rozproszone (zawodne) obliczenia w sieci.

Zauważmy, że prawie każda z metod opisanych powyżej może być implementowana w środowisku obliczeń równoległych w różny nietrywialny sposób dostarczając kolekcji algorytmów o odmiennych własnościach numerycznych. Rozpatrzmy dla przykładu podejście SA. Możemy zaadaptować tą metodę do warunków równoległych w następujący sposób; (a) pojedynczy wątek, konwencjonalne SA, równoległe obliczanie wartości funkcji celu, mała ziarnistość, teoretyczna zbieżność wynikająca z SA, (b) pojedynczy wątek, równoległe SA, podzbiór losowych rozwiązań rozważanych w sąsiedztwie, równoległa ocena próbnych rozwiązań, teoretyczne zbieżność tzw. równoległego SA, (c) badanie stanu równowagi w ustalonej temperaturze, (d) wiele wątków niezależnych, obliczenia gruboziarniste, (e) wiele wątków kooperujących, obliczenia gruboziarniste. Z kolei dla GS mamy: (a) pojedynczy wątek, konwencjonalne GA, równoległe obliczenia wartości funkcji celu, mała ziarnistość, dowód zbieżności wyikający z GS, (b) pojedynczy wątek, równoległa

ocena populacji, (c) wiele wątków niezależnych, obliczenia gruboziarniste, (d) wiele wątków kooperujacych, (e) subpopulacje rozproszone, model migracyjny, dyfuzyjny, wyspowy.

Wymienione przykłady sugerują, że dla jednego ustalonego podejścia sekwencyjnego możemy zaproponować wiele alternatywnych podejść równoległych. Stąd ostateczna liczba podejść równoległych może być znaczna. Dla środowisk równoległych, oprócz oczywistego skrócenia czasu obliczeń obserwowane jest zjawisko *ponad liniowego* przyśpieszenia.

8.5 Wnioski i uwagi

Pełny przegląd metod przybliżonych nie wyczerpuje listy podejść znanych w iteraturze (pominięto m.in. poszukiwanie harmoniczne, elektromagnetyczne). Jakość (dokładność) poszczególnych metod zależy od krajobrazu przestrzeni rozwiązań, szorstkości funkcji celu, występowania wielkiej doliny, rozkładu rozwiązań w przestrzeni i odpowiedniej równowagi pomiędzy intensyfikacją a dywersyfikacją poszukiwań. Aktualnie rekomendowanymi podejściami są: SA, SJ, GS, MS – dla problemów bez żadnych szczególnych własności (SA i SJ dla problemów posiadających duży koszt obliczenia wartości funkcji celu dla pojedynczego rozwiązania) oraz TS, AMS – dla problemów posiadających pewne własności szczególne. Do rozwiązywania instancji o większym rozmiarze polecamy metody równoległe, możliwe obecnie do zaimplementowania już na PC.

9

Podstawowe problemy jednomaszynowe

Stopień skomplikowania problemów szeregowania zależy silnie od liczby maszyn w systemie, struktury zadań oraz od charakteru i liczby dodatkowych ograniczeń. Generalnie, problemy jedno-maszynowe należą do jednych z najprostszych zagadnień szeregowania, choć wiele z nich należy już do klasy problemów NP-trudnych, patrz między innymi przegląd w pracy [152]. Umożliwiają one modelowanie i analizę zarówno prostych systemów wytwarzania jak i wybranych pojedynczych stanowisk w złożonych systemach produkcyjnych. Są one także używane jako problemy pomocnicze przy rozwiązywaniu bardziej złożonych zagadnień szeregowania. Łatwość opisu, liczne własności szczególne, w miarę efektywne algorytmy oraz dobre aplikacje praktyczne powodują, że problemy te są jednymi z częściej analizowanych, prowadząc jednocześnie do rozwoju nowych podejść i algorytmów. Dobre zrozumienie tych podstawowych problemów oraz metod ich rozwiązywania stworzy solidne podstawy dla analizy problemów trudniejszych. W niniejszym rozdziale będziemy analizować niektóre problemy jedno-maszynowe, te z kryterium $C_{\max}$ i $f_{\max}$, w kolejności wzrastającej stopniowo ogólności, a co za tym idzie – wzrastającej złożoności obliczeniowej. Dla omawianych zagadnień będzie zwykle formułowanych kilka alternatywnych metod rozwiązywania z omówieniem ich charakterystyki numerycznej; ma to na celu pełne ukazania spektrum technik i podejść stosowanych do projektowania algorytmów. Ma to na celu pelne ukazanie spektrum technik i podejść stosowanych do projektowania algorytmów.

9.1 Problem podstawowy

Zbiór zadań $\mathcal{J} = \{1, 2, \ldots, n\}$ jest przeznaczony do wykonywania na jednej maszynie o ograniczonej jednostkowej przepustowości. (Mówimy, że maszyna ma przepustowość $k \geqslant 1$ jeśli może wykonywać co najwyżej k zadań równocześnie.) Każde zadanie składa się z pojedynczej operacji posiadającej ustalony, znany a priori, czas wykonywania $p_j > 0$, $1 \leqslant j \leqslant n$. (Zatem $\mathcal{J} = \mathcal{O}$.) Rozwiązaniem jest harmonogram pracy stanowiska reprezentowany przez wektory terminów rozpoczęcia $S = (S_1, \ldots, S_n)$ oraz zakończenia zadań $C = (C_1, \ldots, C_n)$, spełniające powyższe ograniczenia. W praktyce, ponieważ $C_j = S_j + p_j$, zatem rozwiązanie jest całkowicie charakteryzowane przez jeden z tych wektorów. Jeśli funkcja celu jest regularna, to harmonogram optymalny leży w klasie harmonogramów dosuniętych w lewo. Wtedy też każdy harmonogram może być jednoznacznie reprezentowany kolejnością wykonywania zadań na stanowisku, to zaś z kolei jest reprezentowana permutacją $\pi = (\pi(1), \ldots, \pi(n))$ na zbiorze $\mathcal{J}$. W tym przypadku dla danej permutacji π, terminy rozpoczęcia $S_j = C_j - p_j$ i zakończenia wykonywania C_j określone są wzorem

$$C_{\pi(j)} = C_{\pi(j-1)} + p_{\pi(j)}, \quad j = 1, \ldots, n, \tag{9.1}$$

gdzie $\pi(0) = 0$ oraz $C_0 = 0$. Jeśli funkcją kryterialną jest $C_{\max}$ to każda permutacja jest optymalna.

9.2 Terminy gotowości i zakończenia

Wpierw rozważmy uogólnienie problemu podstawowego otrzymane poprzez wprowadzenie w problemie z Rozdz. 9.1 niezerowych terminów gotowości r_j (znane a priori terminy zgłoszenia się zadań). Oznacza to, że każdy poszukiwany harmonogram pracy maszyny musi spełniać warunek $r_j \leqslant S_j$. Problem ten oznaczony $1|r_j|C_{\max}$ posiada algorytm wielomianowy o złożoności $O(n \log n)$, korzystający z następującej własności znanej jako reguła Jacsona: istnieje rozwiązanie optymalne problemu $1|r_j|C_{\max}$, w którym zadania są wykonywane w kolejności niemalejących wartości r_j. Zasada ta ta odpowiada obsłudze zgłoszeń według reguły FIFO (First-in-First-out). Zatem w celu rozwiązania problemu generujemy permutację π odpowiadającą uporządkowaniu zadań według niemalejących wartości r_j, a następnie wyznaczamy odpowiednie terminy rozpoczęcia i zakończenia wykonywania zadań ze wzoru

$$C_{\pi(j)} = \max\{r_{\pi(j)}, C_{\pi(j-1)}\} + p_{\pi(j)}, \quad j = 1, \ldots, n, \tag{9.2}$$

zaś wartość funkcji celu wynosi $C_{\max}(\pi) = C_{\pi(n)}$.

Kolejne uogólnienie można otrzymać wprowadzając w problemie z Rozdz. 9.1 żądane terminy zakończenia d_j. W tym przypadku jako kryterium optymalizacji przyjmujemy maksymalne spóźnienie

$$L_{\max}(\pi) = \max_{1 \leqslant j \leqslant n} L_j = \max_{1 \leqslant j \leqslant n} (C_j - d_j). \tag{9.3}$$

Problem ten oznaczony $1||L_{\max}$ posiada algorytm wielomianowy o złożoności $O(n \log n)$, korzystający z następującej własności: istnieje rozwiązanie optymalne problemu $1||L_{\max}$, w którym zadania są wykonywane w kolejności niemalejących wartości d_j. Zasada ta znana jest pod nazwą EDD (Earliest Due Date) i odpowiada priorytetowej regule obsługi zgłoszeń według pilności. Zatem wpierw generujemy permutację π odpowiadającą uporządkowaniu zadań według niemalejących wartości d_j, a następnie wyznaczamy odpowiednie terminy rozpoczęcia i zakończenia wykonywania zadań ze wzoru (9.1) oraz następnie wartość funkcji celu według (9.3).

Innym praktycznym uogólnieniem problemu z Rozdz. 9.1 jest wprowadzenie czasów dostarczenia zadań q_j mających sens niekrytycznych czynności zakończeniowych (wymagających pewnego czasu) wykonywanych po zasadniczej obsłudze zadania na maszynie. Problem ten oznaczany przez $1|q_j|C_{\max}$ ma swój równoważny odpowiednik w formie problemu $1||L_{\max}$, patrz Rozdz. 9.4 oraz jest "symetryczny" do problemu $1|r_j|C_{\max}$. W tym przypadku symetria oznacza, że optymalne wartości funkcji celu obu problemów symetrycznych są sobie równe zaś permutacje optymalne są wzajemnie odwrotne. Odpowiednio rozwiązane optymalne można wyznaczyć poprzez zastosowanie oczywistej poniższej własności: istnieje rozwiązanie optymalne problemu $1|q_j|C_{\max}$, w którym zadania są wykonywane w kolejności nierosnących wartości q_j. Zasada ta jest szczególnym przypadkiem reguły priorytetowej MWR (Most Work Remaining), według której zadanie o największej pozostałej pracochłonności uzyskuje najwyższy priorytet. Zatem wpierw generujemy permutację π odpowiadającą uporządkowaniu zadań według nierosnacych wartości q_j, a następnie wyznaczamy odpowiednie terminy rozpoczęcia i zakończenia wykonywania zadań ze wzoru (9.1) oraz wartość funkcji celu

$$C_{\max}(\pi) = \max_{1 \leqslant j \leqslant n} (C_j + q_j). \tag{9.4}$$

9.3 Zadania zależne

Zależność zadań oznacza w każdym z wyżej wymienionych przypadków istnienie acyklicznej skierowanej relacji $\mathcal{R}$ implikującej ograniczenie $C_i \leqslant S_j$

dla każdej pary $(i,j) \in \mathcal{R}$. Odpowiednie problemy posiadają oznaczenie $1|r_j, prec|C_{\max}$, $1|prec|L_{\max}$ oraz $1|q_j, prec|C_{\max}$. Właściwe algorytmy rozwiązywania, o złożoności $O(n^2)$, można otrzymać tworząc uporządkowanie *topologiczne* odpowiednich wielkości r_j, d_j lub q_j. (Przez uporządkowanie topologiczne liczb r_j, $j \in \mathcal{J}$, z relacją $\mathcal{R}$ rozumiemy permutację π taką, że dla każdej pary $(\pi(i), \pi(j))$, $i < j$, dla której nie istnieje droga w grafie $(\mathcal{J}, \mathcal{R})$ zachodzi $r_{\pi(i)} \leqslant r_{\pi(j)}$.) W praktyce korzysta się jednak z innego podejścia. Relacja $\mathcal{R}$ w sposób naturalny implikuje zależności pomiędzy wartościami r_j, d_j, q_j pozwalające dokonać modyfikacji danych wejściowych dla każdej pary $(i,j) \in \mathcal{R}$, w najprostszym przypadku według poniższych zasad

$$r_j := \max\{r_j, r_i + p_i\}, \tag{9.5}$$

$$q_i := \max\{q_i, q_j + p_j\}, \tag{9.6}$$

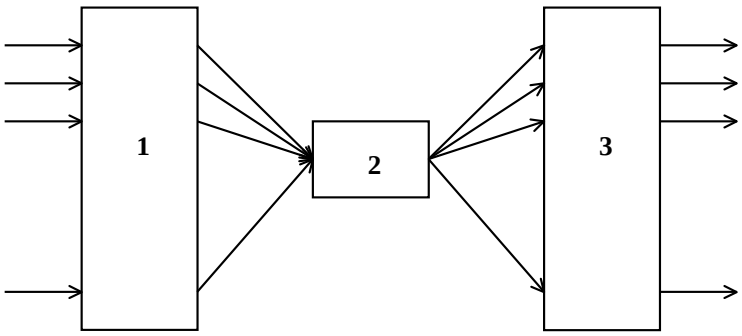
$$d_j := \min\{d_j, d_i - p_i\}. \tag{9.7}$$

bez utraty rozwiązania optymalnego. Modyfikacja ma charakter progresywny, bowiem dokonane zmiany wartości danych mają wpływ na wynik zmian następnych. Zatem jest wykonywana zgodnie z upporządkowaniem liniowym $\mathcal{R}$) od początku dla (9.5) oraz od końca dla (9.6) i (9.7), można ją wykonać w czasie rzędu $O(\max\{n, |\mathcal{R}|\})$. W konsekwencji, każdy z algorytmów opisanych w rozdz. 2.2 zachowuje swą ważność, to znaczy dostarcza rozwiązanie dopuszczalne w sensie $\mathcal{R}$ i optymalne, dla odpowiednich przypadków uwzględniających zadania zależne.

9.4 Czasy przygotowania i dostarczenia

Problem został otrzymany z problemu podstawowego (Rozdz. 9.1) poprzez wprowadzenie niezerowych czasów przygotowawczych r_j implikujących ograniczenie $r_j \leqslant S_j$ oraz czasów dostarczenia zadania $q_j \geqslant 0$. Dostarczenie rozpoczyna się bezpośrednio po zakończeniu wykonywania zadania, przy czym wszystkie zadania mogą byś równocześnie dostarczane. Celem jest wyznaczenie harmonogramu pracy maszyny, który minimalizuje czas w jakim wszystkie zadania zostaną dostarczone. Problem ten jest oznaczany przez $1|r_j, q_j|C_{\max}$.

Problem $1|r_j, q_j|C_{\max}$ jest równoważny problemowi z czasami przygotowawczymi, żądanymi terminami zakończenia oraz kryterium minimalizacji maksymalnego spóźnienia $L_{\max}$ oznaczonego $1|r_j|L_{\max}$ używając notacji z pracy [139]. Istotnie, niech $D = \min_{1 \leqslant j \leqslant n} d_j$. Stąd mamy $q_j = D - d_j \geqslant 0$



Rysunek 9.1: Struktura systemu wytwarzania dla problemu $1|r_j, q_j|C_{\max}$.

oraz

$$L_{\max} = \max_{1 \leqslant j \leqslant n} (C_j - d_j) = \max_{1 \leqslant j \leqslant n} (C_j + q_j) - D = C_{\max} - D \tag{9.8}$$

Problem $1|r_j, q_j|C_{\max}$ jest równoważny problemowi szeregowania zadań na trzech maszynach, z których maszyna 2 jest maszyną o ograniczonej jednostkowej przepustowości z czasami wykonywania zadań p_j, zaś maszyny 1 i 3 są maszynami o nieograniczonej przepustowości z czasami wykonywania r_j oraz q_j odpowiednio. Przyjmujemy, że maszyny o nieograniczonej przepustowości mogą wykonywać dowolnie dużo zadań jednocześnie, zatem zadania na tych maszynach nie muszą być szeregowane (kolejkowane) [224], patrz także Rys. 9.1. Logiczne pojęcie maszyny o nieograniczonej przepustowości może modelować m.in.: (a) fizyczne urządzenia o nieograniczonych możliwościach wykonawczych (np. piec grzewczy), (b) fizyczne urządzenia o ograniczonych możliwościach wykonawczych jednakże nie wymagające szeregowania (np. dzięki znikomo krótkim czasom wykonywaniu zadań na tym urządzeniu, brakowi kolejki, niewystępowaniu konfliktów w dostępie zadań do urządzenia), (c) proces wymagający upływu czasu lecz nie angażujący urządzenia fizycznego (np. dojrzewanie, stygnięcie), (d) zagregowany zbiór maszyn o nieograniczonej przepustowości.

Chociaż wszystkie powyższe sformułowania problemu są wzajemnie równoważne, model $1|r_j, q_j|C_{\max}$ jest używany najczęściej ze względu na *autosymetrię*. Oznacza ona, w tym przypadku, że problem otrzymany przez za-

mianę miejscami wartości r_j z q_j posiada dokładnie taką samą optymalną wartość funkcji celu, osiąganą dla permutacji odwrotnej do tej w problemie przed zamianą.

Problemowi $1|r_j, q_j|C_{\max}$ poświęcono znaczną uwagę w ostatnim dziesięcioleciu ze względu na liczne zastosowania m.in. w szeregowaniu maszyn krytycznych [224], w algorytmach przybliżonych dla problemu gniazdowego [5]), jako dolne ograniczenie dla problemów przepływowych i gniazdowych [43,52].

W pracy [225] pokazano, że problem jest silnie NP-trudny, jednakże dla pewnych przypadków szczególnych istnieją algorytmy wielomianowe.

Oznaczmy przez $C_{\max}(\pi)$ czas, w którym wszystkie zadania zostaną dostarczone dla rozwiązania reprezentowanego permutacją π. Terminy rozpoczęcia S_j i zakończenia C_j wykonywania zadań można znaleźć z zależności rekurencyjnej

$$C_{\pi(j)} = \max\{r_{\pi(j)}, C_{\pi(j-1)}\} + p_{\pi(j)}, \quad j = 1, \ldots, n, \tag{9.9}$$

gdzie $\pi(0) = 0$ oraz $C_0 = 0$. Dodatkowo $S_j = C_j - p_j$, $j = 1, \ldots, n$. Wartość funkcji kryterialnej wynosi

$$C_{\max}(\pi) = \max_{1 \leqslant j \leqslant n} (C_{\pi(j)} + q_{\pi(j)}). \tag{9.10}$$

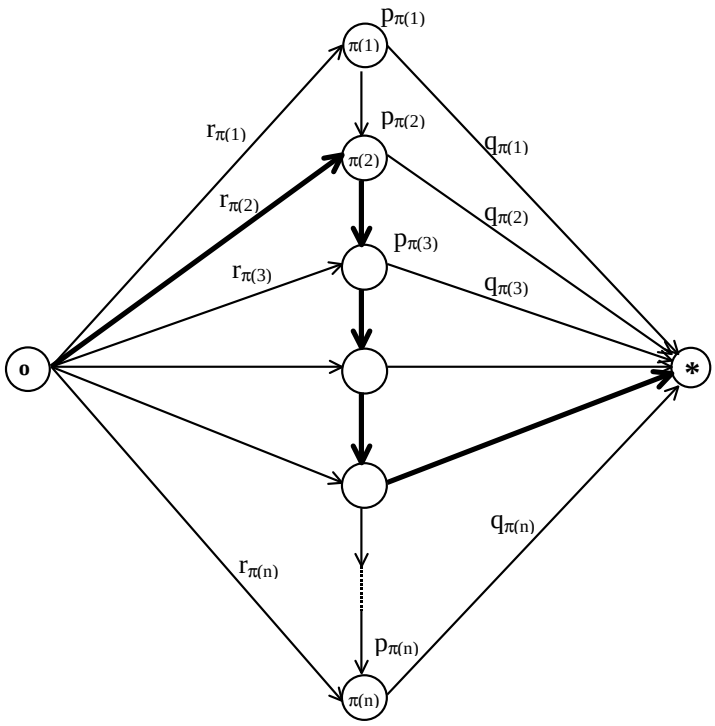
Przy wyprowadzaniu pewnych własności problemu będziemy się także odwoływać do nierekurencyjnej wersji wzoru (9.9)

$$C_{\pi(j)} = \max_{1 \leqslant u \leqslant v \leqslant j} (r_{\pi(u)} + \sum_{s=u}^{v} p_{\pi(s)}) \tag{9.11}$$

oraz odpowiedniej nierekurencyjnej wersji wzoru (9.10)

$$C_{\max}(\pi) = \max_{1 \leqslant u \leqslant v \leqslant n} (r_{\pi(u)} + \sum_{s=u}^{v} p_{\pi(s)} + q_{\pi(v)}) \tag{9.12}$$

Wzory (9.9)–(9.12) posiadają wygodną interpretację grafową. Dla permutacji π konstruujemy graf skierowany $G(\pi) = (\mathcal{J} \cup \{o, *\}, E \cup E(\pi))$, gdzie $\mathcal{J}$ jest zbiorem węzłów powiększonym o dwa węzły sztuczne o (początek) oraz $*$ (koniec), patrz rys. 9.2. Węzły $j \in \mathcal{J}$ posiadają wagę p_j zaś węzły sztuczne – wagę zero. Zbiór krawędzi $E = \bigcup_{j \in \mathcal{J}} \{(o, j), (j, *)\}$ posiada wagi r_j dla krawędzi (o, j) oraz q_j dla krawędzi $(j, *)$ odpowiednio. Zbiór $E(\pi) = \bigcup_{j=1}^{n-1} \{(\pi(j), \pi(j+1))\}$ posiada krawędzie z zerowym obciążeniem. Wartość C_j jest długością najdłuższej drogi prowadzącej do węzła j (wraz z



Rysunek 9.2: Graf $G(\pi)$ dla problemu $1|r_j, q_j|C_{\max}$.

obciążeniem tego węzła) zaś $C_{\max}(\pi)$ jest długością najdłuższej drogi w tym grafie.

Permutację π^*, która minimalizuje $\min_\pi C_{\max}(\pi)$ nazywamy optymalną. Niech $C^* = C_{\max}(\pi^*)$. Każdą parę liczb całkowitych (u, v), $1 \leqslant u \leqslant v \leqslant n$, występujących w prawej stronie wzoru (9.12) będziemy nazywać drogą w π. Para ta całkowicie określa przebieg drogi w odpowiednim grafie $G(\pi)$. Drogę (a, b), dla której

$$a = \min\{u : C_{\max}(\pi) = r_{\pi(u)} + \sum_{s=u}^{b} p_{\pi(s)} + q_{\pi(b)})\} \tag{9.13}$$

będziemy nazywać drogą krytyczną w π. Zadanie $c = \pi(k')$ takie, że $a \leqslant k' < b$, $q_{\pi(k')} < q_{\pi(b)}$ oraz $q_{\pi(k)} \geqslant q_{\pi(b)}$, $k = k' + 1, ..., b$ będzie nazywane zadaniem *interferencyjnym*. Zbiór krytyczny jest definiowany jako $K = \{\pi(k'+1), \ldots, \pi(b)\}$ jeśli k' istnieje oraz $K = \{\pi(a), \ldots, \pi(b)\}$ w przeciwnym przypadku.

Problem $1|r_j, q_j, prec|C_{\max}$ zawierający zadania zależne poprzez relację $\mathcal{R}$, może być rozwiązywany metodami podobnymi jak $1|r_j, q_j|C_{\max}$. Jednak często, dla prostoty rozważań, relację $\mathcal{R}$ eliminuje się z problemu wykonując odpowiednią modyfikację danych wejściowych, analogicznie do opisanej w Rozdz. 9.3. Postępowanie takie nie powoduje utraty rozwiązania optymalnego jednak niesie ze sobą pewne niebezpieczeństwa spowodowane "nieszczelnością" systemu modyfikacji. Pierwszym z nich jest wprowadzenie do zbioru rozwiązań permutacji niedopuszczalnych z punktu widzenia relacji (dla problemu $1|r_j, q_j|C_{\max}$ wszystkie permutacje są dopuszczalne). Drugim - jako konsekwencja pierwszego wymienionego, jedynie w przypadku istnienia wielu permutacji optymalnych – pewna optymalna lecz niedopuszczalna permutacja może zostać zwrócona w wyniku działania algorytmu. O ile pierwszy z wymienionych problemów jest tylko pewną niedogodnością (co najwyżej czas pracy algorytmu zostanie wydłużony), to drugi stanowi już poważny mankament. W niektórych algorytmach (dokładnych i przybliżonych) sposób generacji rozwiązania pozwala "automatycznie" zapobiec drugiemu problemowi już po spełnieniu warunku (9.5) lub (9.6). Dla przeciwdziałania pierwszemu mankamentowi, to jest w celu eliminacji maksymalnie dużej liczby permutacji niedopuszczalnych, wprowadza się bardziej zaawansowane sposoby modyfikacji takie jak na przykład

$$r_j := \max\{r_j, \max_{i \in B_j}(r_i + p_i), r(B_j) + p(B_j)\}, \tag{9.14}$$

$$q_i := \max\{q_i, \max_{i \in A_j}(q_j + p_j), p(A_j) + q(A_j)\}, \tag{9.15}$$

gdzie $B_j = \{i \in \mathcal{J} : \ (i,j) \in \mathcal{R}\}$ jest zbiorem bezpośrednich poprzedników zadania j, zaś $A_j = \{i \in \mathcal{J} : \ (j,i) \in \mathcal{R}\}$ jest zbiorem bezpośrednich następników zadania j. Pozostałe wielkości występujace we wzorach (9.14)–(9.15) są określone następująco: $p(I) = \sum_{s\in I} p_s$, $r(I) = \min_{s\in I} r_s$, $q(I) = \min_{s\in I} q_s$. Złożoność obliczeniowa wykonania modyfikacji jest rzędu $O(\max\{n, |\mathcal{R}|\})$ podobnie jak w Rozdz. 9.3 jednak pracochłonność jest praktycznie większa. W różnych konkretnych algorytmach stosowane są także inne specyficzne, mniej i bardziej złożone, sposoby modyfikacji, które jednakże będą przedstawiane łącznie z każdym opisywanym algorytmem.

9.5 Zadania przerywalne

Dopuszczenie przerywania wykonywania zadań dla problemów opisanych w Rozdz. 9.1 nie zmienia nic w zakresie możliwości ich rozwiązania; odpowiednie algorytmy rozwiązują również te przypadki.

W przypadku NP-trudngo problemu z Rozdz. 9.4 sytuacja jest odmienna. Dopuszczenie przerywania wykonywania zadań jest relaksacją prowadzącą do otrzymania problemu $1|r_j, q_j, pmtn|C_{\max}$ posiadającego algorytm wielomianowy. Możliwość rozwiązania w czasie wielomianowym istnieje już dla zagadnienia nieco ogólniejszego od wymienionego, a mianowicie $1|r_j, prec|f_{\max}$ [27] bez znaczącej zmiany złożoności obliczeniowej. Ponieważ oba problemy mają istotne znaczenie dla budowy i działania bardziej złożonych algorytmów, metoda rozwiązania zostanie opisana szczegółowo dla drugiego z wymienionych jako ogólniejszego. Specjalizowany algorytm dla problemu $1|r_j, q_j, pmtn|C_{\max}$ podano w Rozdz. 9.7.1.

Ze względu na przerywanie wykonywania zadań rozwiązanie nie może być reprezentowane jedynie i jednoznacznie permutacją. Stąd przez uszeregowanie $S = (S_1, \ldots, S_n)$ będziemy rozumieć przyporządkowanie dla każdego zadania ciągu przedziałów czasowych $S_j = ((S_j^i, C_j^i) : \ i = 1, \ldots, l_j)$ w których zadanie to jest wykonywane. Oczywiście dla uzyskania dopuszczalności muszą być spełnione warunki $\sum_{i=1}^{l_j}(C_j^i - S_j^i) = p_j$, $r_j \leqslant S_j^1$, $C_i^{l_i} \leqslant S_j^1$ dla $(i,j) \in R$, oraz maszyna wykonuje co najwyżej jedno zadanie w dowolnej chwili czasowej.

Rozpatrzmy wpierw prostszy problem $1|prec|f_{\max}$ z relacją poprzedzania R. Niech $I \subseteq \mathcal{J}$ oznacza dowolny podzbiór zadań, zaś niech $I' \subseteq I$ będzie zbiorem zadań nie posiadających następników w I. Oznaczmy przez $p(I) = \sum_{j\in I} p_j$, przez $f_{\max}(I;S) = \max_{j\in I} f_j(C_j^{l_j})$ wartość funkcji celu dla zadań ze zbioru I oraz uszeregowania S określonego na tym zbiorze, zaś przez $f^*_{\max}(I) = \min_S f_{\max}(I;S)$ wartość funkcji celu dla uszeregowania optymal-

nego na tym zbiorze. Zachodzą następujące oczywiste nierówności

$$f^*_{\max}(I) \geqslant \min_{j \in I'} f_j(p(I)), \tag{9.16}$$

$$f^*_{\max}(I) \geqslant \max_{j \in I} f^*_{\max}(I \setminus \{j\}). \tag{9.17}$$

Niech $k \in I'$ będzie zadaniem takim, że

$$f_k(p(I)) = \min_{j \in I'} f_j(p(I)). \tag{9.18}$$

Rozważmy klasę uszeregowań z własnością, że k jest wykonywane jako ostatnie w zbiorze I. Dla każdego takiego uszeregowania mamy

$$f_{\max}(I; S) = \max\{f_k(p(I)), f^*_{\max}(I \setminus \{k\})\} \leqslant f^*_{\max}(I). \tag{9.19}$$

Stąd wynika, że istnieje uszeregowanie optymalne w którym zadanie k występuje na ostatniej pozycji w I. Rozpoczynając od $I = \mathcal{J}$ oraz powtarzając powyższe rozumowanie dla $I = \mathcal{J} \setminus \{k\}$ oraz kolejno dla coraz mniejszych zbiorów otrzymamy optymalne uszeregowanie dla całego zbioru $\mathcal{J}$ w czasie $O(n^2)$.

Zanalizujmy dalej nieco ogólniejszy problem $1|r_j|f\max$. Bez straty ogólności rozważań możemy założyć, że zadania są indeksowane zgodnie z niemalejącymi wartościami r_j; w przeciwnym przypadku przenumerowanie może być wykonane w czasie $O(n \log n)$. Zauważmy wpierw, że na skutek dopuszczenia przerwań, w dowolnej chwili czasu maszyna nie może być bezczynna jeśli tylko istnieją zadania gotowe do wykonywania. Stąd czas, w którym zakończone zostaną wszystkie zadania wynika z rozwiązania problemu $1|r_j|C_{\max}$. Uszeregowanie zadań według niemalejących wartości r_j implikuje naturalne rozbicie na *bloki* $B_1, \ldots, B_p$. Blok $B \subseteq \mathcal{J}$ jest definiowany jako minimalny podzbiór zadań wykonywanych bez przestoju maszyny od chwili $S(B) = \min_{j \in B} r_j$ aż do chwili $C(B) = S(B) + p(B)$ taki, że każde zadanie $j \notin B$ jest albo zakończone nie później niż $S(B)$ (tj. $C_j^{l_j} \leqslant S(B)$) lub jest niedostępne przed $C(B)$ (tj. $r_j \geqslant C(B)$). W celu minimalizacji $f_{\max}$ możemy rozważać każdy blok oddzielnie. Zatem w mocy pozostają wzory (9.16) (9.17) dla $I = B$ (ponieważ $R = \emptyset$ zatem w tym przypadku $B' = B$). Dalej, niech $k \in B$ będzie zadaniem określonym analogicznie jak w (9.18). Rozważmy klasę uszeregowań dla bloku B z własnością, że k jest wykonywane wtedy gdy nie ma dostępnego innego zadania. Każde tak określone uszeregowanie składa się z dwóch komplementarnych części: (a) optymalne uszeregowanie dla zbioru $B \setminus \{k\}$, tworzące pewną liczbę pod-bloków $B_1, \ldots, B_b$ tego zbioru, (b) uszeregowanie zadania k, które jest dane przez

$[S(B), C(B)] \setminus \bigcup_{i=1}^{b} [S(B_i), C(B_i)]$; zauważmy, że zadanie k jest kończone w chwili $C(B)$. Dla każdego takiego uszeregowania prawdziwy jest wzór (9.19) dla $I = B$. Stąd wynika, że istnieje uszeregowanie optymalne bloku B, w którym zadanie k jest wykonywane w podany wyżej sposób.

Odpowiedni algorytm dla problemu $1|r_j|f_{\max}$ jest następujący. Określamy bloki początkowe, co można zrealizować w czasie $O(n)$. Następnie dla każdego bloku B powtarzamy ciąg postępowań: wybieramy zadanie k zgodnie z warunkiem (9.18) dla $I' = B' = B$, określamy strukturę blokową zbioru $B \setminus \{k\}$ oraz określamy uszeregowanie zadania k w opisany wyżej sposób. Wymaga to czasu $O(|B|)$. Powtarzamy to postępowanie dla wszystkich bloków. Złożoność obliczeniowa algorytmu jest $O(n^2)$. Algorytm powoduje co najwyżej $n - 1$ przerwań. Fakt ten można pokazać przez indukcję. łatwo sprawdzić, że jest on prawdziwy dla $n = 1$. Przypuśćmy, że jest on prawdziwy dla bloków o rozmiarze mniejszym niż $|B|$. Uszeregowanie dla bloku B ma co najwyżej $|B_i| - 1$ przerwań dla każdego podbloku B_i, $i = 1, \ldots, b$, i co najwyżej b przerwań dla wybranego zadania k. Zatem całkowita liczba przerwań jest nie większa niż $\sum_{i=1}^{b}(|B_i| - 1) + b = |B| - 1$ co kończy dowód.

Przejście do problemu ogólnego $1|r_j, prec|f_{\max}$ jest naturalne. Wpierw implementujemy relację R wprowadzając modyfikację wartości r_j zgodnie ze wzorem (9.5). Dalej stosujemy postępowanie analogiczne jak w problemie $1|r_j|f_{\max}$, stosując wzory (9.16) – (9.18) dla $I = B$, $I' = B'$. Odpowiedni algorytm również posiada złożoność $O(n^2)$.

9.6 Dolne ograniczenia problemu ogólnego

Ponieważ problem $1|r_j, q_j|C_{\max}$ jest NP-trudny, istnieje potrzeba wyznaczania dolnego ograniczenia optymalnej wartości funkcji celu zarówno w schemacie B&B jak i dla oszacowań jakości algorytmów przybliżonych. Istnieje wiele metod konstrukcji dolnych ograniczeń dla rozważanego problemu opartych o różne formy relaksacji ograniczeń i relaksacji funkcji celu. Metody te są oceniane poprzez ich złożoność obliczeniową oraz dokładność szacowania funkcji celu, cechy szczególnie ważne dla zastosowań w schemacie B&B. Ponieważ optymalna wartość funkcji celu jest zwykle nieznana, dokładność szacowania ocenia się zwykle porównując metody wzajemnie między sobą w celu wykazania ich równoważności bądź dominacji.

Zdefiniujmy dla $I \subseteq \mathcal{J}$ następujące pojęcia

$$r(I) = \min_{i \in I} r_i, \; p(I) = \sum_{i \in I} p_i, \; q(I) = \min_{i \in I} q_i \tag{9.20}$$

oraz

$$h(I) = r(I) + p(I) + q(I) \tag{9.21}$$

gdzie $r(\emptyset) = p(\emptyset) = q(\emptyset) = h(\emptyset) = 0$.

Proste dolne ograniczenia

Pierwsze, najprostsze i historycznie najstarsze dolne ograniczenie jest skojarzone z zadaniami i określone wzorem

$$LB_0 = \max_{1 \leqslant j \leqslant n} (r_j + p_j + q_j). \tag{9.22}$$

Złożoność obliczeniowa odpowiedniego algorytmu jest $O(n)$.

Drugie dolne ograniczenie zostało otrzymane poprzez relaksację ograniczenia $r_j \leqslant S_j$ do ograniczenia słabszego $r(\mathcal{J}) \leqslant S_j$. Prowadzi to do problemu szeregowania postaci $1|r_j = r(\mathcal{J}), q_j|C_{\max}$ posiadającego algorytm wielomianowy o złożoności obliczeniowej $O(n \log n)$. Istotnie niech $C^{*q}_{\max}$ będzie optymalną wartością funkcji celu problemu $1|q_j|C_{\max}$ (patrz Rozdz. 9.2). Odpowiednia wartość dolnego ograniczenia wynosi $LB_{1r} = r(\mathcal{J}) + C^{*q}_{\max}$.

Trzecie dolne ograniczenie zostało otrzymane poprzez relaksację funkcji celu $\max_{1 \leqslant j \leqslant n}(C_j + q_j)$ do słabszej postaci $\max_{1 \leqslant j \leqslant n}(C_j + q(\mathcal{J})) = \max_{1 \leqslant j \leqslant n} C_j + q(\mathcal{J})$. Prowadzi do problemu szeregowania postaci $1|r_j, q_j = q(\mathcal{J})|C_{\max}$ o wielomianowej złożoności obliczeniowej rzędu $O(n \log n)$. Istotnie niech $C^{*r}_{\max}$ będzie optymalną wartością funkcji celu problemu $1|r_j|C_{\max}$, patrz Rozdz. 9.2. Odpowiednia wartość dolnego ograniczenia wynosi $LB_{1q} = C^{*r}_{\max} + q(\mathcal{J})$.

Relaksacja liczby zadań

W pracy [51] zostało pokazane, że dla każdego $I \subseteq \mathcal{J}$ zachodzi

$$C_{\max}(\pi^*) \geqslant h(I). \tag{9.23}$$

Istotnie rozpatrzmy następujące kolejne relaksacje problemu $1|r_j, q_j|C_{\max}$. Wpierw relaksujemy liczbę zadań rozpatrując tylko pewien ich podzbiór $I \subseteq \mathcal{J}$. Następnie relaksujemy ograniczenie $r_j \leqslant S_j$ wprowadzając w jego miejsce ograniczenie słabsze $r(I) \leqslant S_j$ oraz funkcję celu $\max_{j \in I}(C_j + q_j)$ do postaci słabszej $\max_{j \in I} C_j + q(I)$, podobnie jak w rozdziale poprzednim. Ostatecznie otrzymujemy problem postaci $1|r_j = r(I), q_j = q(I)|C_{\max}$ ze zbiorem zadań I, dla którego optymalna wartość funkcji celu wynosi $r(I)+p(I)+q(I)$ (patrz także Rozdz. 9.1). Złożoność tego ograniczenia jest $O(|I|)$.

Korzystając z (9.23) możemy natychmiast przyjąć

$$LB_2 = \max_{I \subseteq \mathcal{J}} h(I) \tag{9.24}$$

choć bezpośrednio z (9.24) wynika wykładnicza złożoność obliczeniowa odpowiedniego algorytmu. Zauważmy, że ograniczenie (9.22) jest szczególnym przypadkiem (9.24) takim, że dla wszystkich $I \subseteq \mathcal{J}$ zachodzi $|I| = 1$. W praktyce dość często stosuje się ograniczenie (9.23) dla pewnego specyficznego zbioru zadań.

Ograniczenia przerywalne

Ponieważ problem $1|r_j, q_j, pmtn|C_{\max}$ posiada algorytm wielomianowy, zatem jego optymalna wartość funkcji celu $C^*_{\max,pmtn}$ dostarcza również dolnego ograniczenia. W pracy [51] stwierdzono, że zachodzi

$$LB_2 = C^*_{\max,pmtn} \tag{9.25}$$

zaś poprawny dowód tego faktu podano w pracy [270].

Jak do tej pory jest to najdokładniejsze proste dolne ograniczenie znane obecnie [255].

Ograniczenia zaawansowane

W pracy [266] pokazano dolne ograniczenie, które może być mocniejsze niż (9.23). Dla dowolnego $I \subseteq \mathcal{J}$ oraz dowolnego $m \in (\mathcal{J} \setminus I) \cup \{0\}$ definiujemy wartość $H(I,m)$ następująco

$$H(I,m) = \begin{cases} h(I) + \min\{R_m, p_m, Q_m\} & gdy \quad |I| > 1,\ m \neq 0, \\ h(I) + \min\{R_m, Q_m\} & gdy \quad |I| = 1,\ m \neq 0, \\ h(I) & gdy \quad m = 0 \end{cases} \tag{9.26}$$

gdzie $R_m = [r_m + p_m - r(I)]^+$, $Q_m = [p_m + q_m - q(I)]^+$ oraz $[x]^+ = \max\{x, 0\}$.

Udowodnimy, że zachodzi $C_{\max}(\pi^*) \geqslant H(I,m)$ dla $I \subseteq \mathcal{J}$ oraz $m \in (\mathcal{J} \setminus I) \cup \{0\}$. Istotnie, jeśli $m = 0$ to prawdziwość oszacowania (9.26) wynika z (9.23). Zatem załóżmy $m \neq 0$ oraz rozpatrzmy dwa rozłączne i wyczerpujące przypadki.

Przypadek "$|I| > 1$". Rozważmy problem postaci $1|r'_j, q'_j|C_{\max}$ ze zbiorem zadań $I \cup \{m\}$, terminami gotowości $r'_j = r(I)$, $j \in I$, $r'_m = r_m$, czasami dostarczania $q_j = q(I)$, $j \in I$, $q'_m = q_m$ oraz czasami wykonywania $p'_j = p_j$, $j \in I \cup \{j\}$. Oznaczmy przez C' optymalną wartość funkcji

celu problemu pomocniczego. Ponieważ jest on relaksacją problemu oryginalnego, zatem zachodzi $C^* \geqslant C'$. W optymalnym rozwiązaniu problemu pomocniczego zadanie m musi być wykonywane wg jednej z trzech możliwości: (1) przed zbiorem I, (2) za zbiorem I, (3) wewnątrz zbioru I. Stąd otrzymujemy

$$C^* \geqslant C' \geqslant \min\{\max\{r_m+p_m, r(I)\}+p(I)+q(I), r(I)+p(I)+p_m+q(I),$$
$$r(I)+p(I)+\max\{p_m+q_m, q(I)\}\} \tag{9.27}$$

co implikuje (9.26).

Przypadek "$|I| = 1$". Niech $I = \{i\}$. Zadanie m może być szeregowane tylko przed lub za zadaniem i, zatem otrzymujemy

$$C^* \geqslant \min\{\max\{r_m+p_m, r_i\}+p_i+q_i, r_i+p_i+\max\{p_m+q_m, q_i\}\} \tag{9.28}$$

co implikuje (9.26) również i tym przypadku.

Jest oczywistym, że $H(I,m) \geqslant h(I)$ dla dowolnego $m \in (\mathcal{J} \setminus I) \cup \{0\}$ oraz $I \subseteq \mathcal{J}$. Ogólny wzór (9.26) może dostarczać wielu dolnych ograniczeń, przykładowo dla $I \subset \mathcal{J}$ oraz $m \in \mathcal{J} \setminus I$ mamy

$$\begin{aligned} C^* \geqslant H(I,m) &\geqslant h(I) + \min\{r_m + p_m - r(I), p_m, p_m + q_m - q(I)\} \\ &\geqslant h(I) + \min\{p_m - r(I), p_m, p_m - q(I)\} \\ &= p(I) + p_m + \min\{r(I), q(I)\}. \end{aligned} \tag{9.29}$$

Z kolei stosując (9.26) dla $I = \{i\}$ oraz $m \in \mathcal{J} \setminus I$ w połączeniu z oczywistą nierównością $C^* \geqslant p_m + \max\{r_m, q_m\}$ otrzymujemy

$$C^* \geqslant p_m + \max\{\min\{r_i, q_i\}, r_m, q_m\}. \tag{9.30}$$

Sosując (9.26) dwukrotnie dla $I = \{i\}$ oraz $I = \{j\}$ dla tego samego $m \in \mathcal{J} \setminus \{i,j\}$ otrzymamy

$$C^* \geqslant p_m + \max\{\min\{r_i, q_i\}, \min\{r_j, q_j\}\}. \tag{9.31}$$

9.7 Algorytmy przybliżone problemu ogólnego

Algorytm przybliżony generuje permutację oznaczoną dalej przez π^A. Dla ułatwienia opisu zdefiniujemy częściową permutację na zbiorze $I \subseteq \mathcal{J}$, która będzie oznaczane przez $(\pi)_I$.

j	1	2
r_j	0	ϵ
p_j	1	ϵ
q_j	0	1

Tabela 9.1: Przykład; ϵ jest pewną małą liczbą dodatnią

9.7.1 Algorytm 2-aproksymacyjny

Najprostszą technikę budowy takiego algorytmu zapożyczono z Rozdz. 9.2. Jako rozwiązanie przybliżone można przyjąć permutację π^R wygenerowaną dla problemu zrelaksowanego $1|r_j|C_{\max}$ lub permutację π^Q wygenerowaną dla problemu zrelaksowanego $1|q_j|C_{\max}$. W obu przypadkach otrzymamy algorytm dostarczający dla problemu $1|r_j, q_j|C_{\max}$ uszeregowania o długości nie większej niż 2 razy długość uszeregowania optymalnego (tzw. algorytm 2 -aproksymacyjny). Istotnie, mamy

$$\begin{aligned} C_{\max}(\pi^R) &= \max_{1\leqslant u\leqslant v\leqslant n}(r_{\pi^R(u)} + \sum_{s=u}^{v} p_{\pi^R(s)} + q_{\pi^R(v)}) \\ &\leqslant \max_{1\leqslant u\leqslant v\leqslant n}(r_{\pi^R(u)} + \sum_{s=u}^{v} p_{\pi^R(s)}) + \max_{1\leqslant u\leqslant v\leqslant n} q_{\pi^R(v)} \\ &\leqslant \max_{1\leqslant u\leqslant n}(r_{\pi^R(u)} + \sum_{s=u}^{n} p_{\pi^R(s)}) + \max_{1\leqslant v\leqslant n} q_v \\ &\leqslant C_{\max}(\pi^*) + C_{\max}(\pi^*) = 2C_{\max}(\pi^*) \end{aligned} \tag{9.32}$$

bowiem zarówno $\max_{1\leqslant u\leqslant n}(r_{\pi^R(u)} + \sum_{s=u}^{n} p_{\pi^R(s)})$ (rozwiązanie optymalne problemu $1|r_j|C_{\max}$) jak i $\max_{1\leqslant v\leqslant n} q_v$ stanowią dolne ograniczenie rozwiązania optymalnego $C_{\max}(\pi^*)$ problemu $1|r_j q_j|C_{\max}$. W podobny sposób można wykazać, że π^Q dostarcza rozwiązania 2 -aproksymacyjnego.

Przykład liczbowy potwierdzający obcisłość oszacowania (9.32) przedstawiony jest w Tablicy 9.7.1. Istotnie, $\pi^R = (1, 2)$ oraz $C_{\max}(\pi^R) = 2 + \epsilon$, $\pi^* = (2, 1)$ oraz $C_{\max}(\pi^*) = 2\epsilon + 1$. Zatem iloraz $C_{\max}(\pi^R)/C_{\max}(\pi)$ dąży do 2 jeśli ϵ dąży do zera.

W pracy [326] zaproponowano algorytm (S) który stosuje oczywistą intuicyjnie regułę, znaną jako uogólniona reguła Jacksona: jeżeli maszyna jest wolna oraz co najmniej jedno zadanie jest gotowe do wykonywania, należy

j	1	2	3	4	5	6	7
r_j	10	13	11	20	30	0	30
p_j	5	6	7	4	3	6	2
q_j	7	26	24	21	8	17	0

Tabela 9.2: Przykład liczbowy dla algorytmu S

skierować do wykonywania dostępne zadanie najpilniejsze, to znaczy to z najdłuższym czasem dostarczania. Udowodniono, że algorytm S jest także 2 -aproksymacyjny [199]. Przykład zamieszczony w Tablicy **??** pokazuje, że oszacowanie to jest obcisłe bowiem algorytm S także wygeneruje permutację $\pi^S = (1, 2)$. Ogólny schemat algorytmu podano poniżej.

Algorithm S

(1) Podstaw $i = 1$, $\sigma = ()$, $U = \emptyset$, $t = \min_{1 \leqslant j \leqslant n} r_j$.

(2) W chwili t wybierz zadanie $j^* \in \mathcal{J} \setminus U$ takie, że $q_{j^*} = \max_{j \in \mathcal{J} \setminus U;\, r_j \leqslant t} q_j$

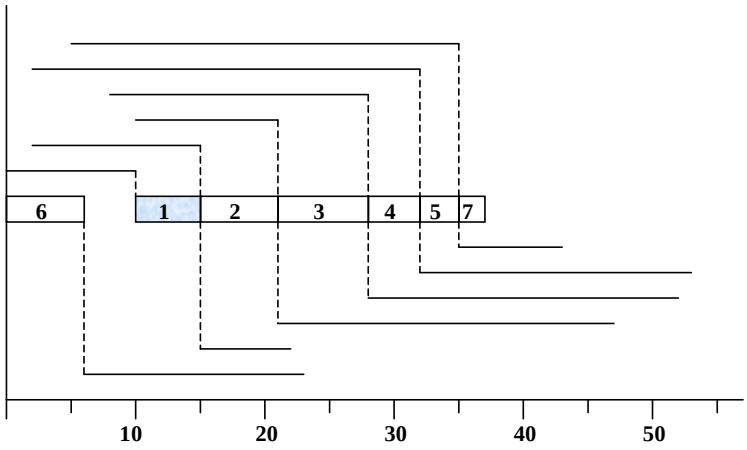
(3) Podstaw $U := U \cup \{j^*\}$, $i := i + 1$, $\sigma(i) = j^*$, $S_{j^*} = t$, oraz $t := \max\{t + p_{j^*}, \min_{j \in \mathcal{J} \setminus U} r_j\}$

(4) Jeśli $U = \mathcal{J}$ to podstaw $\pi^S := \sigma$ i STOP. Inaczej przejdź do Kroku (2).

W każdym kroku algorytmu, zbiór U zawiera zadania uszeregowane, tworzące permutację częściową σ o długości i. Zadania niuszeregowane $\mathcal{J} \setminus U$ tworzą kolejkę, z której pobierane są zadania do obsługi, zgodnie z ich dostępnością i pilnośćią.

Rozważmy przykład problemu z danym w Tablicy 9.7.1. Zastosowanie Algorytmu S dostarcza permutacji $\pi^S = (6, 1, 2, 3, 4, 5, 7)$ oraz wektora terminów rozpoczęcia zadań $S = (10, 15, 21, 28, 32, 0, 35)$ i wartości funkcji celu $C_{\max}(\pi^S) = 53$. Harmonogram pracy maszyny przedstawiono na Rys. 9.3.

Pierwsza implementacja Algorytmu S działająca w czasie $O(n \log n)$ została podana w pracy [51] w oparciu o ideę kopca dwumianowego [69]. Istnieje jednak prostsza implementacja wykorzystująca "zwykły" kopiec binarny [69], strukturę danych polecaną rutynowo dla realizacji statycznej kolejki priorytetowej. W obu implementacjach przyjmujemy, bez straty ogólności rozważań, że zadania są numerowane zgodnie z niemalejącymi wartościami r_j; w przeciwnym przypadku należy zadania przenumerować, można to wykonać w czasie $O(n \log n)$. W technice kopca binarnego *kolejka do stanowiska*, to



Rysunek 9.3: Wykres Gantta dla π^S w przykładzie problemu $1|r_j, q_j|C_{\max}$.

znaczy zbiór zadań nieuszeregowanych i gotowych do wykonywania w chwili t, jest przechowywana w formie kopca zorganizowanego według nierosnących wartości q_j. Kolejka ta jest aktualizowana $2n$ razy, odpowiednio do dodania i usunięcia każdego zadania z kolejki, przy czym każdą z tych operacji można zrealizować na kopca w czasie ograniczonym przez $O(\log n)$. Ponieważ liczba analizowanych chwil t jest rzędu $O(n)$, zatem końcowa złożoność obliczeniowa algorytmu jest $O(n \log n)$.

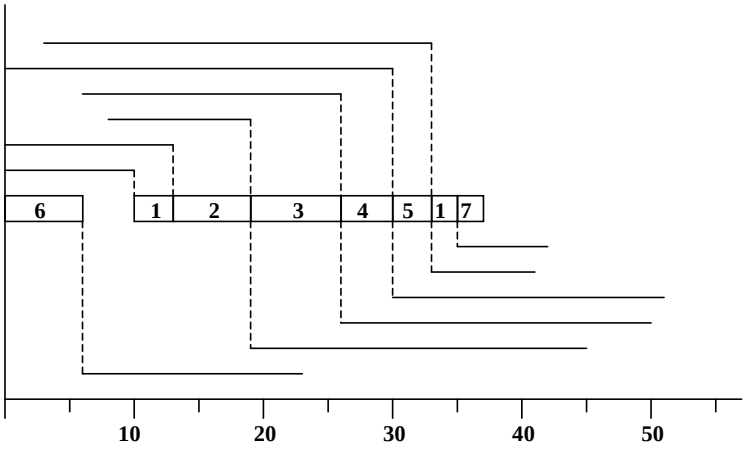
Możliwe jest zastosowanie Algorytmu S dla przypadku zadań zależnych powiązanych relacją R. W tym celu należy wykonać wstępną modyfikację danych wejściowych a następnie wykonać właściwy Algorytm S dla tak zmodyfikowanych danych. Modyfikacja odbywa się dla każdej pary $(i, j) \in R$, progresywnie, według zasady

$$r_j := \max\{r_i, r_j\}, \tag{9.33}$$

$$q_i := \max\{q_i, q_j + p_j\}. \tag{9.34}$$

Otrzymane rozwiązanie jest dopuszczalne bowiem ze względu na warunek (9.33) zadanie j będzie dostępne nie wcześniej niż zadanie i, zaś w przypadku ich jednoczesnego występowania w zbiorze zadań nieuszeregowanych $\mathcal{J} \setminus U$ warunek (9.34) zapewnia, że zadanie i będzie uszeregowane przed zadaniem j.

Algorytm S może być także wykorzystany, po niewielkiej modyfikacji, do rozwiązywania problemu $1|r_j, q_j, pmtn|C_{\max}$. Obowiązuje w tym przypadku



Rysunek 9.4: Wykres Gantta w przykładzie problemu $1|r_j, q_j, pmtn|C_{\max}$.

także uogólniona reguła Jacksona w wersji dopuszczającej przerywanie wykonywania zadań oraz "zwracanie" niewykonanej jeszcze części zadania do kolejki. Przerwanie następuje każdorazowo gdy w kolejce pojawia się zadanie o wyższej pilności niż to aktualnie wykonywane. Zapis algorytmu w tym przypadku jest nieco bardziej złożony bowiem rozwiązanie nie może być reprezentowane permutacją, patrz. Rozdz. 9.5. Można sprawdzić, że algorytm ten również posiada złożoność obliczeniową $O(n \log n)$. Dla przykładu podanego w Tablicy 9.7.1 odpowiednie rozwiązanie przerywalne zostało przedstawione na rys. 9.4. Optymalna wartość funkcji celu wynosi $C^*_{\max,pmtn} = 51$.

9.7.2 Algorytm 3/2-aproksymacyjny

Algorytm P [291] generuje ciąg t, $1 \leqslant t \leqslant u$ permutacji $\pi_1, \ldots, \pi_t$, startując z permutacji wyznaczonej algorytmem S, tzn. $\pi_1 = \pi^S$. Dla każdej permutacji π_i z ciągu wyznaczane są odpowiednie zadania b oraz c. Jeśli tylko c istnieje to generowana jest następna permutacja π_{i+1} poprzez wykonanie podstawienia $r_c = r_{\pi(b)}$ i ponowne zastosowanie algorytmu S. Postępowanie jest kontynuowane tak długo jak zadanie c istnieje ale nie więcej niż u razy, gdzie u jest pewną stałą. Algorytm zwraca jako wynik najlepszą permutację z wygenerowanego ciągu.

Algorytm P dla $u < n$ jest algorytmem 2 -aproksymacyjnym, zaś dla $u \geqslant n$ algorytmem 3/2 -aproksymacyjnym. Pierwszy z wymienionych fak-

j	1	2	3	...	$n-1$	n
r_j	0	ϵ	ϵ	...	ϵ	ϵ
p_j	1	ϵ	ϵ	...	ϵ	ϵ
q_j	0	$1+2(n-2)\epsilon$	$1+2(n-3)\epsilon$	...	$1+2\epsilon$	1

Tabela 9.3: Przykład liczbowy dla algorytmu P; ϵjest pewną małą liczbą.

tów wynika z przykład liczbowego zamieszczonego w Tablicy 9.7.2. Istotnie, dla podanego przykładu otrzymujemy $\pi_1 = \pi^S = (1, 2, \ldots, n)$, $C_{\max}(\pi^S) = 2 + 2(n-2)\epsilon + \epsilon$, oraz następnie $(a, b) = (1, 2)$, $\pi(b) = 2$, $c = 1$. Wykonując jeden krok Algorytmu P otrzymujemy $\pi_2 = (2, 1, 3, 4, \ldots, n$, $C_{\max}(\pi_2) = 2+2(n-1)\epsilon$ oraz $(a, b) = (1, 2)$, $\pi(b) = 3$, $c = 1$. Wykonując kolejne kroki Algorytmu P dla $i = 1, \ldots, n-1$ otrzymujemy $\pi_i = (2, 3, \ldots, i, 1, i+1, \ldots, n)$, $C_{\max}(\pi_i) = 2 + O(n\epsilon)$ oraz zawsze zadanie 1 jest zadaniem interferencyjnym. Łatwo sprawdzić, że rozwiązaniem optymalnym jest permutacja $\pi^* = (2, 3, \ldots, n-1, n, 1)$ oraz $C_{\max}(\pi^*) = 1 + n\epsilon$. Stąd każdy iloraz $C_{\max}(\pi^i)/C_{\max}(\pi^*) \to 2$ jeśli $\epsilon \to 0$. Fakt drugi, potwierdzający oszacowanie 3/2, wynika z dowodu przedstawionego w pracy [291]. Na koniec zauważmy, że dla $u = n$ złożoność obliczeniowa Algorytmu P rzędu $O(n^2 \log n)$.

Rozszerzenie Algorytmu P na przypadek zadań zależnych wymaga nieco więcej zabiegów niż dla Algorytmu S. Każdorazowo gdy zadanie interferencyjne c ma zmieniany termin dostępności wg zasady $r_c := r_{\pi(b)}$ dokonujemy również zmiany terminów dostępności wszystkich zadań j, które występują za zadaniem c w relacji poprzedzania R. To gwarantuje, że kolejno otrzymane rozwiązanie przez zastosowanie Algorytmu S jest dopuszczalne. Również ta zmodyfikowana wersja Algorytmu P (będziemy ją oznaczać dalej P$'$) jest algorytmem 3/2 -aproksymacyjnym.

9.7.3 Algorytm 4/3-aproksymacyjny

Bazując na Algorytmie P w pracy [159] zaproponowano algorytm (HS) będący 4/3 -aproksymacyjnym algorytmem działającym w czasie $O(n^2 \log n)$.

Algorytm HS rozważa oprócz wersji podstawowej, także wersję odwrotną (symetryczną) problemu. Stosując algorytm P, generowane jest co najwyżej $u = 2n-3$ permutacji dla problemu podstawowego oraz co najwyżej $u = 2n-3$ permutacji dla problemu odwrotnego. Szczegółowa postać tego algorytmu przedstawiona jest poniżej.

j	1	2	3	4	5
r_j	0	ϵ	$1+\epsilon$	$2+\epsilon$	2
p_j	1	1	ϵ	ϵ	1
q_j	0	2	$2+\epsilon$	$1+\epsilon$	ϵ

Tabela 9.4: Przykład liczbowy dla algorytmu HS

1. Jeśli nie istnieją w zbiorze $\mathcal{J}$ zadania k oraz l takie, że $p_k, p_l > p(\mathcal{J})/3$, lub istnieje dokładnie jedno zadanie k takie, że $p_k > p(\mathcal{J})/3$, to zastosuj Algorytm P do problemu prostego i odwrotnego z $u = n$ oraz zwróć najlepsze z wygenerowanych rozwiązań.

2. Jeśli istnieją w zbiorze $\mathcal{J}$ zadania k oraz l takie, że $p_k, p_l > p(\mathcal{J})/3$ to wykonaj dwa następujące kroki: (1) dodaj ograniczenie "k poprzedza l" oraz zastosuj Algorytm P$'$ dla problemu prostego i odwrotnego dla $u = 2n - 3$, (2) dodaj ograniczenie "l poprzedza k" oraz zastosuj Algorytm P$'$ dla problemu prostego i odwrotnego dla $u = 2n - 3$. Zwróć najlepsze z wygenerowanych rozwiązań.

W celu pokazania osiągalności oszacowania rozważmy przykład problemu z danym w Tablicy 9.7.3. Rozpatrzmy szczegółowo zachowanie się Algorytmu HS. Ze względu na symetrię danych przykładu wystarczy rozważyć problem prosty, bowiem problem odwrotny dostarczy takich samych rezultatów. Przykład wymaga zastosowana kroku (1) algorytmu. Rozpoczynamy od permutacji $\pi_1 = (1, 2, 3, 4, 5)$, $C_{\max}(\pi_1) = 4 + 2\epsilon$, $(a, b) = (2, 3)$, $c = 2$. Następnie wprowadzamy modyfikację w oparciu o zadanie interferencyjne przyjmując $r_2 = r_1 = 1 + \epsilon$. Ponowne zastosowanie Algorytmu S dostarcza $\pi_2 = (1, 3, 2, 4, 5)$, $C_{\max}(\pi_2) = 4 + 2\epsilon$ oraz występuje brak zadania interferencyjnego, co zatrzymuje Algorytm P. Można sprawdzić, że rozwiązaniem optymalnym jest $\pi^* = (2, 3, 1, 4, 5)$ oraz $C_{\max}(\pi^*) = 3 + 4\epsilon$. Stąd iloraz $C_{\max}(\pi^{HS})/C_{\max}(\pi^*) \to 4/3$ jeśli $\epsilon \to 0$.

Algorytm HS można uogólnić, podobnie jak P, na przypadek występowania zadań zależnych $(1|r_j, q_j, prec|C_{\max})$. Pominąwszy złożone schematy aproksymacyjne opisne w Rozdz. 9.8, jest to najlepszy algorytm aproksymacyjny znany obecnie.

9.7.4 Szybki algorytm 3/2-aproksymacyjny

Podejścia prezentowane w Rozdz. 9.7.2 – 9.7.3 wprowadzają ocenę jakości jako wtórną do algorytmu, tzn. najpierw zaproponowano algorytm a następnie wykonano jego analizę. Jednakże problem projektowania algorytmu można postawić odmiennie: należy wyznaczyć minimalny zbiór permutacji podlegających sprawdzeniu tak aby zagwarantować dokładność 3/2 (lub 4/3) przy założeniu, że każda permutacja jest generowana w niekosztownym czasie co najwyżej $O(n \log n)$. Jeśli ten zbiór zawiera permutację π^S, to ze względu na wartość ilorazu C^S/C^* bliską 2, musimy dołączyć co najmniej jedną dodatkową permutację.

Takie właśnie podejście zaprezentowano w pracy [266], gdzie opisano algorytm 3/2-aproksymacyjny, o złożoności obliczeniowej $O(n \log n)$. Algorytm ten łączy rozszerzoną regułę Jacksona z regułą Johnsona znaną dla dwumaszynowego problemu przepływowego. Algorytm ten bada dwie permutacje w tym π^S. Kombinując ten algorytm z dowolnym innym zawsze otrzymamy algorytm 3/2-aproksymacyjny.

Algorytm NS

Krok 1. Wyznacz π^S używając algorytmu S oraz zadanie interferencyjne c w π_S. Jeśli c nie istnieje to zwróć $\pi^{NS} = \pi^S$.

Krok 2. Wyznacz zbiory $A = \{i \in \mathcal{J} \setminus \{c\} : \quad r_i \leqslant q_i\}$, $B = \{i \in \mathcal{J} \setminus \{c\} : \quad r_i > q_i\}$ oraz permutację π^A poprzez uporządkowanie zadań ze zbioru A wg niemalejących wartości r_i i permutację π^B otrzymaną poprzez uporządkowanie zadań wg nierosnących wartości q_i. Utwórz permutację $\pi^{AB} = \pi^A c \pi^B$.

Krok 3. Zwróć lepszą z permutacji π^{AB}, π^S, tzn. podstaw $\pi^{NS} = \pi^{AB}$ jeśli $C_{\max}(\pi^{AB}) < C_{\max}(\pi^S)$ zaś $\pi^{NS} = \pi^S$ w przeciwnym przypadku.

Pokażemy dalej, że algorytm NS jest 3/2-aproksymacyjny, przy czym oszacowanie to jest dokładne. Pokazano, że jeśli dla π^S zadanie c nie istnieje to permutacja jest optymalna tj. $C^S = C^*$; w przeciwnym przypadku mamy $C^S \leqslant p_c + C^*$. Zatem jeśli algorytm H powrócił bezpośrednio z Kroku 1 to π^H jest optymalna. W przeciwnym przypadku, jeśli $p_c \leqslant C^*/2$ to $C^H \leqslant C^S \leqslant (3/2)C^*$. Zatem rozważmy ostatni przypadek: c istnieje $p_c > C^*/2$. Pokażemy, że $C^{AB} \leqslant (3/2)C^*$. Bez straty ogólności możemy założyć, że $\pi^{AB} = (1, 2, \ldots, n)$ oraz że zadanie interferencyjne w π^S nazywa się także

c. Niech (a, b) będzie drogą krytyczną w π^{AB}. Ze wzoru (9.12) mamy

$$C^{AB} = r_a + \sum_{s=a}^{b} p_s + q_b. \tag{9.35}$$

Możliwe są przy tym trzy rozłączne i wyczerpujące przypadki.

Przypadek "$a = c$, $b = c$". Korzystając z (9.35) oraz z (9.23) dla $I = \{c\}$ otrzymujemy

$$C^{AB} = r_c + p_c + q_c \leqslant C^*. \tag{9.36}$$

Przypadki "$(a \in A,\ b \in A)$ oraz $(a \in B,\ b \in B)$". Korzystając z tego, że $p_c > C^*/2$ oraz z oczywistego dolnego ograniczenia $C^* \geqslant \sum_{s=a}^{b} p_s + p_c$ otrzymujemy

$$p_c > \sum_{s=a}^{b} p_s. \tag{9.37}$$

Jeśli $a, b \in A$ to zgodnie z definicją A mamy $r_a \leqslant r_b$. Podobnie jeśli $a, b \in B$ to $q_b \leqslant q_a$. Stąd mamy

$$r_a + q_b \leqslant \max\{r_a + q_a, r_b + q_b\} \leqslant C^*. \tag{9.38}$$

Ostatecznie z (9.36), (9.38) oraz (9.37) otrzymujemy

$$C^{AB} < C^* + (1/2)\sum_{s=a}^{b} p_s + (1/2)\sum_{s=a}^{b} p_s$$

$$\leqslant C^* + (1/2)(\sum_{s=a}^{b} p_s + p_c) \leqslant (3/2)C^*. \tag{9.39}$$

Przypadki "$(a \in A,\ b = c)$, $(a \in A,\ b \in B)$ oraz $(a = c,\ b \in B)$". Rozpoczniemy od pokazania, że

$$C^* \geqslant \sum_{s=a}^{b} p_s + \min\{r_a, q_b\}. \tag{9.40}$$

Stosując (9.29) dla $I = \{a, \ldots, b\} \setminus \{c\}$ oraz $m = c$ otrzymujemy

$$C^* \geqslant \sum_{s=a}^{b} + \min\{r(I), q(I)\}. \tag{9.41}$$

Nierówność (9.41) implikuje (9.40). Rzeczywiście, jeśli $a \in A$ oraz $b \in B$ wtedy $r(I) \geqslant \min\{r_a, q_b\}$, $q(I) \geqslant \min\{r_a, q_b\}$ oraz $\min\{r(I), q(I)\} \geqslant$

j	1	2	3
r_j	ϵ	$1+\epsilon$	0
p_j	1	ϵ	1
q_j	1	1	0

Tabela 9.5: Przykład dla Algorytmu NS; ϵ jest pewną małą liczbą dodatnią

$\min\{r_a, q_b\}$. Dalej, jeśli $a \in A$ oraz $b = c$ to $\min\{r(I), q(I)\} = r(I) = r_a \geqslant \min\{r_a, q_b\}$. Podobnie, gdy $a = c$ i $b \in B$ mamy $\min\{r(I), q(I)\} = q(I) = q_b \geqslant \min\{r_a, q_b\}$. Dalej pokażemy, że zachodzi

$$C^* \geqslant p_c + \max\{r_a, q_b\}. \tag{9.42}$$

Nierówność (9.42) została otrzymana z (9.30) lub (9.31) kładąc $m = c$ oraz odpowiednie wartości i oraz j, a mianowicie jeśli $(a \in A,\ b = c)$ lub $(a = c,\ b \in B)$ przyjmujemy $i = a$ lub $i = b$ we wzorze (9.30) odpowiednio. Podobnie jeśli $(a \in A,\ b \in B)$ kładziemy $i = a,\ j = b$ we wzorze (9.31).

Ponieważ $p_c > C^*/2$, używając z (9.42) otrzymujemy $p_c > \max\{r_a, q_b\}$ oraz

$$C^* \geqslant 2\max\{r_a, q_b\}. \tag{9.43}$$

Ostatecznie, z (9.35), (9.40) oraz (9.43)

$$\begin{aligned} C^{AB} &= \min\{r_a, q_b\} + \sum_{s=a}^{b} p_s + \max\{r_a, q_b\} \\ &\leqslant C^* + (1/2)C^* \leqslant (3/2)C^* \end{aligned} \tag{9.44}$$

co kończy ten przypadek.

Oszacowanie dokładności najlepsze możliwe co pokazuje następujący przykład z danymi zamieszczonymi w Tablicy 9.7.4. Łatwo sprawdzić, że permutacja $\pi^* = (1, 2, 3)$ jest optymalna oraz $C^* = 2 + 2\epsilon$. Algorytm S generuje permutację $\pi^S = (3, 1, 2)$ oraz $C^S = 3 + \epsilon$. Ponieważ w π^S zadaniem interferencyjnym jest zadanie 3, zatem $c = 3$, $A = \{1\}$, $B = \{2\}$, $\pi^{AB} = (1, 3, 2)$ oraz $C^{AB} = 3 + 2\epsilon$. Stąd $C^{NS}/C^* = (3 + \epsilon)/(2 + 2\epsilon)$ oraz wartość tego wyrażenia dąży do $(3/2)$ jeśli $\epsilon \to 0$.

Zauważmy, że inne mutacje algorytmu posiadają taką samą ocenę. Przykładowo jeśli warunek "... jeśli c nie istnieje ..." występujący w Kroku 1

zastąpić warunkiem "... jeśli c nie istnieje lub $p_c < p(\mathcal{J})/2$ ...". Inną propozycją jest użycie do konstrukcji permutacji π^{AB} w miejsce c zadania $u \in \mathcal{J}$ takiego, że $p_u > p(\mathcal{J})/2$ (permutacja π^{AB} jest generowana tylko wtedy gdy takie u istnieje).

9.8 Schematy aproksymacyjne

Opisany schemat [159] definiuje rodzinę algorytmów, która dla każdego $\epsilon > 0$ wyznacza w czasie wielomianowym rozwiązanie przybliżone π^A takie, że $C^A/C^* \leqslant 1+\epsilon$, gdzie $C^A = C_{\max}(\pi^A)$. Do konstrukcji schematu wykorzystuje się rozwiązanie przybliżone problemu pomocniczego $1|r_j \in Z, q_j|C_{\max}$, w którym zadania posiadają skończoną liczbę $k \stackrel{\text{def}}{=} |Z| < n$ różnych terminów gotowości. Rozwiązanie tego ostatniego problemu można otrzymać poprzez pomocniczy schemat oparty na kodowaniu jedynkowym czasów zadań lub schemat operujący na szkicach uszeregowań. Zauważmy, że jeśli dysponujemy żądanym pomocniczym schematem aproksymacyjnym dla problemu $1|r_j \in Z, q_j|C_{\max}$, to możemy skonstruować w czasie wielomianowym schemat aproksymacyjny dla problemu $1|r_j, q_j|C_{\max}$ według następującej zasady. Niech $\Delta = \frac{\epsilon}{2} \max_{1 \leqslant j \leqslant n} r_j$. Zaokrąglamy wartości terminów gotowości zadań do najbliższej całkowitej wielokrotności Δ poprzez podstawienie $r_j := \lfloor \frac{r_j}{\Delta} \rfloor \Delta$. Zaokrąglenie powoduje, że zadania otrzymują $2/\epsilon + 1$ różnych terminów gotowości. Niech $S = (S_1, \ldots, S_n)$ będzie wektorem terminów rozpoczęcia wykonywania zadań otrzymanych w wyniku rozwiązania $1|r_j \in Z, q_j|C_{\max}$. Wektor $(S_1 + \Delta, \ldots, S_n + \Delta)$ jest dopuszczalny dla problemu $1|r_j, q_j|C_{\max}$ przy czym długość uszeregowania jest nie większa niż $(1 + \frac{\epsilon}{2})C^* + \Delta$. Ponieważ $r_j \leqslant C^*$, zatem $\Delta \leqslant \frac{\epsilon}{2} C^*$ oraz otrzymane rozwiązanie ma długość nie większą niż $(1 + \epsilon)C^*$.

Kluczowym elementem opisywanego schematu jest problem pomocniczy $1|r_j \in Z, q_j|C_{\max}$. Do jego rozwiązania proponuje się zastosowanie jednego z dwóch opisanych poniżej schematów aproksymacyjnych. Oznaczmy przez k liczbę różnych terminów gotowości zaś przez $Z = \{z_1, \ldots, z_k\}$ uporządkowany zbiór ich wartości; dla wygody zapisu przyjmijmy także $z_{k+1} = \infty$.

Schemat pomocniczy oparty na kodowaniu jedynkowym

Bazą schematu jest algorytm PD (opisany dalej), który dla problemu $1|r_j \in Z, q_j|C_{\max}$ wyznacza rozwiązanie optymalne w czasie $O(nkP^{2k})$.

Pokażemy wpierw, że algorytm ten prowadzi do poszukiwanego wielomianowego schematu aproksymacyjny. Problem $1|r_j \in Z, q_j|C_{\max}$ z danymi

r_j, p_j, q_j aproksymujemy problemem takiego samego typu lecz z danymi "przeskalowanymi" $\tilde{r}_j = r_j/\delta$, $\tilde{p}_j = \lfloor p_j/\delta \rfloor$, $\tilde{q}_j = q_j/\delta$, gdzie $\delta = \epsilon P/n$, $P = p(\mathcal{J})$. Ponieważ zachodzi

$$\sum_{j=1}^{n} \tilde{p}_j \leqslant \sum_{j=1}^{n} p_j/\delta = P/\delta = n/\epsilon, \tag{9.45}$$

to wspomniany algorytm PD dla problemu $1|r_j \in Z, q_j|C_{\max}$ z danymi $\tilde{r}_j$, $\tilde{p}_j$, $\tilde{q}_j$ wyznaczy rozwiązanie optymalne w czasie $O(nk(n/\epsilon)^{2k})$, czyli w czasie wielomianowo zależnym od n oraz $1/\epsilon$. Otrzymane rozwiązanie optymalne $S^* = (S_1^*, \ldots, S_n^*)$ generuje następnie rozwiązanie optymalne $\delta S^* = (\delta S_1^*, \ldots, \delta S_n^*)$ dla problemu z danymi "przeskalowanymi w górę", to jest δr_j, $\delta\tilde{p}_j$, δq_j. Aproksymując p_j przez $\delta\tilde{p}_j$ dokonano zaokrąglenia, zatem otrzymana długość uszeregowania dla problemu $1|r_j \in Z, q_j|C_{\max}$ z danymi δr_j, $\delta\tilde{p}_j$, δq_j jest nie większa niż optymalna wartość $C_{\max}(\pi^*)$ dla problemu wyjściowego $1|r_j \in Z, q_j|C_{\max}$ z danymi r_j, p_j, q_j. "Rozszerzając" otrzymane rozwiązanie δS^* do rozwiązania dopuszczalnego problemu wyjściowego wystarczy powiększyć czasy wykonywania wszystkich zadań o wartość $\delta = \epsilon P/n$, co odpowiada terminom rozpoczęcia $(\delta S_1^*, \delta(S_2^*+1), \ldots, \delta(S_n^*+n-1))$. Otrzymujemy wtedy uszeregowanie o długości nie większej niż $C_{\max}(\pi^*) + \epsilon P \leqslant (1+\epsilon)C_{\max}(\pi^*)$.

Podamy teraz wspomniany na wstępie algorytm PD wyznaczający rozwiązanie optymalne problemu $1|r_j \in Z, q_j|C_{\max}$ w czasie $O(nkP^{2k})$. Ponieważ Z jest zbiorem uporządkowanym, to wszystkie uszeregowania $S = (S_1, \ldots, S_n)$ spełniają warunek $S_j \in [z_1, z_k + P]$, $j = 1, \ldots, n$. Przedział $[z_1, z_k + P]$ dekomponujemy na k rozłącznych podprzedziałów postaci $[z_i + w_i, z_{i+1} + w_{i+1})$, gdzie $z_i + w_i < z_{i+1} + w_{i+1}$ oraz $0 \leqslant w_i \leqslant P$, $i = 1, \ldots, k$ są pewnymi liczbami. Każdy wektor $w = (w_1, \ldots, w_k)$ nazywamy *rozkładem kanonicznym* uszeregowań. Dla każdego S istnieje w takie, że $S_j \in [z_i + w_i, z_{i+1} + w_{i+1})$, dla pewnego i oraz zadanie j jest w całości wykonywane w tym przedziale. Wszystkich rozkładów w jest nie więcej niż P^k. Rozkład kanoniczny pozwala nam zastąpić wybór wartości n zmiennych S_j, $j = 1, \ldots, n$ wyborem wartości k zmiennych w_i, $i = 1, \ldots, k$, co jest korzystne gdy $k < n$. Nie wszystkie wektory w generują uszeregowanie dopuszczalne, podobnie zresztą jak nie wszystkie S. W każdym przedziale $[z_i + w_i, z_{i+1} + w_{i+1})$ musi być wykonany w całości pewien podzbiór zadań, które zgodnie z definicją w_{i+1} są dostępne już w z_i. Zatem optymalną kolejność zadań w tym podzbiorze można określić zgodnie z odwrotną regułą Jacksona.

Algorytm PD wywołuje Algorytm PD-C dla każdego przydziału kano-

nicznego w i wymaga by zadania były indeksowane zgodnie z malejacymi wartościami końcówek, $q_1 \geqslant \ldots \geqslant q_n$. Dla każdego ustalonego w Algorytm PD-C poszukuje najlepszego przydziału zadań do przedziałów czasowych $[z_i + w_i, z_{i+1} + w_{z+1})$, $i = 1, \ldots, k-1$. Niech t_i oznacza sumą czasów wykonywania zadań już przydzielonych do przedziału $[z_i + w_i, z_{i+1} + w_{z+1})$ zaś j – kolejno przydzielane zadanie. Oznaczmy przez $F_w(t; j) = F_w(t_1 \ldots, t_n; j)$ minimalną długość uszeregowania dla zadań ze zbioru $\{1, \ldots, j\}$ zajmujących czas t_i w każdym przedziale. Ze względu na dopuszczalność musi zachodzić

$$\sum_{i=1}^{k} t_i = \sum_{s=1}^{j} p_s \tag{9.46}$$

oraz

$$z_i + w_i + t_i \leqslant z_{i+1} + w_{i+1},\ i = 1, \ldots, k-1. \tag{9.47}$$

Niech T^j określa zbiór wszystkich wartości t spełniających warunki (9.46)–(9.47). Podstawowa zależność rekurencyjna dla PD-C dana jest wzorem

$$F_w(t; j) = \min_{1 \leqslant i \leqslant k;\ r_j \leqslant z_i} \max\{F_w(t_1, \ldots, t_{i-1}, t_i - p_j, t_{i+1}, \ldots, t_n; j-1),$$

$$z_i + w_i + t_i + q_j\},\ t \in T^j,\ j = 1, \ldots, n, \tag{9.48}$$

gdzie $F_w(t; 0) = \infty$ dla każdego t z wyjątkiem $F_w(0; 0) = 0$. Jeśli zadanie j zostało, w wyniku zastosowania wzoru (9.48), wprowadzone do przedziału i-tego, to jest umieszczane jako ostatnie za wszystkimi zadaniami już uszeregowanymi w tym przedziale. Celem algorytmu PD-C jest wyznaczenie wartości $\min_{t \in T^n} F(t; n)$ oraz odpowiadającego mu uszeregowania.

Złożoność obliczeniowa schematu dla problemu $1|r_j, q_j|C_{\max}$, z wykorzystaniem pomocniczego algorytmu opartego na kodowaniu jedynkowym dla problemu $1|r_j \in Z, q_j|C_{\max}$, jest rzędu $O(16^{1/\epsilon}(n/\epsilon)^{3+4/\epsilon})$. Dla $\epsilon = 1/2$ złożoność ta jest rzędu $O(2^{19}n^{11})$ (porównaj z $O(n \log n)$ dla algorytmu NS), zaś dla $\epsilon = 1/3$ odpowiednio $O(9 \cdot 6^{12}n^{15})$ (porównaj z $O(n^2 \log n)$ dla HS).

Schemat pomocniczy oparty na szkicach

Schemat ten wyznacza rozwiązanie przybliżone problemu $1|r_j \in Z, q_j|C_{\max}$ korzystając z pojęcia *szkicu*, który określa dokładne rozmieszczenie zadań *dużych* oraz przybliżone rozmieszczenie zadań *małych* w uszeregowaniu. Żądane rozwiązanie jest poszukiwane metodą przeglądu szkiców.

Wielkość zadania jest oceniana przy pomocy wartości progowej

$$t = \frac{\epsilon P}{2k}. \tag{9.49}$$

Stąd mamy zbiór zadań dużych i małych, odpowiednio

$$A = \{j : \ p_j \geqslant t\}, \ \ B = \{j : \ p_j < t\}. \tag{9.50}$$

Zadań dużych jest nie więcej niż $P/t = 2k/\epsilon$.

Rozważmy pewne uszeregowanie $S = (S_1, \ldots, S_n)$, dla kórego zdefiniujmy zbiory

$$I_i = \{j \in A : \ S_j \in [z_i, z_{i+1})\}, \ \ K_i = \{j \in B : \ S_j \in [z_i, z_{i+1})\}. \tag{9.51}$$

dla $i = 1, \ldots, k$. W sposób oczywisty zadania ze zbioru $I_i \cup K_i$ są wykonywane przed zadaniami ze zbioru $I_{i+1} \cup K_{i+1}$ dla $i = 1, \ldots, k-1$. *Szkicem uszeregowania* S nazywamy zestaw $\{(I_i, \nu_i)\}$, $i = 1, \ldots, k$, gdzie

$$\nu_i = \lceil p(K_i)/t \rceil. \tag{9.52}$$

Liczba możliwych szkiców jest iloczynem liczby przypisań efektywnych terminów gotowości do zadań dużych oraz liczby możliwych wyborów wielkości $\nu_1, \ldots, \nu_k$ dla zadań małych, czyli $O(k^{|A|}) = O(k^{4k/\epsilon})$ oraz $O(k^{P/t}) = O(k^{4k/\epsilon})$, odpowiednio.

Dla danego szkicu rozwiązanie dopuszczalne S wyznaczamy posługując się pewnym chciwym algorytmem przybliżonym. Algorytm ten wpierw konstruuje zbiory K_i, $i = 1, \ldots, k$ w oparciu o dany wektor $\nu = (\nu_1, \ldots, \nu_k)$. Następnie każdy zbiór $I_i \cup K_i$ jest porządkowany zgodnie z odwrotną regułą Jacksona według nierosnących wartości q_j i określany są terminy rozpoczęcia zadań. Uszeregowanie jest tworzone poprzez konkatenację uporządkowanych zbiorów w kolejności od $I_1 \cup K_1$ do $I_k \cup K_k$.

Niech $N_i = \nu_i t$. Algorytm chciwy rozpoczyna pracę od podstawienia $K_i = \emptyset$, $i = 1, \ldots, k$. Następnie, dla $i = 1, \ldots, k$ kolejno wyznaczamy zbiór

$$W_i = \{j \in B : \ (j \notin K_s, \ s = 1, \ldots, k), \ (r_j \leqslant z_i)\}, \tag{9.53}$$

który po uporządkowaniu według nierosnących wartości q_j dostarcza listy dla centralnej części algorytmu chciwego. Kolejne zadania z listy są wybierane i dodawane do K_i zgodnie z zasadą chciwości, tak by łączna suma czasów zadań wybranych nie przekraczała lub przekraczała w stopniu nieznacznym wartość N_i.

Złożoność obliczeniowa schematu dla problemu $1|r_j, q_j|C_{\max}$, z wykorzystaniem pomocniczego algorytmu opartego na schematach dla problemu $1|r_j \in Z, q_j|C_{\max}$, jest rzędu $O(n \log n + n(4/\epsilon)^{8/\epsilon^2+8/\epsilon+2})$. Dla $\epsilon = 1/2$ złożoność ta jest rzędu $O(n \log n + 2^{150} n)$, zaś dla $\epsilon = 1/3$ odpowiednio $O(n \log n + 6^{196} n)$.

9.9 Algorytmy przeglądu

Metody przeglądu dla problemu $1|r_j, q_j|C_{\max}$ były analizowane między innymi w pracach [25,51,129,238]. Jak dotychczas najbardziej elegancka wersja pochodzi z pracy [51] i jest porównywalna, w sensie efektywności, z nieco bardziej skomplikowanym algorytmem blokowym [129].

9.9.1 Algorytm Carliera

Opisywany algorytm jest schematem B&B, który generuje dla każdego węzła w drzewie rozwiązań kompletne rozwiązanie używając do tego celu algorytmu S. Na bazie tego rozwiązanie określana jest zasada podziału, dolne i górne ograniczenia oraz reguły eliminacji, korzystając z pewnych własności π^S opisanych poniżej.

Podstawowe własności

Niech π^S będzie permutacją wygenerowaną Algorytmem S z drogą krytyczną (a, b) odpowiednio do definicji podanej w Rozdz. 9.4. Zachodzą dwa wyłączające się wzajemnie przypadki: (1) jeśli π^S nie jest optymalna to istnieje zadanie $c \in \mathcal{J}$ oraz zbiór $K \subseteq \mathcal{J}$ takie, że $h(K) > C_{\max}(\pi^S) - p_c$ oraz w rozwiązaniu optymalnym c jest wykonywane przed lub za wszystkimi zadaniami z K; (2) jeśli π^S jest optymalna to istnieje $I \subseteq \mathcal{J}$ taki, że $C_{\max}(\pi^S) = h(I)$.

Istotnie, w celu dowodu własności, bez straty ogólności możemy założyć, że $\pi^S = \pi$ gdzie $\pi(i) = i$, $i = 1, \ldots, n$. Zatem zgodnie ze wzorem mamy $C\max(\pi) = r_a + p(I) + q_b$, gdzie $I = \{a, a+1, \ldots, b\}$.

Wpierw pokażemy, że żadne zadanie nie jest wykonywane w przedziale $(r_a - 1, r_a)$. Przypuśćmy, że istnieje i rozpoczynające się w $S_i < r_a$ oraz $S_i + p_i > r_a$. Stąd $S_i + p_i + p(I) + q_b > C_{\max}(\pi)$ co przeczy temu, że $C_{\max}(\pi)$ jest maksymalne. Dalej pokażemy, że $r_a = r(I)$. Istotnie, ponieważ maszyna jest wolna w przedziale czasu $(r_a - 1, r_a)$. Stąd w algorytmie gdy szeregujemy a w chwili $t = r_a$ mamy $t = r_a = \min_{j \in \mathcal{J} \setminus U} r_j$ oraz $r_a = \min_{j \in I} r_j = r(I)$ ponieważ $I \subseteq \mathcal{J} \setminus U$. Wreszcie, jeśli $q_b = q(I)$ to długość ścieżki krytycznej jest równa $C_{\max}(\pi) = r_a + p(I) + q_b = r(I) + p(I) + q(I) = h(I)$ zatem rozwiązanie jest optymalne. Występuje to zawsze gdy spełniona jest zależność "jeśli $r_i \leqslant r_j$ to $q_i \geqslant q_j$". Przykładowo, jeśli wszystkie r_j sa równe lub wszystkie q_j są równe. W przypadku przeciwnym do poprzedniego, to znaczy gdy $q_b > q(I)$, istnieje zadanie i takie, że $i < b$ oraz $q_i < q_b$; oznaczmy przez c największy indeks takiego zadania. Zdefiniujmy zbiór $K =$

$\{c+1,\ldots,b\}$. Zgodnie z definicją mamy $q_c < q_j$, $j \in K$. Łatwo zauważyć, że zachodzi $S_c < r_j$, $j \in K$. Istotnie, gdyby zachodziło $S_c \geqslant r_i$ dla pewnego $i \in K$, to zadanie i byłoby gotowe do wykonywania w chwili czasowej $t = S_c$, w której podejmowano decyzje o uszeregowaniu zadania c, i zadanie to zostałoby uszeregowane w chwili t zamiast c ze względu na warunek $q_j > q_c$. Zatem otrzymujemy nierówność prawdziwą dla każdego $j \in K$

$$r_j > S_c = r_a + \sum_{t=a}^{c-1} p_t \tag{9.54}$$

implikującą oczywistą nierówność

$$r(K) = \min_{j\in K} r_j > S_c = r_a + \sum_{t=a}^{c-1} p_t. \tag{9.55}$$

Zgodnie z definicją zbioru K mamy $q_b = \min_{j\in K} q_j = q(K)$. Zatem ostatecznie

$$h(K) = r(K) + p(K) + q(K) > r_a + p(I) + q_b - p_c = C_{\max}(\pi) - p_c \tag{9.56}$$

co kończy dowód własności.

Odpowiednio, zadanie c oraz zbiór K zdefiniowany w Rozdz. 9.4 spełniają warunki podanej własności. W przykładzie opisanym w Tablicy 9.7.1 mamy $c = 1$, $K = \{2, 3, 4\}$, $h(K) = 49$, patrz także Rys. 9.3, 9.5.

Zasada podziału

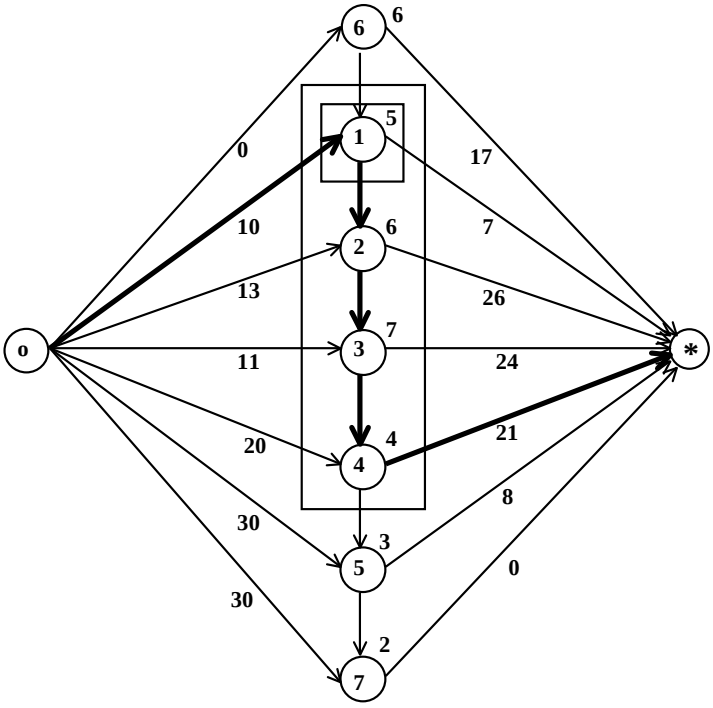
Z każdym węzłem drzewa rozwiązań kojarzymy:

1. problem $1|r_j, q_j|C_{\max}$ z danymi $(r_j, p_j, q_j, j = 1,\ldots,n)$; wartości r_j, q_j będą ulegały modyfikacji w czasie pracy algorytmu,

2. permutację π^S wygenerowaną Algorytmem S wraz z odpowiadającą mu wartością funkcji celu $C_{\max}(\pi^S)$, drogą krytyczną (a, b), zadaniem interferencyjnym c oraz zbiorem krytycznym K, zgodnie z definicjami podanymi w Rozdz. 9.4

3. dolne ograniczenie wartości celu LB dla wszystkich rozwiązań problemu z danymi charakterystycznymi dla tego węzła,

4. górne ograniczenie UB wartości funkcji celu dla wszystkich rozwiązań sprawdzonych przed analizą rozpatrywanego węzła.



Rysunek 9.5: Interpretacja zadania c oraz zbioru K na grafie.

Przyjmujemy początkowo $UB = \infty$. Wyznaczamy π^S poprzez zastosowanie Algorytmu S, a w dalszej kolejności odpowiadającą mu wartość funkcji celu $C_{\max}(\pi^S)$, drogę krytyczną (a, b), zadanie interferencyjne c oraz zbiór krytyczny K, zgodnie z definicjami podanymi w Rozdz. 9.4. Jeśli c nie istnieje to otrzymane rozwiązanie jest optymalne dla rozważanego problemu. W przeciwnym przypadku kreowane są 2+1 problemy potomne (następniki w drzewie rozwiązań), z których 2 podlegają sprawdzeniu zaś jeden jest zawsze eliminowany.

Dwa podstawowe problemy potomne są określone następująco:

(A) "zadanie c jest wykonywane przed wszystkimi zadaniami ze zbioru K". Warunek ten jest uwzględniany poprzez modyfikację danych problemu według zasady

$$q_c := \max\{q_c, p(K) + q_b\}. \tag{9.57}$$

(B) "zadanie c jest wykonywane za wszystkimi zadaniami ze zbioru K". Warunek ten jest uwzględniany poprzez modyfikację danych problemu według zasady

$$r_c := \max\{r_c, r(K) + p(K)\}. \tag{9.58}$$

Zauważmy, że $p(K) = C_{\pi^S(b)} - C_c$ czyli może być wyznaczony w czasie $O(1)$.

Górne ograniczenie

Jako górne ograniczenie przyjmuje się $UB := \min\{UB, C_{\max}(\pi^S)\}$ każdorazowo po wygenerowaniu π^S dla wszystkich analizowanych problemów potomnych.

Eliminacja

Dla rozpatrywanej permutacji π^S tworzony jest zbiór

$$L = \{i \in \mathcal{J} \setminus K : \ p_i > UB - h(K)\}. \tag{9.59}$$

Jeśli $i \in L$ to i musi być wykonywane przed lub za K. Odpowiednio budowane są dwa dodatkowe testy eliminacyjne

1. Jeśli $i \in L$ oraz
$$r_i + p_i + p(K) + q_b \geqslant UB \tag{9.60}$$
to i musi być wykonywane za wszystkimi zadaniami z K co pozwala na wykonanie modyfikacji
$$r_i := \max\{r_i, r(K) + p(K)\} \tag{9.61}$$

2. Jeśli $i \in L$ oraz
$$r(K) + p_i + p(K) + q_i \geqslant UB \tag{9.62}$$
to i musi być wykonywane przed wszystkimi zadaniami z K co pozwala na wykonanie modyfikacji
$$q_i := \max\{q_i, q(K) + p(K)\} \tag{9.63}$$

Dolne ograniczenie

Proponuje się zastosować
$$LB = \max\{C^*_{\max,pmtn}, h(K), h(K \cup \{c\})\} \tag{9.64}$$
gdzie $C^*_{\max,pmtn}$ jest optymalną wartością funkcji celu problemu postaci $1|r_j, q_j, pmtn|C_{\max}$ dla danych r_j, q_j, $j = 1, \ldots, n$ skojarzonych z analizowanym węzłem.

Strategia przeglądania

Realizowana jest strategia "w głąb". Z bieżącego węzła drzewa rozwiązań generowane są co najwyżej dwa węzły potomne w kolejności wynikającej z pomocniczego dolnego ograniczenia. Wybrany węzeł staje się bieżącym, zaś ten drugi staje się chwilowo odłożonym. Po zamknięciu (sprawdzeniu lub wyeliminowaniu) bieżącego węzła kontynuowany jest proces analizy węzłów poczynając od ostatnio odłożonego węzła. Poszukiwania kończą się gdy wszystkie odłożone węzły zostaną przeanalizowane.

9.9.2 Algorytm blokowy

Bezpośrednie wykorzystanie grafowego modelu problemu oraz wzoru (9.12) pozwala na otrzymanie pewnych teoretycznych własności użytecznych, między innymi, przy konstrukcji algorytmów dokładnych opartych na schemacie B&B. Własności te stanowią podstawę *podejścia blokowego*, oryginalnego kierunku badawczego opracowanego pierwotnie w ICT PWr, a następnie stosowanego z powodzeniem w wielu innych ośrodkach zagranicznych. Uniwersalność proponowanego podejścia czyni je przydatnym zarówno w algorytmach dokładnych jak i metodach przybliżonych.

Podstawowe własności

Niech π będzie pewną permutacją z drogą krytyczną (a, b), patrz Rozdz. 9.4. Ciąg zadań $B = (\pi(a), \ldots, \pi(b))$ będziemy nazywać *blokiem* zadań w π.

Zadanie $\pi(a)$ jest *początkowym* zadaniem tego bloku, zaś $\pi(b)$ odpowiednio zadaniem *końcowym*. Dla uproszczenia zapisu zbiór zadań należących do bloku B będziemy również oznaczać przez B. Następujące twierdzenie, przeformułowanie Twierdzenia 2 z pracy Grabowskiego stanowi podstawę własności blokowych: jeśli $\beta \in \Pi$ jest dowolną permutacją taką, że $C_{\max}(\beta) < C_{\max}(\pi)$, to istnieje zadanie $\pi(k) \in B \setminus \{\pi(a)\}$ (lub zadanie $\pi(k) \in B \setminus \{\pi(b)\}$) takie, że w permutacji β zadanie $\pi(k)$ poprzedza wszystkie pozostałe zadania bloku (występuje za wszystkimi pozostałymi zadaniami bloku). Jedno z najwcześniejszych zastosowań twierdzenia blokowego prowadzi do algorytmu opartego na schemacie podziału i ograniczeń (B&B) opisanego poniżej.

Schemat podziału

Z każdym węzłem drzewa rozwiązań kojarzymy:

1. relację Y częściowego porządku w zbiorze zadań implikującą ograniczenie *"jeśli $(i,j) \in Y$ to zadanie i musi być zakończone przed rozpoczęciem zadania j"*,

2. zbiór permutacji $\Pi(Y)$ spełniających wymagania relacji częściowego porządku Y, tzn. $\Pi(Y) = \{\sigma \in \Pi : \ ((\sigma(i), \sigma(j)) \in Y) \rightarrow (i < j)\}$,

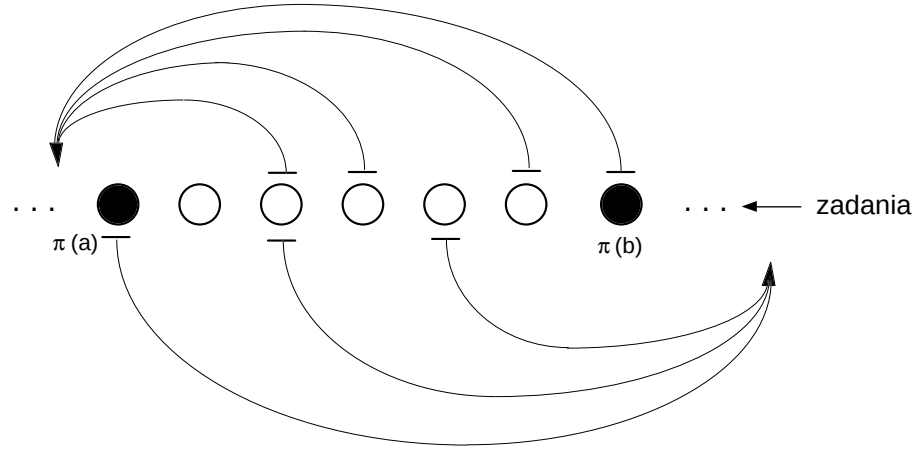
3. kompletne rozwiązanie $\pi \in \Pi(Y)$, będące reprezentantem zbioru $\Pi(Y)$, wraz z odpowiadającą mu wartością funkcji celu $C_{\max}(\pi)$ i drogą krytyczną (a, b),

4. dolne ograniczenie wartości celu $LB(Y)$ dla wszystkich rozwiązań z $\Pi(Y)$,

5. górne ograniczenie $UB(Y)$ wartości funkcji celu dla wszystkich rozwiązań sprawdzonych przed rozważeniem zbioru $\Pi(Y)$.

Przyjmujemy początkowo $Y = \emptyset$ oraz $UB(Y) = \infty$. Stąd $\Pi(Y) = \Pi$. Zakładamy, że π jest dowolną permutacją startową. Podział zbioru $\Pi(Y)$ przeprowadzamy w oparciu o permutację π, konstruując równocześnie potomne zbiory permutacji $\Pi(Y^k)$ oraz odpowiednich reprezentantów $\beta_k \in \Pi(Y^k)$, $k = 1, \ldots, s$, gdzie s jest pewną liczbą zależną od π i wyznaczoną w sposób opisany w dalszym ciągu. Relacje częściowego porządku Y^k są określone przez złożenie pewnych zdań logicznych oraz ich negacji.

Każdą nową permutację β, potomek π, generujemy przesuwając jedno zadanie w π przed (lub za) pierwsze (lub ostatnie) zadanie bloku, tj. na pozycję a (lub b, odpowiednio), Rys. 9.6. Zadanie, które będą przesuwane



Rysunek 9.6: Schemat przesuwania zadań.

na pozycję a są wybierane ze zbioru kandydatów (patrz rys. 9.7)

$$E_a = \{j \in \mathcal{J} : \ j \in B, \ \Delta_a(j) < 0\} \tag{9.65}$$

gdzie $\Delta_a(j) = r_j - r_{\pi(a)}$, zaś te które mogą być przesuwane na pozycję b są wybierane ze zbioru

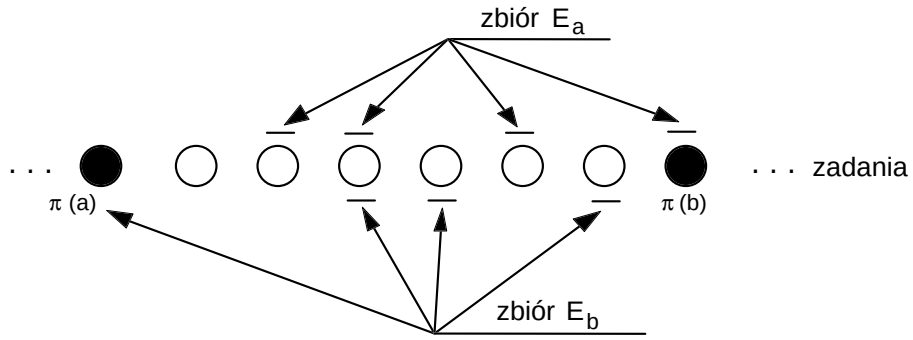
$$E_b = \{j \in \mathcal{J} : \ j \in B, \ \Delta_b(j) < 0\} \tag{9.66}$$

gdzie $\Delta_b(j) = q_j - q_{\pi(b)}$. Stąd łączna liczba następników dla π jest równa $s = e_a + e_b = |E_a| + |E_b|$.

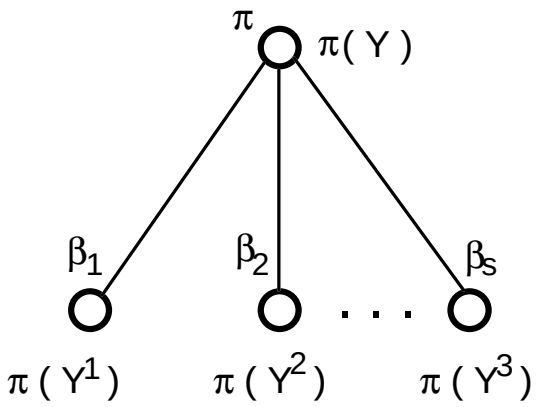
W celu określenia odpowiednich zbiorów $\Pi(Y^k)$ załóżmy, że permutacje są generowane w pewien uporządkowany sposób $\beta_1, \ldots, \beta_{e_a}, \beta_{e_a}, \beta_{e_a+1}, \ldots, \beta_s$, gdzie $\beta_1, \ldots, \beta_{e_a}$ są permutacjami generowanymi przez przesuwanie zadań ze zbioru kandydatów E_a, zaś $\beta_{e_a+1}, \ldots, \beta_s$ – poprzez przesuwanie zadań ze zbioru E_b. Odpowiednio do generowanych następników β_r, $r = 1, \ldots, s$ zbiór $\Pi(Y)$ jest dzielony na $s+1$ podzbiorów mianowicie $\Pi(Y^r)$, $r = 1, \ldots, s$ oraz $\Pi(Y) \setminus \bigcup_{r=1}^{s} \Pi(Y^r)$, patrz Rys. 9.8.

Jeśli permutacja β_r, dla pewnego r, $1 \leqslant r \leqslant e_a$, została wygenerowana z π poprzez przesunięcie zadania $j \in E_a$, wtedy $\Pi(Y^r)$ jest zbiorem tych permutacji z $\Pi(Y)$, które spełniają własność *"zadanie j musi być wykonywane jako pierwsze zadanie pośród zadań $E'_a = E_a \cup \{\pi(a)\}$"*. Własność ta może być wyrażona relacją poprzedzania postaci, rys. 9.9 (a),

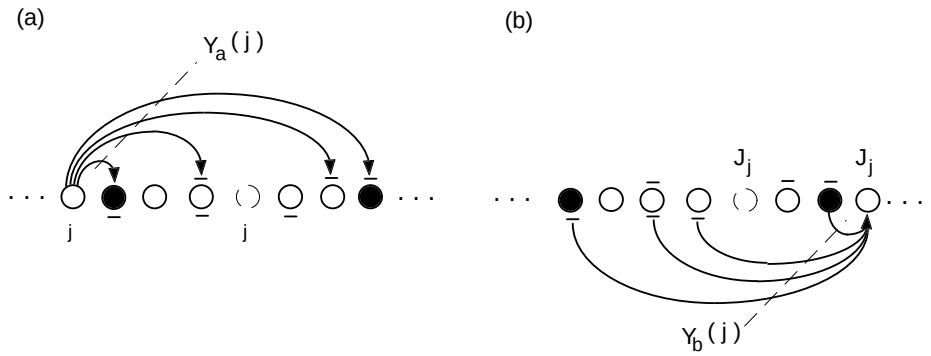
$$Y_a(j) = \{(j,k); \ k \in E'_a \setminus \{j\}\}. \tag{9.67}$$



Rysunek 9.7: Zbiory E_a i E_b.



Rysunek 9.8: Drzewo rozwiązań.



Rysunek 9.9: Relacja poprzedzania $Y_a(j)$.

Zatem mamy $Y^r = Y \cup Y_a(j)$ oraz Y^r musi być przestrzegana w każdym następniku względem β_r w drzewie rozwiązań, tj. dla $\beta \in \Pi(X)$, $Y^r \subseteq X$. Zauważmy, że zbiory $\Pi(Y^t)$, $t > r$, powinny uwzględniać negację powyższego warunku logicznego wyrażoną zdaniem *"zadanie j nie może być wykonywane jako pierwsze zadanie spośród zadań E'_a"*. Niestety, podany warunek nie może być przedstawiony w postaci relacyjnej. Dopiero łączne dopełnienie warunków logicznych dla wszystkich $j \in E_a$, wyrażone zdaniem logicznym *"zadanie $\pi(a)$ musi być wykonywane jako pierwsze spośród zadań E'_a"* może być opisane relacją poprzedzania postaci, rys. 9.10,

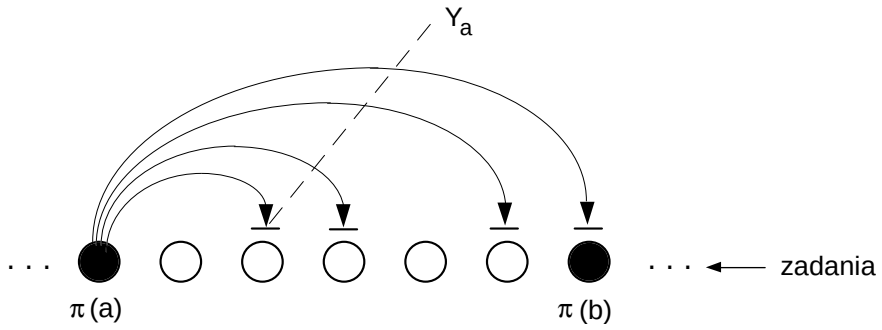
$$Y_a = \{(\pi(a), k);\ k \in (E_a\}. \tag{9.68}$$

Stąd zbiory $\Pi(Y^t)$, $t > e_a$ muszą spełniać (9.68) czyli $Y_a \subset Y^t$ dla $t > e_a$.

Z kolei, jeśli permutacja β_r, $e_a < r \leqslant s$ została wygenerowana z π poprzez przesunięcie zadania $j \in E_b$ wtedy $\Pi(Y^r)$ jest zbiorem permutacji z $\Pi(Y)$, które spełniają (9.67) oraz własność *"zadanie j musi być wykonywane jako ostatnie zadanie spośród zadań $E'_b = E_b \cup \{\pi(b)\}$"*. Ta ostatnia własność jest wyrażona poprzez relację poprzedzania, rys. 9.9(b),

$$Y_b(j) = \{(k, j);\ k \in E'_b \setminus \{j\}\}. \tag{9.69}$$

Zatem mamy $Y^r = Y \cup Y_a \cup Y_b(j)$. Podobnie jak poprzednio, negacja warunku logicznego związana z przenoszenie zadań z E_b nie może być przedstawiona w postaci relacji poprzedzania. Dopiero łączne dopełnienie warunków logicznych dla zadań $j \in E_b$, może być wyrażone zdaniem *"zadanie $\pi(b)$*



Rysunek 9.10: Relacja poprzedzania Y_a .

musi być wykonywane jako ostatnie spośród zadań E'_b*"*, oraz konsekwentnie odpowiednią relacją poprzedzania postaci, patrz złożenie rys. 9.11,

$$Y_b = \{(k, \pi(b));\ k \in E_b\}. \tag{9.70}$$

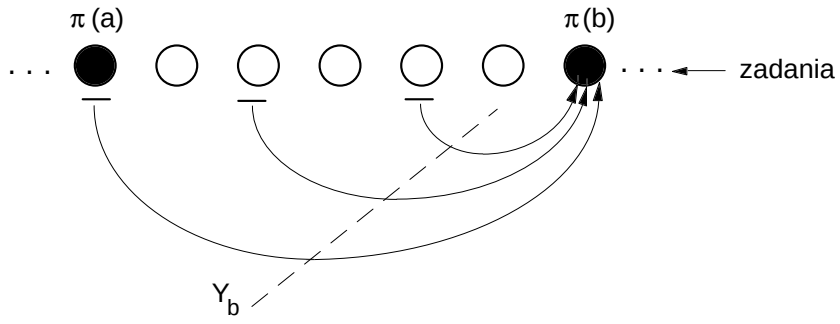
Zatem zbiór $\Pi(Y) \backslash \bigcup_{r=1}^{s} \Pi(Y^r)$ zawiera permutacje spełniające ograniczenia relacji Y rozszerzonej o $Y_a \cup Y_b$, rys. 9.12.

Problemy skojarzone ze zbiorami $\Pi(Y^r)$ są analizowane w kolejności wynikającej z przyjętej strategii rozwiązywania. Problem skojarzony ze zbiorem $\Pi(Y) \setminus \bigcup_{r=1}^{s} \Pi(Y^r)$ jest zawsze eliminowany na mocy rozszerzonej własności blokowej podanej poniżej.

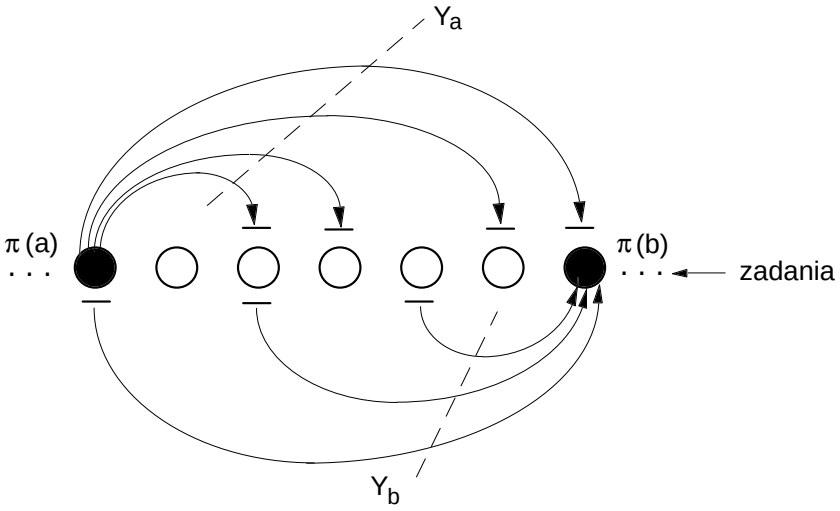
Eliminacja

Dalej wykorzystamy pewne uogólnienie bazowej własności blokowej: niech $\pi \in \Pi(Y)$, dla pewnego Y, zaś Y_a, Y_b niech będą zbiorami określonymi przez (9.68) i (9.70) dla tej π. Dla każdej permutacji $\gamma \in \Pi(Y \cup Y_a \cup Y(b$ zachodzi $C_{\max}(\gamma) \geqslant C_{\max}(\pi)$. Powyższa własność pozwala zawsze eliminować $s+1$-szy problem potomny dla każdego π.

Eliminacja problemu częściowego zachodzi także w dowolnym z trzech przypadków: (1) jeśli $E_a = \emptyset = E_b$, (2) jeśli $LB(\pi) \geqslant UB$, oraz (3) jeśli $\Pi(Y^r) = \emptyset$. Odpowiednio do (1) i (2) wszystkie problemy częściowe potomne w stosunku do π są eliminowane, zaś w przypadku (3) eliminacji podlega tylko problem skojarzony ze stosownym $\Pi(Y^r)$. Przyjmujemy, że $\Pi(Y^r) = \emptyset$



Rysunek 9.11: Relacja poprzedzania Y_b .



Rysunek 9.12: Relacja poprzedzania $Y_a \cup Y_b$.

jeśli Y^r jest relacją zawierającą cykl. Fakt ten może być wykryty *przed* wykonaniem przesunięcia zadania poprzez badanie struktury relacji Y. W praktyce, "automatyczne" pomijanie problemów częściowych typu (3) jest realizowane poprzez prze-definiowanie zbiorów E_a oraz E_b następująco

$$E_a = \{\pi(i) \in \mathcal{J} : \ \pi(i) \in B, \ \Delta_a(\pi(i)) < 0, \ ((\pi(k), \pi(i)) \notin Y, \ k = a, \ldots, i-1)\} \tag{9.71}$$

$$E_b = \{\pi(i) \in \mathcal{J} : \ \pi(i) \in B, \ \Delta_b(\pi(i)) < 0, \ ((\pi(i), \pi(k)) \notin Y, \ k = i+1, \ldots, b)\}. \tag{9.72}$$

Można sprawdzić, że własność pozostaje w mocy dla zbiorów kandydatów zdefiniowanych przez (9.71) – (9.72).

Przypadki szczególne

Poniżej przedstawiono listę przypadków szczególnych, które wykrycie i uwzględnienie może znacząco przyśpieszyć pracę algorytmu.

(A) "$E_b \setminus \{\pi(b)\} \neq \emptyset$ oraz $E_b \setminus \{\pi(b)\} \neq \emptyset$"

W tym przypadku usuwamy zadania $\pi(b$ oraz $\pi(a)$ ze zbiorów E_a oraz E_b zakładając, że

$$E_a := E_a \setminus \{\pi(b)\}, \ \ E_b := E_b \setminus \{\pi(a)\}. \tag{9.73}$$

To z kolei implikuje, że ograniczenie "$\pi(a)$ musi być wykonane przed $\pi(b)$" nie należy ani do Y^a ani do Y^b. Ponieważ istnieje co najmniej jedno zadanie $j \in E_a \setminus \{\pi(b)\}$ oraz co najmniej jedno zadanie $j \in E_b \setminus \{\pi(a)\}$, zatem $Y_b \neq \emptyset$ oraz $Y_a \neq \emptyset$. W konsekwencji twierdzenie blokowe pozostaje ważne, ponieważ dla tego przypadku mamy jeszcze $\Delta_a(s_1) \geqslant 0$ oraz $\Delta_b(s_v) \geqslant 0$, wymagane przez dowód wymienionego twierdzenia.

(B) "$\pi(b) \in E_a$, $\pi(a) \in E_b$ oraz warunki przypadku (A) nie są spełnione"

W tym przypadku ograniczenia kolejnościowe $(\pi(a), \pi(b))$ są zawarte w Y_a oraz Y_b. Aby uniknąć powielania ograniczeń kolejnościowych możemy przyjąć albo

$$E_a := E_a \setminus \{\pi(b)\} \tag{9.74}$$

albo

$$E_b := E_b \setminus \{\pi(a)\}. \tag{9.75}$$

Poniważ w naszym algorytmie zbiór Y_a jest ustalony przy przesuwaniu zadań $j \in E_b$ na pozycję a, przyjmujemy (9.75).

(C) "$E_b = \{\pi(a)\}$, $E_a \setminus \{\pi(b)\} \neq \emptyset$ oraz warunki przypadku (A) oraz (B) nie są spełnione"

Argumenty podobne jak poprzednio pozwalają nam przyjąć $E_b := \emptyset$.

(D) "$E_a = \{\pi(b)\}$, $E_b \setminus \{\pi(a)\} \neq \emptyset$ oraz warunki przypadku (A) oraz (B) nie są spełnione"

Argumenty podobne jak poprzednio pozwalają nam przyjąć $E_a := \emptyset$.

Implementacja relacji

Informacje zawarte w relacji Y mogą zostać wykorzystane do modyfikacji wartości r_j oraz q_j, które z kolei mają wpływ na zmniejszenie liczby kandydatów w zbiorach E_a, E_b oraz na wzmocnienia wartości dolnego ograniczenia przy jednoczesnym uproszczeniu techniki jego wyznaczania. Możliwe są przy tym dwa odmienne podejścia: (a) bazujące na całej relacji Y (technikę tą opisano szczegółowo w końcowej części Rozdz. 9.4), (b) realizujące modyfikację przyrostowo dla $Y \setminus Y^r$. Ta ostatnia technika zależy od Y^r, które z kolei zależy od przyjętego schematu podziału. W naszym przypadku przyjmuje ona następującą postać.

Niech j będzie zadaniem, które przesunięto na pozycję a w π. Wówczas wprowadzenie relacji $Y_a(j)$ odpowiada modyfikacji przyrostowej odpowiednio do wzorów

$$r_i := \max\{r_i, r_j + p_j\},\ j \in E'_a,\ i \neq j, \tag{9.76}$$

$$q_j := \max\{q_j, \sum_{t \in E'_a} p_t - p_j + \overline{q}^*\}, \tag{9.77}$$

gdzie

$$\overline{q}^* = \min_{t \in E'_a \setminus \{j\}} q_t \tag{9.78}$$

Niech j będzie zadaniem, które przesunięto na pozycję b w π. Wówczas wprowadzenie relacji $Y^a \cup Y_b(j)$ odpowiada modyfikacji przyrostowej wykonywanej dwustopniowo. Wpierw wprowadzamy modyfikację skojarzoną z Y^a odpowiednio do wzorów

$$r_i := \max\{r_i, r_{\pi(a)} + p_{\pi(a)}\},\ j \in E_a, \tag{9.79}$$

$$q_{\pi(a)} := \max\{q_{\pi(a)}, \sum_{t \in E_a} p_t + q^*\}, \tag{9.80}$$

gdzie

$$q^* = \min_{t \in E_a} q_t. \tag{9.81}$$

Następnie wprowadzamy modyfikację skojarzoną z $Y^b(j)$ odpowiednio do wzorów

$$r_j := \max\{r_j, \sum_{t \in E'_b} p_t - p_j + \overline{r}^*\}, \tag{9.82}$$

$$q_i := \max\{q_i, p_j + q_j\},\ j \in E'_b,\ i \neq j, \tag{9.83}$$

gdzie

$$\overline{r}^* = \min_{t \in E'_b \setminus \{j\}} r_t. \tag{9.84}$$

Zauważmy, że wzrost wartości otrzymany w drugim kroku może implikować kolejny wzrost wartości w kroku pierwszym, zatem wykonanie jednego dodatkowego cyklu modyfikacji może dostarczyć istotnie lepszych rezultatów.

Dolne ograniczenia

W celu uproszczenie (ujednolicenie) techniki wyznaczania dolnego ograniczenia jak również w celu zmniejszenie złożoności obliczeniowej odpowiedniej procedury numerycznej ograniczenia są wyznaczane dla problemu $1|r_j, q_j|C_{\max}$ ze zmodyfikowanymi wartościami r_j, q_j zamiast dla problemu $1|r_j, q_j, prec|C_{\max}$ z relacją Y oraz oryginalnymi wartościami r_j, q_j. Wszystkie metody opisane w Rozdz. 9.6 mogą być tu zastosowane. Niemniej pewne dodatkowe podejścia są możliwe dzięki wykorzystaniu własności blokowych.

Oprócz prostego ograniczenia (9.22), obliczanie obu dolnych ograniczeń Opartych na zrelaksowanych problemach jednomaszynowych wydaje się być nadmiernie kosztowne. Oznaczmy przez $LB2r$ oraz $LB2q$ optymalne wartości funkcji celu odpowiednich problemów $1|r_j = r(\mathcal{J}), q_j|C_{\max}$ oraz $1|r_j, q_j = q(\mathcal{J})|C_{\max}$. Proponujemy zastosować tylko $LB2q$ jeśli $\max_{i\in\mathcal{J}} r_i - \min_{i\in\mathcal{J}} r_i < \max_{i\in\mathcal{J}} q_i - \min_{i\in\mathcal{J}} q_i$ oraz tylko $LB2r$ w przeciwnym przypadku.

Kolejne dolne ograniczenie na $C_{\max}(\gamma)$, $\gamma \in \Pi(\beta_r)$, Gdzie β_r jest pewnym bezpośrednim następnikiem π, wynika z rozważań przeprowadzonych poniżej. Wybierzmy do konstrukcji ograniczenia pewien specyficzny podzbiór zadań $I \subseteq \mathcal{J}$ a mianowicie $I = B$. Jest oczywistym, że wartość $r(B)+p(B)+q(B)$ jest także dolnym ograniczeniem. Będziemy jednak dążyć do konstrukcji lepszego oszacowania. Zakładając, że β_r została otrzymana z π przez przesunięcie zadania $j \in E_a$ na pozycję a, to dolnym ograniczeniem jest

$$LB3a(j) = r_j + p(B) + q(B). \tag{9.85}$$

Dodając i odejmując $r_{\pi(a)}$ i $q_{\pi(b)}$ otrzymujemy

$$LB3a(j) = r_j - r_{\pi(a)} + r_{\pi(a)} + p(B) + q_{\pi(b)} - q_{\pi(b)} + q(B)$$

$$= \Delta_a(j) + C_{\max}(\pi) + \min_{i \in B} \Delta_b(i). \tag{9.86}$$

Obliczenie LB3a(j) wymaga $O(n)$ kroków dla każdego π. Przez analogię, Zakładając, że β_r została otrzymana z π przez przesunięcie zadania $j \in E_a$ na pozycję a, to dolnym ograniczeniem jest

$$LB3b(j) = r_{\pi(a)} = p(B) + q_j = r_{\pi(a)} + p(B) + q_{\pi(b)} - q_{\pi(b)} + q_j$$

$$= C_{\max}(\pi) + \Delta_b(j). \tag{9.87}$$

Strategia poszukiwania

Proponowany algorytm oparty na schemacie podziału i ograniczeń używa strategii "w głąb" z powrotami. Wartość górnego ograniczenia jest aktualizowana w każdym węźle drzewa rozwiązań według zasady $UB := \min\{UB, C_{\max}(\pi)\}$. Z bieżącego węzła w drzewie rozwiązań, węzły potomne są sprawdzane w kolejności zgodnej z niemalejącymi wartościami $\Delta_x(j)$, gdzie $j \in E^x$, $x \in \{a, b\}$. Taka strategia umożliwia szybką poprawę wartości górnego ograniczenia.

9.10 Uwagi

Oba opisane algorytmy dokładne, mimo iż odnoszą się do problemu NP-trudnego, powszechnie są uważane za jedne z bardziej efektywnych schematów B&B, bowiem umożliwiają rozwiązywanie bez trudu przykładów o $n \leqslant 1,000$. Z tego względu stosowane są jako pomocnicze dla bardziej złożonych zagadnień. Stanowią bazę, między innymi, dla jednomaszynowych problemów rozszerzonych (dyskutowanych dalej), niektórych problemów z nieregularnymi funkcjami celu, NP-trudnych dolnych ograniczeń problemów gniazdowych, metody przesuwanego wąskiego gardła dla problemu gniazdowego.

Jakość metod przybliżonych

Porównanie teoretycznych własności algorytmów z Rozdz. 9.7.1– 9.8 przedstawiono w Tabl. 9.10. Jak już wspomniano, przeprowadzona analiza teoretyczna nie charakteryzuje całkowicie tych algorytmów. W uzupełniającej analizie eksperymentalnej odrzucono schematy aproksymacyjne ze względu na znaczną złożoność obliczeniową. Do badań wykorzystano losowe przykłady testowe generowane według schematu opisanego w pracy [51]. Dla

algorytm	η	złożoność obliczeniowa
R,Q	2	$O(n \log n)$
S	2	$O(n \log n)$
P	3/2	$O(n^2 \log n)$
HS	4/3	$O(n^2 \log n)$
NS	3/2	$O(n \log n)$
A-1	$1+\epsilon$	$O(16^{1/\epsilon}(n/\epsilon)^{3+4/\epsilon})$
A-2	$1+\epsilon$	$O(n \log n + n(4/\epsilon)^{8/\epsilon^2+8/\epsilon+2})$

Tabela 9.6: Algorytmy dla problemu $1|r_j, q_j|C_{\max}$

każdego n=50,100,150,.. ..,1000 oraz parametru $F = 16, 17, 18, ..., 25$ generowano próbkę 20 przykładów; wartości r_j, p_j, q_j były generowane z równomiernym rozkładem prawdopodobieństwa pomiędzy 1 oraz $r_{\max}$, $p_{\max}$, $q_{\max}$ odpowiednio. Przyjęto $p_{\max} = 50$, $r_{\max} = q_{\max} = nF$. Łącznie wygenerowano 4000 przykładów. Przykłady z $F = 18, 19, 20$ uważane są za najtrudniejsze [51]. Algorytm HS dostarczył rozwiązanie optymalne w 95% przypadków. Algorytm NS dostarczył lepszych rozwiązań niż algorytm S (tzn. $C^{NS} < C^S$) w 89 procentach przypadków (3572 przykładów). Ponieważ algorytm NS ma teoretyczną ocenę jakości gorszą niż HS, można się spodziewać podobnej zależności także w porównawczej analizie eksperymentalnej obu algorytmów. Potwierdzają to również wykonane testy komputerowe. Ponieważ algorytm NS ma konkurencyjną n -krotnie mniejszą złożoność obliczeniową niż HS, istotnym jest czy NS dostarcza rozwiązań bliższych S czy HS. W celu odpowiedzi na to pytanie w przeprowadzonym teście obliczono błąd względny

$$\eta^{HS} = \frac{C^S - C^{NS}}{C^S - C^{HS}} \tag{9.88}$$

dla każdego przykładu takiego, że $C^S > C^{HS}$. Stwierdzono zachodzenie nierówności $C^S > C^{HS}$ w 94 procentach przypadków (3750 przykładów). Odpowiedni średnia wartość błędu względnego dla tych przykładów wynosiła 0.86. Otrzymaliśmy $\eta^{HS} = 1$ (tzn. $C^{NS} = C^{HS}$) dla 2840 przykładów (76 procent z 3750). Ponieważ $C^P \geqslant C^{HS}$, zatem $\eta^P \geqslant \eta^{HS}$. Oznacza to, że algorytm H generuje rozwiązania bliższe tym z algorytmów P i HS niż temu z algorytmu S chociaż ma n -krotnie mniejszą złożoność obliczeniową.

Rozszerzenia

Korzystając z równoważności kryteriów optymalizacji, większość otrzymanych wyników można przenieść bezpośrednio lub z niewielkimi zmianami na kryteria klasy min-max z Rozdz. 4.3, patrz na przykład praca [392] dotycząca problemu $1|r_j, prec|F_{\max}$. Dalsze rozszerzenia w kierunku zagadnień inspirowanych praktyką dotyczą problemu $1|r_j, q_j, prec|C_{\max}$, w którym dla par zadań związanych relacją poprzedzań $\mathcal{R}$ są określone dolne (górne) ograniczenia czasów oczekiwania, tak zwane *okna czasowe* (time windows). Skrajnie ostre wymagania tego typu prowadzą do żądań *bez czekania* (no wait) lub ze ściśle określonym okresem oczekiwania dla wybranych par zadań. Wyznaczenie (istnienie) jakiegokolwiek rozwiązania dopuszczalnego w takim przypadku jest problemem nietrywialnym. Oprócz bezpośrednich zastosowań praktycznych, problemy tego typu są generowane w schemacie dolnych ograniczeń gniazdowego problemu szeregowania oraz w metodzie przesuwanego wąskiego gardła [86]. Niezależnie od powyższego, ostatni wymieniony problem przy założeniu, że $\mathcal{R}$ jest sumą rozłącznych łańcuchów niewielkiej długości (dwa, trzy), znany jako problem z *opóźnieniami* (time lags), ma zastosowanie w systemach wytwarzania, w tym FMS [148]. Problemy z kryteriami klasy min-max stanowią także bazę dla analizy niektórych problemów z kryteriami nieregularnymi klasy min-max, przy czym związki są dość złożone [43,344]. Oddzielną grupę problemów stanowią uogólnienia dopuszczajace występowanie przezbrojeń stanowiska pomiędzy wykonywanymi zadaniami.

10

Kosztowe problemy jednomaszynowe

Problemy jednomaszynowe z kryterium regularnym, addytywnym klasy sum są zdecydowanie trudniejsze od problemów z porzedniego rozdziału, posiadają mniej szczególnych własności oraz wymagają całkowicie odmiennych metod rozwiązywania. Przykłady ich zastosowań są dość liczne, zaś kryteria addytywne mają dobre uzasadnienie ekonomiczne. Rozpoczynając od sformułowania problemu podstawowego, będziemy stopniowo wprowadzać kolejne rozszerzenia i uogólnienia.

Rozpatrzmy następujacy problem wyjściowy. Należy wyznaczyć harmonogram pracy stanowiska zawierającego jedną maszynę przy założeniu, że zbiór zadań $\mathcal{J} = \{1, 2, \ldots, n\}$ ma być wykonywany na stanowisku sekwencyjnie, j-te zadanie posiada termin gotowości r_j, czas obsługi $p_j > 0$ oraz niemalejącą funkcję $f_j(t)$ reprezentującą koszt związany z zakończeniem wykonywania tego zadania w chwili czasowej t. Harmonogram jest określony poprzez wektor terminów rozpoczeęcia wykonywania zadań $S = (S_1, \ldots, S_n)$, zaś kryterium polega na minimalizacji wartości wskaźnika $\sum_{j=1}^{n} f_j(C_j)$, gdzie $C_j = S_j + p_j$ jest terminem zakończenia wykonywania zadania j. Poszukiwany harmonogram musi również przestrzegać wymagań technologicznego porządku realizacji zadań danego w formie relacji poprzedzań $\mathcal{R} \subset \mathcal{J} \times \mathcal{J}$.

Problemy ze znanymi wskaźnikami jakości funkcjonowania stanowiska można otrzymać zakładając szczególną postać funkcji $f_j(t)$. Tak np. dla $f_j(t) = \frac{1}{n}(t - r_j)$ poszukujemy rozwiązania z kryterium minimalnego średniego czasu przepływu elementów przez stanowisko, dla $f_j(t) = w_j(t - r_j)$ – z kryterium ważonego czasu przepływu, dla $f_j(t) = \max\{0, t - d_j\}$ – z kryterium łącznego spóźnienia względem żądanych terminów zakończenia d_j,

itp.

Relacja $\mathcal{R}$, która generuje ograniczenia postaci $S_i + p_i \leqslant S_j$, $(i,j) \in \mathcal{R}$, może być wyrażona rownoważnie poprzez zbiory bezpośrednich następników

$$\mathcal{A}_j(\mathcal{R}) = \{i \in \mathcal{N} : \ (j,i) \in \mathcal{R}\} \tag{10.1}$$

i/ lub bezpośrednich poprzedników

$$\mathcal{B}_j(\mathcal{R}) = \{i \in \mathcal{N} : \ (i,j) \in \mathcal{R}\}, \tag{10.2}$$

$j = 1, \ldots, n$. Ze względu na regularność funkcji celu, rozwiązania optymalnego należy poszukiwać w zbiorze harmonogramów dosuniętych w lewo na osi czasu. Zatem każde rozwiązanie może być reprezentowane jednoznacznie permutacją $\pi = (\pi(1), \ldots, \pi(n))$ na zbiorze $\mathcal{N}$. Permutacja jest dopuszczalna jeśli dla każdej pary (i,j) takiej, że $(\pi(i), \pi(j)) \in \mathcal{R}$ zachodzi $i < j$. Oczywiście zbiór rozwiązań dopuszczalnych ma liczność nie wieksza niż $n!$. Dla danej dopuszczalnej π terminy rozpocęcia S_j, $j \in \mathcal{N}$ można zanaleźć w czasie $O(n)$ używając oczywistej zależności

$$S_{\pi(j)} = \max\{r_{\pi(j)}, S_{\pi(j-1)} + p_{\pi(j-1)}\}, \ \ j = 1, 2, \ldots, n, \tag{10.3}$$

gdzie $\pi(0) = 0$ i $S_0 = 0 = p_0$). Odpowiednie terminy zakończenia dla danej S są określone jednoznacznie i równe $C_j = S_j + p_j$, $j \in \mathcal{N}$.

Dla pewnych szczególnych postaci funkcji $f_j(t)$ problem można rozwiązać w czasie wielomianowym. W ogólnym przypadku problem jest silnie NP-trudny.

10.1 Przypadki wielomianowe

Zauważmy wpierw, że przypadek $r_j = const$, $j = 1, \ldots, n$, jest sprowadzalny do $r_j = 0$, $j = 1, \ldots, n$, poprzez odpowiednie przesunięcie czasu. Wszystkie wymienione poniżej przypadki szczególne odnoszą się do sytuacji $r_j = 0$, $j = 1, \ldots, n$. Najpierw rozważmy problemy dla których $\mathcal{R} = \emptyset$. Jeśli funkcje $f_j(t)$ są postaci $f_j(t) = t$ (kryterium $\sum C_j$ oraz $\overline{C}$) to optymalną kolejność można uzyskać poprzez uszeregowanie zadań zgodnie z niemalejącymi wartościami p_j (reguła SPT). Jeśli funkcje $f_j(t)$ są postaci $f_j(t) = w_j t$ (kryterium $\sum w_j C_j$) to optymalną kolejność można uzyskać poprzez uszeregowanie zadań zgodnie z niemalejącymi wartościami p_j/w_j (reguła WSPT). Rezultat ten pozostaje ważny dla dowolnych liniowych funkcji $f_j(t) = w_j t + v_j$. Został on w dalszym ciągu uogólniony na funkcje $f_j(t)$

quasi-liniowe postaci

$$f_j(t) = \int_{t-p_j}^{t} (\alpha_j \phi(u) + \beta_j) du \tag{10.4}$$

gdzie $g(u)$ jest ciągłą, monotoniczną funkcją gęstości kosztu. W tym ostatnim przypadku optymalną kolejność można otrzymać poprzez ich uszeregowanie zadań zgodnie z nierosnących wartościami w_j.

Kolejne przypadki odnoszą się do funkcji kosztów nie wymienionych powyżej przy założeniu $\mathcal{R} = \emptyset$ oraz $p_j = const$, $j = 1, \ldots, n$. Problem ten można sprowadzić, poprzez skalowanie czasu, do przypadku $p_j = 1$, $j = 1, \ldots, n$. Jego rozwiązanie można uzyskać wyznaczając optymalny przydział zadań do jednostkowych przedziałów czasowych

$$\min_{\pi} \sum_{j=1}^{n} f_{\pi(j)}(j) \tag{10.5}$$

Problemu przydziału posiada algorytm wielomianowy $O(n^3)$. Rezultat ten nie daje się uogólnić na przypadek niepustej $\mathcal{R}$.

10.2 Model całkowitoliczbowy

Problem można sformułować jako zadanie optymalizacji nieliniowej zawierającej zmienne decyzyjne ciągle i dyskretne, tzw. zadanie mieszane. Niektóre z tak postawionych zadań, przy założeniu szczególnych postaci funkcji $f_j(t)$, mogą być sprowadzone do zadań PL, PLC, PLB rozwiązywanych następnie odpowiednimi metodami ogólnymi omawianymi wcześniej. Odpowiedni model ogólny ma postać

$$\min_{S} \sum_{j=1}^{n} f_j(S_j + p_j) \tag{10.6}$$

$$(S_i + p_i \leqslant S_j) \dot{\vee} (S_j + p_j \leqslant S_i), \quad i, j \in \mathcal{N}, \quad i \neq j, \tag{10.7}$$

$$S_i + p_i \leqslant S_j, \quad (i, j) \in \mathcal{R}, \tag{10.8}$$

$$r_j \leqslant S_j, \quad j = 1, \ldots, n. \tag{10.9}$$

Warunek dysjunktywny (wyłączająca alternatywa) (10.7) stanowi, że dla każdych dwóch zadań $i, j \in \mathcal{N}$ zachodzi albo "i poprzedza j" albo "j poprzedza i". Warunki takie należy wprowadzić dla każdego zbioru dwuelementowego zadań $\{i, j\}$, $i, j \in \mathcal{N}$, $i \neq j$. Niewygodny zapis przebiegania

po indeksach w prawej części wzoru (10.7) można zastąpić, bez straty ogólności, prostszym zapisem $1 \leqslant i < j \leqslant n$ lub równoważnie $j = i+1, \ldots, n$, $i = 1, \ldots, n$. Warunek (10.7) można przedstawić w formie następujących ograniczeń liniowych. Niech y_{ij} będzie zmienną binarną określoną następująco $y_{ij} = 1$ jeśli i poprzedza j oraz zero w przeciwnym przypadku, $j = i+1, \ldots, n$, $i = 1, \ldots, n$. Wynikowy problem optymalizacji binarno-ciągłej ma postać

$$\min_{S} \sum_{j=1}^{n} f_j(S_j + p_j), \tag{10.10}$$

$$S_i + p_i - S_j \leqslant K(1 - y_{ij}), \;\; j = i+1, \ldots, n, \;\; i = 1, \ldots, n, \tag{10.11}$$

$$S_j + p_j - S_i \leqslant K y_{ij}, \;\; j = i+1, \ldots, n, \;\; i = 1, \ldots, n, \tag{10.12}$$

$$S_i + p_i \leqslant S_j, \;\; (i,j) \in \mathcal{R}, \tag{10.13}$$

$$r_j \leqslant S_j, \;\; j = 1, \ldots, n, \tag{10.14}$$

$$y_{ij} \in \{0,1\}, \;\; j = i+1, \ldots, n, \;\; i = 1, \ldots, n, \tag{10.15}$$

gdzie K jest pewną dostatecznie dużą liczbą (można przyjąć np. $K = \sum_{j=1}^{n} p_j + \max_{1 \leqslant j \leqslant n} r_j$). Problem posiada n zmiennych ciągłych, $n(n-1)/2$ zmiennych binarnych oraz $n(n-1) + n + |\mathcal{R}|$ liniowych warunków ograniczających.

Jeden z najczęściej rozważanych problemów $1|r_j, prec| \sum w_j T_j$, w którym funkcje kosztu są postaci $f_j(t) = w_j \max\{0, t - d_j\}$, może być przedstawiony w formie zadania programowania liniowego binarno-ciągłego. Mimo, iż odpowiednie metody ogólne mogą byż zastosowane do jego rozwiązania, prezentowane podejście uważane jest za jedno z mniej efektywnych ze względu na rozmiar dostarczanego modelu oraz kłopoty z uzyskaniem jakiegokolwiek rozwiązania dopuszczalnego w przypadku konieczności przedwczesnego przerwania pracy algorytmu.

10.3 Algorytm PD

Wykładnicza wersja schematu PD jest dedykowana dla NP-trudnych przypadków problemu $1|\sum f_i$. Podstawowa wersja algorytmu jest formułowana dla przypadku $r_j = 0$, $j \in \mathcal{N}$ oraz $\mathcal{R} = \emptyset$ [23]. Korzysta się przy tym w sposób znaczący z faktu, że problem taki posiada ustalony końcowy punkt horyzontu polanowania równy $\sum_{j=1}^{n} p_j$. Rozszerzenia na przypadek niepustej $\mathcal{R}$ [327] oraz różnych r_j zostaną omówione w dalszej części rozdziału.

Oznaczmy przez $F(\mathcal{I})$ minimalną wartość funkcji celu $\sum_{j\in\mathcal{I}} f_j(C_j)$ którą można osiągnąć szeregując zadania ze zbioru $\mathcal{I} \subseteq \mathcal{J}$. Wartości $F(\mathcal{I})$, $\mathcal{I} \subseteq \mathcal{J}$ można policzyć korzystając z następującej zależności rekurencyjnej

$$F(\mathcal{I}) = \min_{k\in\mathcal{I}} F(\mathcal{I} \setminus \{k\}) + f_k(p(\mathcal{I})), \;\; \mathcal{I} \subseteq \mathcal{N} \tag{10.16}$$

gdzie

$$p(\mathcal{I}) = \sum_{j\in\mathcal{I}} p_j \tag{10.17}$$

oraz $F(\emptyset) = 0$. Celem jest wyznaczenie wartości $F(\mathcal{J})$, która jest optymalną wartością kryterium dla postawionego problemu. Aby otrzymać optymalny harmonogram pracy stanowiska należy równolegle z wyznaczaniem $F(\mathcal{I})$ śledzić wartość $v(\mathcal{I})$ zdefiniowaną następująco

$$v(\mathcal{I}) = k^* \text{ takie, że } F(\mathcal{I} \setminus \{k^*\}) + f_{k^*}(p(\mathcal{I})) = F(\mathcal{I}). \tag{10.18}$$

Wielkość $v(\mathcal{I})$ jest indeksem k dla którego osiągane jest minimum po prawej stronie wzoru (10.16). Następnie po wyznaczeniu $F(\mathcal{J})$ oraz $v(\langle\rangle)$, $\mathcal{I} \subseteq \mathcal{N}$, przeprowadzany jest proces konstrukcji harmonogramu optymalnego. W tym celu startując od $\mathcal{I} = \mathcal{J}$ wykonujemy podstawienia $\pi^*(|\mathcal{I}|) := v(\mathcal{I})$ oraz $\mathcal{I} := \mathcal{I} \setminus \{v(\mathcal{I})\}$ tak długo aż $\mathcal{I} \neq \emptyset$. Optymalne terminy zakończenia wykonywania zadań można zanaleźć używając wzoru (10.3) dla optymalnej kolejności π^* wyznaczonej powyżej. Ponieważ do konstrukcji harmonogramu optymalnego potrzebne są tylko pewne jednakże nieznane apriori wartości $v(\mathcal{I})$, można zrezygnować z pamiętania wszystkich $v(\mathcal{I})$ na rzecz ich odtwarzania na żądanie z $F(\mathcal{I})$.

Podany algorytm wymaga $O(2^n)$ komórek pamięci na przechowywanie danych oraz czasu obliczeń rzędu $O(n2^n)$. Można podać bardziej oszczędną implementację algorytmu wymagająca tylko $\binom{n}{a}$, $a = [n/2]$, komórek pamięci na przechowywanie wyników $F(\mathcal{I})$, jednakże oprócz znacznego skomplikowania struktury programu nie uzyskamy znaczącej poprawy ogólnych cech numerycznych algorytmu. Kluczem do podstawowej implementacji jest technika przeprowadzenia nietypowej rekurencji przebiegającej po podzbiorach zbioru. Każdy zbiór $\mathcal{I}$ może być odwzorowany wzajemnie jednoznacznie w liczbą binarną (etykietę) $L(\mathcal{I}) = \sum_{j\in\mathcal{I}} L(j)$, gdzie $L(j) = 2^{j-1}$ jest etykietą pojedynczego elementu. Wszystkim pozbiorom zbioru $\mathcal{N}$ zostaną w ten sposób przypisane w sposób unikalny etykiety o wartościach z $\{0, \ldots, 2^n - 1\}$. Etykieta $L(\mathcal{I})$ określa lokalizację tj. indeks w tablicy jednowskaźnikowej pod którym należy umieścić $F(\mathcal{I})$ oraz $v(\mathcal{I})$. Wzór (10.16) wymaga aby wartość

$F(\mathcal{I} \setminus \{k\})$ była policzona przed $F(\mathcal{I})$ dla każdego k oraz $\mathcal{I}$. Można to zrealizować generując ciąg podzbiorów $\mathcal{I}^0$,$\mathcal{I}^1$, ..., taki, że $\mathcal{I}^0 = \emptyset$ oraz $\mathcal{I}^{s+1}$ został otrzymany z $\mathcal{I}^s$ realizując następujące postępowanie. Dla danego $s \in \mathcal{S}$, niech i oznacza minimalny indeks taki, że $i \notin \mathcal{I}^s$. Jeśli taki indeks nie istnieje to musi zachodzić $\mathcal{I}^s = \mathcal{N}$. Odpowiedni ciag podzbiorów określamy wzorem $\mathcal{I}^{s+1} = (\mathcal{I}^s \setminus \{j : 1 \leqslant j < i\}) \cup \{i\}$. łatwo zauważyć, że wartości etykiet dla podzbiorów ciągu są kolejnymi liczbami naturalnymi, tzn. $L(\mathcal{I}^s) = s$ $s = 0, 1, \ldots, 2^n - 1$. Zatem bądź rekurencja przebiega po ciągu podzbiorów $\mathcal{I}^s$ okreśslonym powyżej, bądź też po etykietach tych podzbiorów bowiem każda etykieta jednoznacznie identyfikuje podzbiór.

Rozszerzenie podanego podejścia dla niepustej $\mathcal{R}$ wymaga nieznacznej modyfikacji wzoru rekurencyjnego (10.16) oraz znaczącej modyfikacji strony organizacyjnej algorytmu. Wzór (10.16) przyjmuje w tym przypadku postać

$$F(\mathcal{I}) = \min_{k \in K(\mathcal{I})} F(\mathcal{I} \setminus \{k\}) + f_k(p(\mathcal{I})), \;\; \mathcal{I} \subseteq \mathcal{J} \tag{10.19}$$

gdzie $K(\mathcal{I})$ jest zbiorem zadań z $\mathcal{I}$ nie posiadających następników w zbiorze $\mathcal{I}$, tzn. $K(\mathcal{I}) = \{j \in \mathcal{I} : \quad \mathcal{A}_j \cap \mathcal{I} = \emptyset\}$. Zauważmy, zę wzór (10.19) może być stosowany tylko dla dopuszczalnych podzbiórów $\mathcal{I}$, tzn. takich że dla każdego $j \in \mathcal{I}$ zachodzi $B_j \subset \mathcal{I}$. Wprowadzenie niepustej $\mathcal{R}$ jest korzystne bowiem pominijanie niedopuszczalnych podzbiorów przyśpiesza pracę algorytmu oraz zmniejszajsza jego żądania pamięciowe. Aby tą własność stosownie wykorzystać załóżmy wpierw, bez zmniejszenia ogólności, że zadania są indeksowane tak że dla każdego $(k, l) \in \mathcal{R}$ zachodzi $k < l$. Odpowiedni ciąg podzbiorów $\mathcal{I}^s$ określamy następująco $\mathcal{I}^{s+1} = (\mathcal{I}^s \setminus \{j : 1 \leqslant j < i, \quad j \in K(\mathcal{I}^s)\}) \cup \{i\}$, gdzie i jest indeksem zdefiniowanym dla $\mathcal{I}^s$ jak poprzednio. Liczba podzbiorów w ciągu zależy od postaci relacji $\mathcal{R}$. Użyta poprzednio technika adresowania pamięci danych dla $F(\mathcal{I})$ poprzez bezpośrednie użycie etykiet zbiorów nie powinna być tutaj stosowana, bowiem wymaga pamięci o wielkości wykładniczej przy jednoczesnym stosunkowo małym poziomie jej wykorzystania. Zamiast tej techniki proponuje się zastosowanie bardziej wyrafinowanych struktur pamięciowych, np. tablicy haszowej, do przechowywania wartości $F(\mathcal{I})$. Odpowiednia funkcja haszująca może przyjąć postać $h(\mathcal{I}) = L(\mathcal{I}) \; mod \; M$, gdzie mod jest operatorem dzielenia modulo, $L(\mathcal{I})$ jest etykietą zdefiniowaną w poprzednim paragrafie zaś M jest wielkością tablicy haszowej, np. $M = (2^n d)/((2^n - 1)\,|\mathcal{R}| + d)$, $d = n(n-1)/2$. Z kolei w pracy [327] zaproponowano specyficzny sposób generowania oryginalnych etykiet odpowiadających adresom zlokalizowanym w prawie-zwartym niewielkim obszarze pamięci. Każde zadanie j otrzymuje przypisaną etykietę $L(j)$ taką, że etykieta dowolnego podzbioru $\mathcal{I}^s$ z ciągu jest unikalna i rów-

na $L(\mathcal{I}^s) = \sum_{j\in\mathcal{I}^s} L(j)$. Zakładając numerację zadań spełniającą warunek wymieniony na początku tego paragrafu, etykiety są określone następująco $L(j) = t(j) - a(j) - b(j) + 1$, gdzie $t(j) = \sum_{k=1}^{j-1} L(j)$, $a(j) = \sum_{k=1;\ j\in\mathcal{A}_j^*}^{j-1} L(j)$, $b(j) = \sum_{k=1;\ j\in\mathcal{B}_j^*}^{j-1} L(j)$ dla $j = 1, \ldots, n$, gdzie $\mathcal{A}_j^*$ i $\mathcal{B}_j^*$ są tranzytywnymi domknięciami zbiorów $\mathcal{A}_j$ i $\mathcal{B}_j$ odpowiednio.

Rozszerzenie początkowego (podstawowego) podejścia na przypadek różnych terminów gotowości jest zdecydowanie bardziej kłopotliwe. Niech $F(\mathcal{I}, t)$ oznacza minimalną wartość funkcji celu $\sum_{j\in\mathcal{I}} f_j(C_j)$ którą można osiągnoąć szeregując zadania ze zbioru $\mathcal{I} \subseteq \mathcal{J}$ przy ograniczeniu, zę wszystkie zadania zostaną wykonane przed terminem t. Niech T oznacza horyzont czasowy harmonogramu, np. $T = \max_{j\in\mathcal{J}} r_j + p(\mathcal{J})$. Wówczas zachodzi

$$F(\mathcal{I}, t) = \min\{F(\mathcal{I}, t-1), \min_{k\in\mathcal{I}}(F(\mathcal{I}\backslash\{k\}, t-p_k) + g_k(t))\},\ 0 < t \leqslant T,\ \mathcal{I} \subseteq \mathcal{J}, \tag{10.20}$$

gdzie

$$g_j(t) = \begin{cases} \infty & dla \quad r_j + p_j > t \\ f_j(t) & dla \quad r_j + p_j \leqslant t \end{cases} \tag{10.21}$$

zaś $F(\emptyset, t) = 0$, $0 \leqslant t \leqslant T$, $F(\mathcal{I}, 0) = \infty$, $\mathcal{I} \subset \mathcal{J}$, $\mathcal{I} \neq \emptyset$. Celem naszym jest wyznaczenie $F(\mathcal{N}, T)$. Proces obliczania wzoru (10.20) można przyśpieszyć wykorzystując pewne oczywiste zależności. I tak, $F(\mathcal{I}, t) = \infty$ dla $0 < t < J(\mathcal{I})$, gdzie $J(\mathcal{I})$ jest minimalnym terminem zakończenia wykonywania wszystkich zadań ze zbioru $\mathcal{I}$. Wartość $J(\mathcal{I})$ można otrzymać szeregując elementy zbioru $\mathcal{I}$ wg niemalejących wartości r_j. Dalej, zachodzi $F(\mathcal{I}, t) = F(\mathcal{I}, t-1)$ dla $t > \max_{j\in\mathcal{I}} r_j + p(\mathcal{I})$. Niestety, mimo podanych udogodnień, złożoność obliczeniowa algorytmu jest ogólnie rzędu $O(Tn2^n)$, zaś żądania pamięciowe – $O(T2^n)$. Dodatkowo, otwartym pozostaje sposób adresacji pamięci w celu efektywnego jej wykorzystania.

10.4 Algorytm B&B

Implementację algorytmu opiszemy dla przypadku $r_j = 0$, $j \in \mathcal{J}$, oraz $\mathcal{R} = \emptyset$, bowiem uogólnienia nie zmieniają istotnie jej charakteru. Następnie naszkicujemy tylko sposób jej rozszerzenie na przypadek $\mathcal{R} \neq \emptyset$. W algorytmie tym korzysta się z faktu, że pojedyncze rozwiązanie może być reprezentowane permutacją na zbiorze $\mathcal{J}$. Dla potrzeb schematu B&B odwołamy się więc do drzewa rozwiązań dla problemów permutacyjnych. Istnieje wiele schematów budowy takich drzew, tutaj ograniczymy się do najczęściej używanych i stosunkowo prostych drzew regularnych opartych na porządkach cześciowych, budowanych począwszy od pozycji pierwszej w permutacji.

Węzeł 0, korzeń drzewa, na poziomie zerowym, przedstawia wszystkie rozwiazania tj. wszystkie permutacje na $\mathcal{J}$. Z węzła 0 wychodzi n gałezi prowadzących do n węzłów poziomu pierwszego. Każdy z tych węzłów reprezentuje rozwiązania, w których na pierwszym miejscu znajduje się jeden element, różny dla różnych węzłów. Z każdego węzła na poziomie 1 wychodzi $n-1$ krawędzi do $n-1$ nowych wezłów na poziomie 2. Zatem poziom 2 posiada łącznie $n(n-1)$ węzłów. Każdy węzeł na poziomie 2 przedstawia zbiór rozwiazań, w którym na początku ustalono kolejność dwóch elementów.

Tak więc węzeł na l-tym poziomie drzewa rozwiązań, $l = 0, \ldots, n$ jest charakteryzowany przez permutację częściową zadań $\sigma = (\sigma(1), \ldots, \sigma(l))$ oraz odpowiada wszystkim rozwiazaniom problemu reprezentowanych przez permutacje identyczne z σ na l pierwszych pozycjach. W tym przypadku zbiór bezpośrednich następników węzła otrzymujemy umieszczając na pozycji $l+1$ nieuszeregowane jeszcze zadanie $j \in \mathcal{U} \stackrel{\text{def}}{=} \mathcal{J} \setminus \mathcal{S}$, gdzie $\mathcal{S} = \{\sigma(j) : j = 1, \ldots, l\}$ jest zbiorem zadań już uszeregowanych.

Z powyższego wynika, że każdemu węzłowi możemy przyporządkować relację poprzedzania $\mathcal{R}_\sigma$ z jądrem $\mathcal{R}^*_\sigma$ w postaci

$$\mathcal{R}^*_\sigma \stackrel{\text{def}}{=} \{(\sigma(j-1), \sigma(j)) : \ j = 2, \ldots, l\} \cup \{(\sigma(l), j); \ j \in \mathcal{U}\}. \qquad (10.22)$$

Samą zaś relację $\mathcal{R}_\sigma$ można otrzymać z $\mathcal{R}^*_\sigma$ dodając do $\mathcal{R}_\sigma$ wszystkie pary wynikające z przechodniości. Relacja $\mathcal{R}_\sigma$ jest ustalona w węźle odpowiadającym permutacji częściowej σ w tym sensie, że wymagania częściowego porządku musza być spełnione w każdym następniku względem σ w drzewie rozwiązań.

Z budowy drzewa wynika natychmiast, że liście (węzły terminalne) reprezentują wszystkie permutacje na zbiorze $\mathcal{J}$, to znaczy jest ich $n!$. Liczbę wzystkich węzłów w drzewie możemy oszacować jako

$$1 + n + n(n-1) + \ldots + n! = n(\frac{1}{n!} + \frac{1}{(n-1)!} + \ldots \frac{1}{1!}) \approx (e-1)n! \quad (10.23)$$

Istnieje "symetryczy" sposób tworzenia drzewa poprzez budowę częściowej permutacji od końca. Jest to niewątpliwie korzystniejsze w kontekście wyznaczania wartości dolnego ograniczenia oraz reguł eliminacji. W niektórych zastosowaniach używa się też schematu mieszanego budującego uszeregowanie zarówno od początku jak i od końca, zapewniającego, przy wykorzystaniu *adaptacyjnej* reguły wyboru, lepsze wyniki działania schematu B&B. Adaptacyjność reguły jest tu rozumiana jako dynamiczne podejmowanie decyzji dotyczacej kolejno szeregowanego zadania oraz kierunku jego umieszczania (pierwsza, ostatnia wolna pozycja).

Kryteria eliminacji

Rozszerzenie podejścia poprzez wprowdzenie niepustej początkowej relacji poprzedzań $\mathcal{R}$ jest zwykle korzystne, bowiem część węzłów drzewa odpowiadających rozwiązaniom niedopuszczalnym jest a priori eliminowana. W wielu przypadkach relację $\mathcal{R}$ można uzyskać poprzez zastosowanie *kryteriów eliminacyjnych*. Najczęściej mają one postać warunkó nakładanych na parę zadań i oraz j, przy spełnieniu których istniej rozwiązanie optymalne takie, że $i \to j$. Warunki te mogą odnosić się także do kombinacji zadanie - zbiór zadań. Kryteria eliminacji mogą być *globalne* oraz *lokalne*. Kryterium globalne odnosi się do całego zbioru rozwiązań, lokalne - do pewnego podzbioru. Kryteria lokalne wykorzystują głównie fakt istnienia częściowego uporządkowania zadań i pomagają w wyborze kolejnego zadania do uszeregowania. Kryteria eliminacji są zwykle formułowane specyficznie dla funkcji kryterialnych, bowiem w przypadkach szczególnych są bardziej skuteczne niż w przypadku ogólnym. Oznaczmy przez $P(K) = \sum_{s \in K} p_s$ dla pewnego ustalonego podzbioru $K \subseteq \mathcal{J}$, oraz przedstawmy aktualnie istniejącą relację $\mathcal{R}$ za pomocą zbioru bezpośrednich poprzedników $\mathcal{B}_j$ i następników $\mathcal{A}_j$. Zauważmy, że dowolne uszeregowanie powoduje wykonywanie zadań w sposób ciągły w przedziale $[0, P(\mathcal{J})]$.

W pracy [93] zebrano wyprowadzone wcześniej specyficzne kryteria dla problemu $1||\sum T_i$. Jeśli co najmniej jeden z poniższych warunków jest spełniony to istnieje rozwiązanie optymalne, w którym $i \to j$.

$$p_i \leqslant p_j,\ d_i \leqslant \max\{p_j + P(\mathcal{B}_j), d_j\} \tag{10.24}$$

$$p_i > p_j,\ d_i \leqslant d_j,\ d_j + p_j \geqslant P(| \setminus \mathcal{A}_i) \tag{10.25}$$

$$d_j \leqslant P(\mathcal{J} \setminus \mathcal{A}_i) \tag{10.26}$$

W pracy [224] zebrano i wykazano szereg ogólnych kryteriów, które po przyjęciu szczególnych postaci funkcji $f_j(t)$ dostarczają konkretnych warunków w kryteriach. Jeśli co najmniej jeden z poniższych warunków jest spełniony to istnieje rozwiązanie optymalne, w którym $i \to j$.

$$p_i \leqslant p_j, \text{ oraz}$$

$$(f_i(t) - f_j(t)) \text{ jest niemalejąca w } [P(\mathcal{B}_j), P(\mathcal{J} \setminus \mathcal{A}_i)], \tag{10.27}$$

$$f_j(P(\mathcal{B}_j) + p_j) = f_j(P(\mathcal{J} \setminus \mathcal{A}_i) - p_j), \text{ oraz}$$

$$(f_i(t) - f_j(t)) \text{ jest niemalejąca w } [P(\mathcal{J} \setminus \mathcal{A}_i) - p_j, P(\mathcal{J} \setminus \mathcal{A}_i)], \tag{10.28}$$

$$f_j(P(\mathcal{B}_j) + p_j) = f_j(P(\mathcal{A}_i)), \tag{10.29}$$

$$f_j(p_j) = f_j(P(\mathcal{J})), \tag{10.30}$$

Warunki (10.24)–(10.26) są szczególnymi przypadkami (10.27) –(10.30), odpowiednio. Możliwe są dalsze rozszerzenia dla problemu z $r_j \geqslant 0$, chociaż w tym przypadku warunki eliminacji zachodzą stosunkowo rzadziej. Zauważmy, że wynik badania kryterium eliminacji dla pewnej pary zadań i, j zależy od historii i kolejności badania wcześniejszych par zadań, co powoduje nietrywialność problemu implementacji tych kryteriów.

Dolne ograniczenie

Wyznaczenie wartości dolnego ograniczenia w przypadku kryterium addytywnego jest problemem ogólnie dość trudnym, powodujac powszechnie znaną słabość schematów B&B. Dolne ograniczenie jest sumą dwóch niezależnych składników: funkcji celu pochodzacej od rozwiązania częściowego σ oraz dolnego oszacowania wartości funkcji celu dla zbioru zadań nieuszeregowanych $\mathcal{U}$. Pierwszy składnik jest zwykle dokładny z wyjątkiem nielicznych przypadków szczególnych. Przykładowo, jeśli $r_j \geqslant 0$ zaś σ jest sekwencją końcową, to ze względu na nieznany horyzont uszeregowania, możliwe jest uzyskanie tylko oszacowania tego składnika. Jedną z możliwych technik pozyskiwania dolnych ograniczeń dla stosunkowo dużej klasy zagadnień jest relaksacja dualna, dyskutowana szczegółowo w następnym rozdziale. Inne dolne ograniczenia są projektowane indywidualnie, dla poszczególnych dla poszczególnych podklas problemów. I tak, wprowadzając relaksację dopuszczającą przerywanie zadań możemy sprowadzić problem do *zagadnieia transportowego*. W tym celu każde zadanie $j \in \mathcal{U}$ dzielimy na p_i/q zadań, gdzie q jest największym wspólnym dzielnikiem liczb p_j, $j \in \mathcal{U}$. Następnie dokonujemy przydziału poszczególnych zadań do $P(\mathcal{U})/q$ jednakowych przedziałów czasowych w przedziale $[P(\mathcal{S}), P(\mathcal{J})]$. Definicja funkcji kosztu tak otrzymanych podzielonych zadań wymaga pewnej staranności i dla przypadku kryterium będącego kombinacją liniową kryteriów $\sum w_i T_i$ oraz $\sum w_i C_i$ została podana w pracy [224]. W pracy [224] podano inne podejście do budowy dolnego ograniczenia oparte na *liniowym zagadnieniu przydziału* (assignment problem, AP), posiadajacego algorytm o złożoności $O(n^3)$.

10.5 Podejście dualne

Możliwe są rożne formy podejścia dualnego w zależności od postaci modelu pierwotnego. Najcześciej spotykany jest wariant oparty na dyskretyzacji czasu [93].

Podstawowa wersja algorytmu jest formułowana dla przypadku $r_j = 0$, $j = 1, \ldots, n$ oraz $\mathcal{R} = \emptyset$. Rozszerzenia zostaną omówione w dalszej cześci rozdziału. Podejście to wymaga założenia, że p_j są całkowite, istotnego dla przyjętej metody rozwiązania. Założenie to jest zazwyczaj spełnione w praktyce. W konsekwencji terminy rozpoczęcia i zakończenia czynności przyjmują wyłącznie wartości całowite ze zbioru $\{0, 1, \ldots, T\}$, gdzie T jest górnym ograniczeniem horyzontu czasowego dla harmonogramu. Założenie to jest równoważne niejawnemu przyjęciu, że czas jest dyskretny. Dalej wyprowadzimy warunek na ograniczoną (jednostkową) przepustowość maszyny. Dla danego $S = (S_1, \ldots, S_n)$ zdefinujemy zbiór zadań wykonywanych w przedziale $[t-1, t]$

$$\mathcal{I}_t(S) \stackrel{\text{def}}{=} \{j \in \mathcal{N} : \ t < S_j \leqslant t - p_j\} \tag{10.31}$$

oraz liczbę zadań wykonywanych w tym przedziale

$$g_t(S) \stackrel{\text{def}}{=} |\mathcal{I}_t(S|\,, \tag{10.32}$$

$t = 1, \ldots, T$.

10.5.1 Problem podstawowy.

Należy przyjąć $T = \sum_{j=1}^{n} p_j$, bowiem każdy harmonogram jest dosunięty w lewo na osi czasu zaś stanowisko wykonuje zadania bez przestoju.

Problem podstawowy można zapisać w formie zadania optymalizacji nieliniowej postaci

$$\min_{S} \sum_{j=1}^{n} f_j(S_j + p_j) \tag{10.33}$$

przy ograniczeniach

$$g_t(S) = 1, \ \ t = 1, 2, \ldots, T, \tag{10.34}$$

$$0 \leqslant S_j \leqslant T - p_j, \ \ j = 1, \ldots, n. \tag{10.35}$$

Przejście do problemu dualnego wymaga określenia, które z ograniczeń problemu pierwotnego będą uwzględnione bezpośrednio, zaś które poprzez ceny dualne. W rozważanym przypadku ograniczenie (10.35) używamy do wprowadzenia zbioru dopuszczalnych terminów rozpoczęcia zadań

$$\mathcal{S} \stackrel{\text{def}}{=} \{S = (S_1, \ldots, S_n) : \ 0 \leqslant S_j \leqslant T - p_j, \ j = 1, \ldots, n\}, \tag{10.36}$$

zaś dla (10.34) wprowadzamy zmienne dualne $u \stackrel{\text{def}}{=} (u_1, \ldots, u_T)$, otrzymując w końcowym efekcie funkcję Lagrange'a, przekształconą dalej do wygodniejszej postaci

$$L(S,u) \stackrel{\text{def}}{=} \sum_{j=1}^{n} f_j(S_j+p_j) + \sum_{t=1}^{T} u_t(g_t(S)-1)$$

$$= \sum_{j=1}^{n}(f_j(S_j+p_j) + \sum_{t=S_j+1}^{S_j+pj} u_t) - \sum_{t=1}^{T} u_t = \sum_{j=1}^{n} L_j(S_j,u) - \sum_{t=1}^{T} u_t \qquad (10.37)$$

gdzie

$$L_j(S_j,u) \stackrel{\text{def}}{=} f_j(S_j+p_j) + \sum_{t=S_j+1}^{S_j+p_j} u_t = f_j(S_j) + U_{S_j+p_j} - U_{S_j} \qquad (10.38)$$

oraz

$$U_t = \sum_{s=1}^{t} u_s, \;\; t = 1, \ldots T. \qquad (10.39)$$

Zmienne dualne u mają dobrą interpretację praktyczną. Wartość u_t może być traktowana jako dodatkowy koszt użycia (dzierżawy) maszyny w przedziale czasu $[t-1,t]$, zaś $\sum_{t=S_j+1}^{S_j+p_j} u_t$ jest całkowitym takim kosztem dodawanym do kosztu wykonania zadania w przedziale czasu $[S_j, S_j+p_j]$. Zauważmy, że jeśli S^* jest rozwiązaniem optymalnym problemu (10.33)–(10.35), to dla dowolnego u zachodzi

$$\sum_{j=1}^{n} f_j(S_j^*+p_j) = \sum_{j=1}^{n} f_j(S_j^*+p_j) + \sum_{t=1}^{T} u_t(g_t(S^*)-1) \geqslant \min_{S\in\mathcal{S}} L(S,u) \stackrel{\text{def}}{=} W(u), \qquad (10.40)$$

czyli $W(u)$ dostarcza dolnego ograniczenia wartości funkcji celu (10.33). Dodatkowo, jeśli dla pewnego u oraz S' takiego, że $L(S',u) = \min_{S\in\mathcal{S}} L(S,u)$ zachodzi $g_t(S') = 1, t = 1, \ldots, T$ (tj. S' jest dopuszczalne w sensie ograniczeń (10.34)–(10.35)), to S' jest rozwiązaniem optymalnym problemu (10.33)–(10.35). Zatem odpowiednio, problem dualny rozwiązujemy poprzez ekstremalizację

$$\max_{u} W(u). \qquad (10.41)$$

Problemu (10.41) można rozwiązać stosując dwupoziomową dekompozycję. Uwzględniając (10.36)–(10.38), otrzymujemy dla ustalonego u,

$$W(u) = \min_{S\in\mathcal{S}} L(S,u) = \sum_{j=1}^{n} \min_{0\leqslant S_j\leqslant T-p_j} L_j(S_j,u) - \sum_{t=1}^{T} u_t = \sum_{j=1}^{n} V_j(u) - \sum_{t=1}^{T} u_t \qquad (10.42)$$

gdzie

$$V_j(u) \stackrel{\text{def}}{=} \min_{0 \leqslant S_j \leqslant T-p_j} L_j(S_j, u) = \min_{S_j=0,1,\ldots,T-p_j} L_j(S_j, u) \tag{10.43}$$

Każdy problem (10.43) (dla ustalonego u) możemy rozwiązać poprzez bezpośredni przegląd całkowitoliczbowych wartości terminu rozpoczęcia. Ponieważ wzór (10.39) może być zapisany w formie rekurencyjnej $U_t = U_{t-1} + u_t$, $t = 1, \ldots, T$, $U_0 = 0$, obliczenie wartości U_t można zrealizować w czasie rzędu $O(T)$. Zatem, na mocy wzoru (10.38), rozwiązanie (10.43) wymaga czasu $O(T)$, zaś wyznaczenie $W(u)$ ze wzoru (10.42) wymaga czasu rzędu $O(nT)$.

Problem (10.41) jest zadaniem optymalizacji ciągłej, nieliniowej (dokładniej odcinkami liniowej), słabo wypukłej, jednakże ogólnie nieróżniczkowalnej. W celu potwierdzenia podanych własności należy zauważyć, że funkcja $W(u)$ może być przedstawiona w postaci

$$min\{c_k + u \times e_k, k = 1, \ldots, K\}, \tag{10.44}$$

dla pewnych c_k, $e_k = (e_{k1}, \ldots, e_{kt})$ oraz K, gdzie $\times$ jest operatorem iloczynu skalarnego. Istotnie, niech $S = (S_1, \ldots, S_n)$ będzie wektorem terminów rozpoczęcia zadań dla problemu (10.33)–(10.35) przy założeniu całkowitoliczbowych wartości S_j. Liczba wszystkich możliwych S (łącznie z niedopuszczalnymi) jest równa $(T)^n$. Ponumerujmy wszystkie takie wektory należące do $\mathcal{S}$ kolejnymi liczbami naturalnymi $k = 1, \ldots, K$, ustalając tym samym wzajemnie jednoznaczne odwzorowanie pomiędzy $S \in \mathcal{S}$ a jego indeksem k. Oczywiście K jest rzędu $O(T^n)$. Dla każdego S oraz skojarzonego z nim indeksu k kładziemy $c_k \stackrel{\text{def}}{=} \sum_{j=1}^{n} f_j(S_j)$ oraz $e_{kt} \stackrel{\text{def}}{=} g_t(S) - 1$, $t = 1, \ldots, T$. łatwo sprawdzić, że wzór (10.44) opisuje funkcję odcinkami liniową, słabo wypukłą. Formalnie, problem (10.44) można sprowadzić do zadania programowania liniowego (PL) jednakże o tak znacznym rozmiarze; dla praktycznego przykładu gdzie $T = 1000$ oraz $n = 100$ rozmiar zadania PL jest 10^3 zmiennych oraz rząd 10^{300} warunków ograniczających, że nawet metoda generacji kolumn nie może być polecana w tym przypadku.

Do rozwiązania problemu (10.41) zaleca się metodę subgradientu [93] która generuje ciąg rozwiązań $u^0, u^1, u^2, \ldots$ według zależności

$$u^{s+1} = P_U(u^s - \alpha^s W'(u^s)),\ s = 0, 1, \ldots, \tag{10.45}$$

gdzie $W'(u^s)$ jest subgradientem funkcji $W(u)$ w punkcie u^s, P_U operatorem projekcji na zbiór rozwiązań dopuszczalnych problemu, zaś α^s jest długościa kroku. Odpowiedni dobór ciągu α^s gwarantuje teoretyczną zbieżność procedury do rozwiązania optymalnego $W(u^*)$. Dla problemu (10.42)

otrzymujemy odpowiednie wzory szczególne: $u_t^0 = 0$, $t = 1, \ldots, T$ oraz

$$u_t^{s+1} = u_t^s - \alpha^s(g_t(S^s) - 1), \quad t = 1, \ldots, T. \tag{10.46}$$

Dobór ciągu α^s ma znaczący wpływ na własności metody w tym stabilność, zbieżność raz szybkość zbieżności. Zbieżność zapewnia już ciąg spełniajacy warunki $\lim_{s\to\infty} \alpha^s = 0$, $\sum_{s=0}^{\infty} \alpha^s = \infty$, czyli na przykład ciąg harmoniczny $\alpha^s = c/s$ dla pewnej stałej c. W przypadku znajomości wartości optymalnej $W(u^*)$ można użyć ciągu

$$\alpha^s = \lambda^s \frac{W(u^s) - W(u^*)}{\| g(S^s) - e \|^2}, \tag{10.47}$$

gdzie $0 < \lambda^s \leqslant 2$, S^s jest rozwiązaniem problemu (10.42) dla zmiennych dualnych u^s, e jest wektorem jednostkowym, $\| \cdot \|$ jest pewną normą, dla którego także wykazano zbieżność. Ponieważ wartość $W(u^*)$ jest a priori nieznana, w praktyce zastępowana jest górnym oszacowaniem UB^s modyfikowanym w kolejnych krokach procedury mimio, iż postępowanie takie nie posiada teoretycznych gwarancji zbieżności. Wtedy przyjmujemy $UB^{s+1} = \min\{UB^s, H^s\}$, gdzie $UB^0 = \infty$ zaś H^s jest wartością funkcji celu (10.33) dla pewnego rozwiązania dopuszczalnego (w sensie warunków (10.34)–(10.35)) otrzymanego pomocniczym algorytmem przybliżonym w s-tej iteracji. Przykładowo, w s-tej iteracji generujemy permutację π zgodną z uszeregowaniem zadań według niemalejących wartości terminów S_j^s; dla tej permutacji stosujemy wzór (10.3) dostarczający dopuszczalnych terminów rozpoczęcia zadań S, dla których następnie obliczamy wartość $H^s = \sum_{j\in\mathcal{N}} f_j(S_j)$. Przyjmując normę euklidesową w (10.47) mamy

$$\| g(S^s) - e \|^2 = \sum_{t=1}^{T} (g_t(S^s) - 1) \tag{10.48}$$

Ciąg (10.47) posiada pewną swobodę dotyczącą wyboru λ^s oraz swobodę dotyczącą wyznaczania H^s. Wielu autorów przyjmuje $\lambda^0 = 2.0$ oraz zmniejsza tą wartość dwukrotnie jeśli w kolejnych k iteracjach ($k \approx 5$) nie zaobserwowano wzrostu wartości $W(u^k)$. Takie postępowanie, podobnie jak poprzednie, powoduje dla niektórych przykładów słabą zbieżność $W(u^k)$ do UB^k, Rys. 10.1. Eksperymenty komputerowe wskazują, że schemat zmian λ^s powinien być dobierany indywidualnie dla konkretnych przykładów problemu, a nalepiej – adaptacyjny. Co więcej, bardziej wyrafinowane metody wyznaczania H^s poprawiających UB^s mają zdecydowanie mniejszy wpływ

Rysunek 10.1: Zbieżność UB^s oraz $W(u^s)$ dla α^s wg .

na szybkość zbieżności niż ciąg λ^s. Przypuszczalnie stagnacja procesu poszukiwań obserwowana na Rys. 10.1 jest powodowana zbyt szybką zbieżnością ciągu α^s do zera, lub też wystąpieniem zjawiska zygzakowania charakterystycznego dla metody subgradientowej. Tym kłopotom można zaradzić wprowadzając modyfikacje schematu zmian λ^s. W pierwszym przypadku proponuje się nieznacznie zwiększć λ^s jeśli nie zaobserwowano wzrostu $W(u^s)$, np. $\lambda^{s+1} = \kappa\lambda^s$ jeśli $W(u^s) < W(u^{s-1})$, gdzie $\kappa > 1$. W drugim przypadku proponuje się wprowadzenie elementów losowości, np. przyjmując, że κ jest realizacją zmiennej losowej o wartości średniej $\overline{\kappa} \geqslant 1$. Otrzymane wyniki sa zwykle zdecydowanie lepsze niż w schemacie "klasycznym", porównaj Rys. 10.2 oraz 10.3 otrzymane dla tego samego problemu i przykładu.

10.5.2 Problem rozszerzony.

Uogólnienie podanego podejścia jest możliwe, przy pewnym jego skomplikowaniu, w co najmniej dwóch odmiennych kierunkach – w celu uwzględnienia różnych terminów gotowości oraz dla $\mathcal{R} \neq \emptyset$. Rozpatrzmy wpierw pierwszy przypadek. Model (10.33)–(10.35) trzeba wówczas zastąpić przez

$$\min_{S} \sum_{j=1}^{n} f_j(S_j + p_j) \tag{10.49}$$

Rysunek 10.2: Zbieżność UB^s oraz $W(u^s)$ dla ulosowionego α^s.

Rysunek 10.3: Zbieżność UB^s oraz $W(u^s)$ dla ulosowionego α^s.

przy ograniczeniach

$$g_t(S) \leqslant 1, \;\; t = 1, 2, \ldots, T, \tag{10.50}$$

$$r_j \leqslant S_j \leqslant T - p_j, \;\; j = 1, \ldots, n. \tag{10.51}$$

Wartość T z poprzedniego rozdziału przyjmuje interpretację górnego ograniczenie horyzontu czasowego dla harmonogramu oraz wymaga ponownego poprawnego określenia. W rozważanym przypadku należałoby przyjąć raczej np. $T = \max_{j\in\mathcal{J}} r_j + \sum_{j\in\mathcal{J}} p_j$. Pamiętając, że wartość ta ma bezpośredni wpływ na złożoność obliczeniową odpowiednich algorytmów należy dążyć do jak najbardziej precyzyjnego oszacowania jej wartości. Zauważmy dalej, że warunek (10.34) zmienił istotnie swój charakter, porónaj z (10.50), bowiem dopuszcza bezczynność maszyny w pewnych przedziałach czasu.

Analogicznie jak poprzednio, jeśli S^* jest rozwiązaniem optymalnym problemu (10.33)–(10.35), to dla dowolnego $u \geqslant 0$ zachodzi

$$\sum_{j=1}^{n} f_j(S_j^*) \geqslant \sum_{j=1}^{n} f_j(S_j^*) + \sum_{t=1}^{T} u_t(g_t(S^*) - 1) \geqslant \min_{S\in\mathcal{S}} L(S, u) \stackrel{\text{def}}{=} W(u), \tag{10.52}$$

czyli $W(u)$ dostarcza dolnego ograniczenia wartości funkcji celu (10.33), jednakże tylko dla $u \geqslant 0$. Zmienia to istotnie odpowiedni rezultat z poprzedniego rozdziału. Dodatkowo, jeśli dla pewnego u oraz S' takiego, że $L(S', u) = \min_{S\in\mathcal{S}} L(S, u)$ zachodzi $g_t(S') \leqslant 1$, $t = 1, \ldots, T$ (tj. S' jest dopuszczalne w sensie ograniczeń (10.50)–(10.51)) oraz równoczesnie jest spełnione $\sum_{t=1}^{T} u_t(g_t(S') - 1) = 0$, to S' jest rozwiązaniem optymalnym problemu (10.49)–(10.51). Jak widać, odpowiednie warunki optymalności uległy pewnej komplikacji. Problem dualny rozwiązujemy poprzez ekstremalizację

$$\max_{u\geqslant 0} W(u). \tag{10.53}$$

Podobnie jak poprzednio, problem (10.53) rozwiązujemy poprzez dwupoziomową dekompozycję, przy czym

$$\mathcal{S} \stackrel{\text{def}}{=} \{S = (S_1, \ldots, S_n) : \; r_j \leqslant S_j \leqslant T - p_j, \; j = 1, \ldots, n\}, \tag{10.54}$$

$$W(u) = \sum_{j=1}^{n} \min_{r_j \leqslant S_j \leqslant T-p_j} L_j(S_j, u) - \sum_{t=1}^{T} u_t = \sum_{j=1}^{n} V_j(u) - \sum_{t=1}^{T} u_t \tag{10.55}$$

gdzie

$$V_j(u) \stackrel{\text{def}}{=} \min_{r_j \leqslant S_j \leqslant T-p_j} L_j(S_j, u) = \min_{S_j = r_j, r_j+1, \ldots, T-p_j} L_j(S_j, u) \tag{10.56}$$

Rysunek 10.4: Zbieżność UB^s oraz $W(u^s)$ dla ulosowionego α^s.

Odpowiedni cechy numeryczne algorytmu dolnego poziomu pozostają w mocy. Ponieważ problem (10.55) jest zadaniem optymalizacji z ograniczeniami $u \in \mathcal{U} = \{u : \ u \geqslant 0\}$, zgodnie z wzorem (10.45), po wprowadzaniu projekcji na zbiór rozwiązań dopuszczalnych otrzymujemy

$$u_t^{s+1} = \max\{0, u_t^s + \alpha^s(g_t(S^s) - 1)\}, \ \ t = 1, \ldots, T. \tag{10.57}$$

gdzie α^s jest ciągiem opisanym jak poprzednio. Ze względu na postać problemu (10.55) możliwe jest alternatywne zastosowanie techniki *warunkowego subgradientu* [217] charakteryzującego się korzstniejszą zbieżnością od metod ogólnych z Rozdz. 8.3.6. Jednakże, badania numeryczne pokazują, że wprowadzenie ulosowienia zmiany α^s jest zdecydowanie najkorzystniejszym rozwiązaniem dla uzyskania szybkiej zbieżności procedury.

10.5.3 Dalsze rozszerzenia

Uogólnienie problemu (10.49)–(10.51) na przypadek $\mathcal{R} \neq \emptyset$ wymaga dodania do (10.49)–(10.51) warunku

$$S_i + p_i - S_j \leqslant 0, \ (i,j) \in \mathcal{R}. \tag{10.58}$$

W dalszym ciągu powstaje dylemat czy ograniczenie to uwzględnić w definicji zbioru $\mathcal{S}$, czy poprzez ceny dualne. Ogólnie, wprowadzenie (10.58) do

definicji zbioru $\mathcal{S}$ powoduje zależność składowych wektora S oraz niemożność wykonania dekompozycji na dolnym poziomie na niezależnych podzadań postaci $\min_{r_j \leqslant S_j \leqslant T-p_j} L_j(S_j, u, v)$. Niemniej istnieje możliwość efektywnego rozwiązania zadania dolnego poziomu dla szczególnych postaci $\mathcal{R}$, co zostanie wykorzystane dalej. Załóżmy więc, że $\mathcal{R}$ została zdekomponowana (arbitralnie) na dwa podzbiory $\mathcal{T}$ oraz $\mathcal{Q} \stackrel{\text{def}}{=} \mathcal{R} \setminus \mathcal{T}$ takie, że graf $(\mathcal{J}, \mathcal{T})$ jest drzewem lub lasem niezakorzenionym, tzn. dla każdego j zachodzi $|\mathcal{A}_j(\mathcal{T})| \leqslant 1$. Odpowiednio ograniczenie (10.58) zostaje zdekomponowane na dwa ograniczenia odpowiadające $\mathcal{T}$ oraz $\mathcal{Q}$, przy czym pierwsze z nich będzie uwzględnione poprzez zbiór $\mathcal{S}$ zaś drugie poprzez ceny dualne. W dalszym ciągu definiujemy

$$\mathcal{S} \stackrel{\text{def}}{=} \{S:\ r_j \leqslant S_j \leqslant T - p_j,\ j = 1, \ldots, n,\ S_i + p_i \leqslant S_j,\ (i,j) \in \mathcal{T}\} \quad (10.59)$$

Wprowadzając zmienne dualne u jak poprzednio oraz zmienne dualne v_{ij}, $(i,j) \in \mathcal{Q}$ skojarzone z każdym ograniczeniem (10.58) dla $\mathcal{Q}$ otrzymujemy funkcję Lagrange'a, przekształconą następnie do wygodniejszej postaci

$$L(S,u,v) = \sum_{j=1}^{n} f_j(S_j) + \sum_{t=1}^{T} u_t(g_t(S) - 1) + \sum_{(i,j)\in\mathcal{Q}} v_{ij}(S_i + p_i - S_j)$$

$$= \sum_{j=1}^{n}(f_j[S_j) + \sum_{t=S_j+1}^{S_j+p_j} u_t + S_j(\sum_{i\in\mathcal{A}_j(\mathcal{Q})} v_{ji} - \sum_{i\in\mathcal{B}_j(\mathcal{Q})} v_{ij})] - \sum_{t=1}^{T} u_t + \sum_{(i,j)\in\mathcal{Q}''} v_{ij}p_i$$

$$= \sum_{j=1}^{n} L_j(S_j,u,v) - \sum_{t=1}^{T} u_t + \sum_{(i,j)\in\mathcal{Q}} v_{ij}p_i \quad (10.60)$$

gdzie

$$L_j(S_j,u,v) \stackrel{\text{def}}{=} f_j(S_j) + \sum_{t=S_j+1}^{S_j+p_j} u_t + S_j(\sum_{i\in\mathcal{A}_j(\mathcal{Q})} v_{ji} - \sum_{i\in\mathcal{B}_j(\mathcal{Q})} v_{ij}). \quad (10.61)$$

Przez oczywistą analogię wartość

$$W(u,v) \stackrel{\text{def}}{=} \min_{S\in\mathcal{S}} \sum_{j=1}^{n} L_j(S_j,u,v) - \sum_{t=1}^{T} u_t + \sum_{(i,j)\in\mathcal{Q}} v_{ij}p_i \quad (10.62)$$

stanowi dolne ograniczenie optymalnej wartości funkcji celu dla dowolnych $u \geqslant 0$, $v \geqslant 0$. Odpowiedni problem dualny ma postać

$$\max_{u\geqslant 0} \max_{v\geqslant 0} W(u,v). \quad (10.63)$$

Podobnie jak poprzednio, jeśli dla S' wyznaczonego z rozwiązania problemu $\min_{S\in\mathcal{S}} L(S,u,v)$ dla pewnych $u \geqslant 0$, $v \geqslant 0$ spełnione są ograniczenia (10.50), (10.58) dla zbioru $\mathcal{Q}$ oraz dodatkowo zachodzą warunki $\sum_{t=1}^{T} u_t(g_t(S')-1)=0$, $\sum_{(i,j)\in\mathcal{Q}} v_{ij}(S_i+p_i-S_j)=0$, to otrzymane rozwiązanie S' jest optymalne.

W dalszym ciągu potrzebny jest tylko metoda rozwiązania problemu dolnego poziomu. Przyjmijmy, że numeracja zadań jest zgodna z relacją $\mathcal{T}$, tzn. dla każdego j mamy $\mathcal{B}_j(\mathcal{T}) \subset \{1,\ldots,j-1\}$. Niech $Z_j(t)$ oznacza minimalną wartość funkcji celu (10.49) przy uszeregowaniu zadań ze zbioru $\mathcal{B}_j(\mathcal{T})\cup\{j\}$ przy założeniu, zę kończą się one przed momentem czasowym t. Zachodzi następująca zależność rekurencyjna wynikająca z zasady programowania dynamicznego

$$Z_j(t) = \begin{cases} \infty & 0 \leqslant t < a_j \\ \min\{Z_j(t-1), L_j(t,u,v) + \sum_{i\in\mathcal{B}(\mathcal{Q})} Z_i(t-p_j)\} & a_j \leqslant t \leqslant b_j \\ Z_j(t-1) & b_j < t \leqslant T \end{cases} \tag{10.64}$$

Wartości a_j i b_j są odpowiednio dolnym i górnym ograniczeniem terminu zakończenia zadania j, $j\in\mathcal{J}$. Dla ich wyznaczenia można posłużyć się grafem skierowanym $(\mathcal{J},\mathcal{R})$ w którym obciążenie wierzchołka j jest równe p_j, $j\in\mathcal{N}$. Wartość a_j jest długością najdłuższej drogi w tym grafie dochodzącej do wierzchołka j (łącznie z obciążeniem tego wierzchołka). Odpowiednio, wartość b_j jest równa T minus długość najdłuższej drogi wychodzącej z wierzchołka j (bez obciążenia tego wierzchołka). Wyznaczenie wszystkich wartości a_j, b_j jest możliwe w czasie $O(|\mathcal{R}|+n)$. Odpowiedni wzór do rozwiązania zadania górnego poziomu ma postać

$$u_t^{s+1} = u_t^s + \alpha^s(g_t(C^s)-1),\ t=1,\ldots,T, \tag{10.65}$$

$$v_{ij}^{s+1} = \max\{0, v_{ij}^s + \alpha^s(C_i^s - C_j^s + p_j)\},\ (i,j)\in\mathcal{Q}, \tag{10.66}$$

gdzie

$$\alpha^s = \lambda^s \frac{UB^s - V(u^s,v^s)}{\sqrt{\sum_{t=1}^{T}(g_t(C^s)-1)^2 + \sum_{(i,j)\in\mathcal{Q}}(C_i^s - C_j^s + p_j)^2}}. \tag{10.67}$$

W wielu przypadkach relacja $\mathcal{R}$ jest dana arbitralnie, może też ona pochodzić z kryteriów eliminacji lub reguły podziału w schemacie B&B.

10.6 Alternatywne podejście dualne

Odmienne wyniki można otrzymać wychodząc od modelu (10.10)–(10.15). Zajmijmy się wpierw przypadkiem $\mathcal{R}=\emptyset$. Odpowiednio dla tego sformu-

łowania, ograniczenia (10.13)–(10.15) uwzględniamy wprowadzając zbiory dopuszczalnych wartości dla zmiennych S oraz y

$$\mathcal{S} \stackrel{\text{def}}{=} \{S:\ r_j \leqslant S_j \leqslant T - p_j,\ j = 1, \ldots, n\} \tag{10.68}$$

$$\mathcal{Y} \stackrel{\text{def}}{=} \{y_{ij}:\ \ y_{ij} \in \{0, 1\},\ \ j = i+1, \ldots, n,\ \ i = 1, \ldots, n\} \tag{10.69}$$

gdzie T jest górnym ograniczeniem horyzontu planowania; można na przykład przyjąć $T = \max_{j\in\mathcal{J}} r_j + \sum_{j\in\mathcal{J}} p_j$. Dalej wprowadzając zmienne dualne u_{ij} dla każdego ograniczenia (10.11) oraz zmienne v_{ij} dla każdego ograniczenia (10.12) otrzymujemy funkcję Lagrange'a postaci

$$\begin{aligned}
L(S,y,u,v) &= \sum_{j=1}^{n} f_j(S_j+p_j) + \sum_{i=1}^{n}\sum_{j=i+1}^{n} u_{ij}(S_i + p_i - S_j - K(1-y_{ij})) \\
&+ \sum_{i=1}^{n}\sum_{j=i+1}^{n} v_{ij}(S_j + p_j - S_i - Ky_{ij}) = \sum_{j=1}^{n} f_j(S_j+p_j) + \sum_{i=1}^{n} S_i\Big(\sum_{j=i+1}^{n} u_{ij}\Big) \\
&+ \sum_{i=1}^{n} p_i\Big(\sum_{j=i+1}^{n} u_{ij}\Big) - \sum_{i=1}^{n}\sum_{j=i+1}^{n} u_{ij}S_j - K\sum_{i=1}^{n}\sum_{j=i+1}^{n} u_{ij} + K\sum_{i=1}^{n}\sum_{j=i+1}^{n} u_{ij}y_{ij} \\
&+ \sum_{i=1}^{n}\sum_{j=i+1}^{n} v_{ij}S_j + \sum_{i=1}^{n}\sum_{j=i+1}^{n} v_{ij}p_j - \sum_{i=1}^{n} S_i\Big(\sum_{j=i+1}^{n} v_{ij}\Big) - K\sum_{i=1}^{n}\sum_{j=i+1}^{n} v_{ij}y_{ij} \\
&= \sum_{j=1}^{n} f_j(S_j+p_j) + \sum_{j=1}^{n} S_j\Big(\sum_{i=j+1}^{n} u_{ji}\Big) - \sum_{j=1}^{n} S_j\Big(\sum_{i=1}^{j-1} u_{ij}\Big) + \sum_{j=1}^{n} S_j\Big(\sum_{i=}^{j-1} v_{ij}\Big) \\
&\quad - \sum_{j=1}^{n} S_j\Big(\sum_{i=j+1}^{n} v_{ji}\Big) + K\sum_{i=1}^{n}\sum_{j=i+1}^{n} u_{ij}y_{ij} - K\sum_{i=1}^{n}\sum_{j=i+1}^{n} v_{ij}y_{ij} \\
&\quad + \sum_{j=1}^{n} p_j\Big(\sum_{i=1}^{j-1} v_{ij}\Big) + \sum_{j=1}^{n} p_j\Big(\sum_{i=j+1}^{n} u_{ji}\Big) - K\sum_{i=1}^{n}\sum_{j=i+1}^{n} u_{ij} \\
&= \sum_{j=1}^{n}\Big[f_j(S_j+p_j) + S_j\Big(\sum_{i=j+1}^{n} u_{ji} - \sum_{i=1}^{j-1} u_{ij} + \sum_{i=1}^{j-1} v_{ij} - \sum_{i=j+1}^{n} v_{ji}\Big)\Big] \\
&+ K\sum_{i=1}^{n}\sum_{j=i+1}^{n}(u_{ij}-v_{ij})y_{ij} + \sum_{j=1}^{n} p_j\Big(\sum_{i=1}^{j-1} v_{ij} + \sum_{i=j+1}^{n} u_{ji}\Big) - K\sum_{i=1}^{n}\sum_{j=i+1}^{n} u_{ij}
\end{aligned} \tag{10.70}$$

Oznaczmy przez

$$W(u,v) = \min_{S\in\mathcal{S}} \min_{y\in\mathcal{Y}} L(S,y,u,v). \tag{10.71}$$

Podobnie jak poprzednio $W(u,v)$ jest dolnym ograniczeniem optymalnej wartości funkcji celu problemu (**??**)–(**??**) dla dowolnych $u, v \geqslant 0$. W celu rozwiązania problemu (10.71) należy zauważyć, że (10.70) jest sumą składników, z których każdy jest zależny od niektórych zmiennych, mianowicie

$$L(S,y,u,v) = \sum_{j=1}^{n} L_j(S_j,u,v) + K\sum_{i=1}^{n}\sum_{j=i+1}^{n} Q_{ij}(y_{ij},u,v) + V(u,v) \quad (10.72)$$

gdzie

$$L_j(S_j,u,v) = f_j(S_j+p_j)+\alpha_j S_j,\ \alpha_j = \sum_{j=i+1}^{n}(u_{ij}-v_{ji}) - \sum_{i=1}^{j-1}(u_{ij}-v_{ij}) \quad (10.73)$$

$$Q_{ij}(y_{ij},u,v) = (u_{ij}-v_{ij})y_{ij} \quad (10.74)$$

$$V(u,v) = \sum_{j=1}^{n} p_j\Big(\sum_{i=1}^{j-1} v_{ij} + \sum_{i=j+1}^{n} u_{ji}\Big) - K\sum_{i=1}^{n}\sum_{j=i+1}^{n} u_{ij}. \quad (10.75)$$

Stąd możemy dokonać dekompozycji zadania minimalizacji (10.71)

$$W(u,v) = \sum_{j=1}^{n} \min_{r_j \leqslant S_j \leqslant T-p_j} L_j(S_j,u,v)$$

$$+\sum_{i=1}^{n}\sum_{j=i+1}^{n} \min_{y_{ij}\in\{0,1\}} Q_{ij}(y_{ij},u,v) + V(u,v). \quad (10.76)$$

Rozwiązania optymalne podproblemów wchodzących w skład (10.76) można otrzymać stosunkowo łatwo. Istotnie, jeśli $u_{ij} \geqslant v_{ij}$ to $Q_{ij}(y_{ij},u,v)$ osiąga minimum dla $y^*_{ij} = 0$; jeśli $u_{ij} < v_{ij}$ to $y^*_{ij} = 1$. Funkcja $L_j(S_j,u,v)$ jest funkcją jednej zmiennej S_j, jej minimum w przedziale $[r_j, T-p_j]$ można wyznaczyć numerycznie (dla ogólnej postaci $f_j(t)$) lub parametrycznie (dla szczególnych postaci funkcji $f_j(t)$, np. liniowa lub afiniczna). Zauważmy, że jeśli $\alpha_j \geqslant 0$ to $S^*_j = r_j$ dla dowolnej niemalejącej funkcji $f_j(t)$. W przeciwnym przypadku ekstremum S^*_j wypada w jednym z punktów r_j, $T-p_j$ lub w pewnym punkcie wewnątrz przedziału $[r_j, T-p_j]$, w zależności od wzajemnej relacji pomiędzy $f_j(t)$ oraz α_j. Ostatecznie wyznaczenie $W(u,v)$ dla ustalonego u oraz v wymaga czasu rzędu $O(n^2)$ zakładając, że minimum $L_j(S_j,u,v)$ wyznaczymy w czasie $O(1)$. Maksymalizację $\max_{u,v\geqslant 0} W(u,v)$ można wykonać posługując się metodą subgradientową, analogicznie jak w rozdziałach poprzednich.

Biorąc pod uwagę, że dla ustalonego problemu szeregowania można sformułować kilka różnych modeli matematycznych, każdy taki problem implikuje jeden lub kilka alternatywnych problemów dualnych. Postać wynikowych problemów dualnych zależy od praktycznych możliwości ich rozwiązania, a mianowicie ich złożoności obliczeniowej i szybkości zbiegania do rozwiązania optymalnego. Decyzja, który z problemów dualnych jest najkorzystniejszy dla praktycznych zastosowań, zależy głównie od wyników numerycznych osiąganych w testach komputerowych. Przykładowo, problem z przezbrojeniami $1|setup|\sum f_i$, będący ougólnieniem problemu podstawowego $1||\sum f_i$, dość trudno modelować w schemacie z czasem dyskretnym, patrz Rozdz. 10.5, natomiast zupełnie naturalnie modeluje się go w zadaniu (**??**)-(**??**). W takich przypadkach wybór podejścia jest oczywisty.

10.7 Proste algorytmy przybliżone

Algorytmy przybliżone są głównie projektowane dla problemów z kryteriami $\sum T_j$, $\frac{1}{n}\sum T_j$, $\sum w_j T_j$, $\frac{1}{n}\sum w_j T_j$ najbardziej interesującymi dla praktyków, patrz na przykład przegląd w pracy [341]. Wśród algorytmów konstrukcyjnych pojawia się duża grupa metod przybliżających rozwiązanie optymalne rozwiązaniem dla problemu pokrewnego. I tak, metoda SPT, dedykowana dla problemu $1||\sum T_j$, porządkuje zadania wg niemalejących wartości p_j wykorzystując tym samym rozwiązanie optymalne problemu $1||\sum(C_j - d_j)$. Metoda WSPT dla problemu $1||\sum w_j T_j$ porządkuje zadania wg niemalejących wartości p_j/w_j wykorzystując optymalne rozwiązanie problemu $1||\sum w_j(C_j - d_j)$. Obie metody dostarczają rozwiązanie optymalne jeżeli wszystnie zadania są spóźnione, tzn. $\max_{j\in\mathcal{J}} d_j < \min_{j\in\mathcal{J}} p_j$. Metoda EDD porządkuje zadania wg niemalejących wartości d_j wykorzystując tym samym optymalne rozwiązanie problemu $1||T_{\max}$. Metoda EDD dostarcza rozwiązanie optymalne dla $1||\sum T_j$ jeśli co najwyżej jedno zadanie w otrzymanej sekwencji jest spóźnione. Metoda MWSPT (modyfikowana WSPT) porządkuje zadania wg nierosnących wartości $(p(\mathcal{J}) - d_j)\frac{w_j}{p_j}$. Metody SPT, WSPT, EDD, MWSPT posiadają złożoność obliczeniową $O(nlogn)$.

Drugą grupę metod stanowią dynamiczne reguły priorytetowe, które tradycyjnie operują na następujących pojęcach: zbiór $\mathcal{S} \subset \mathcal{J}$ elementów już uszeregowanych posiadających ustalony porządek wykonywania σ, zbiór zadań jeszcze nie uszeregowanych $\mathcal{U} \stackrel{\text{def}}{=} \mathcal{J} \setminus \mathcal{S}$ oraz zadanie $k \in \mathcal{U}$ wybrane do uszeregowania, które po uszeregowaniu tworzy permutacje częściową σk. Rozpoczynamy od $\mathcal{S} = \emptyset$. Algorytm MDD (modified due date), dedykowany dla problemu $1||\sum T_j$, jako kolejne do uszeregowania wybiera zadanie k dla

którego wartość $\max\{d_k, p(\mathcal{S}) + p_k\}$ jest najmniejsza. Reguła API (aparent priority index) jako kolejne wybiera zadanie k o największej wartości

$$A_k = \frac{w_k}{p_k}\, e^{-Z \max\{d_k - p(\mathcal{S}) - p_k, 0\}} \tag{10.77}$$

gdzie $Z = \omega \frac{|\mathcal{U}|}{p(\mathcal{U})}$ zaś $\omega \in [0.5, 2.0]$ jest pewnym parametrem. Parametr ω jest dobierany eksperymentalnie, przykładowo proponowane są wartości $\omega = 0.5$ dla TF=0.2, $\omega = 0.9$ dla TF=0.4, $\omega = 2.0$ dla TF¿0.4, gdzie TF jest współczynnikiem spóźnienia $TF = 1 - \sum_{j=1}^{n} d_j/(n \sum_{j=1}^{n} p_j)$, patrz także Rozdz. 10.9.

Bardziej skomplikowane reguły można spotkać w algorytmach WI oraz PSK dedykowanych dla problemu $1||\sum T_j$. Algorytmy te, przy założeniu, żezadania są indeksowane zgodnie z regułą SPT (lub EDD), sprawdzają pewne warunki logiczne pomiędzy parami zadań w zbiorze $\mathcal{U}$ w celu wyboru zadania k. Niestety, proces ten może być zapisany tylko w formie procedury, nie zaś pojedynczego wzoru. Reguły MDD, AP, PSK, WI dostarczają algorytmów o złożonści obliczeniowej $O(n^2)$.

Pewną klasę algorytmów przybliżonych można otrzymać poprzez *aproksymację* funkcji kosztu $f_j(t)$ funkcją $g_j(t)$ klasy (10.4). W konsekwencji w miejsce problemu $1||\sum f_j$ rowiązuje się problem $1||\sum g_j$, który dostarcza przybliżonej kolejności wykonywania zadań dla problemu $1||\sum f_j$. W wersji statycznej algorytmu przeprowadza się aproksymację wszystkich funkcji kosztu w przedziale $[0, p(\mathcal{J})]$ lub $[p_j, p(\mathcal{J})]$, a następnie rozwązuje problem $1||\sum g_j$. W wersji dynamicznej przeprowadza się aproksymację funkcji kosztu zadań ze zbioru $\mathcal{U}$ w przdziale $[p(\mathcal{S}), p(\mathcal{J})]$ lub $[p(\mathcal{S}) + p_j, p(\mathcal{J})]$, a następnie wybiera do uszeregowania zadanie o największej wartości w_j otrzymanej z aproksymacji. W każdym z wymienionych przypadków aproksymacja przeprowadzana może być w sensie jednej z typowym norm

$$l_\infty = \max_{a \leqslant t \leqslant b} \max_{1 \leqslant j \leqslant n} |f_j(t) - g_j(t)| \tag{10.78}$$

$$l_2 = \int_a^b \sum_{j=1}^{n} |f_j(t) - g_j(t)| \tag{10.79}$$

gdzie

$$g_j(t) = \int_{t-p_j}^{t} (\alpha_j \phi(u) + \beta_j) du \tag{10.80}$$

zaś $[a, b]$ jest przedziałem aproksymacji. Rozwiązanie problemu aproksymacji wymaga wyznaczenia α_j, β_j, $j \in \mathcal{J}$ oraz niemalejącej funkcji $\phi(u)$ minimalizujących odpowiednią normę i jest w ogólnym przypadkach dość kłopotliwe. Wynika to z faktu, że nie istnieją metody efektywnego uzyskiwania

aproksymacji jednostajnych (w sensie normy l_∞) wyjatkiem kilku szczególnych przypadków, zaś zwykle funkcje $f_j(t)$ są nieróżniczkowalne. Z tego też względu proponuje się rozważać pewne uproszczone problemy aproksymacji w jednej z następujących postaci:

- zakładając, że funkcja $\phi(u)$ jest znana z dokładnością do pewnych (nieznanych) parametrów, należy wyznaczyć α_j, β_j, $j \in \mathcal{J}$ oraz parametry tej funkcji,
- zakładając, że funkcja $\phi(u)$ jest znana, należy wyznaczyć α_j, β_j, $j \in \mathcal{J}$,
- zakładając, że funkcja $\phi(u)$ oraz β_j, $j \in \mathcal{J}$ są znane, należy wyznaczyć α_j, $j \in \mathcal{J}$.

Pewnym ułatwieniem rozwiazywaniu wymienionych problemów jest fakt, że dla algorytmu przybliżonego potrzebne są tylko watości α_j, $j \in \mathcal{J}$. Niekiedy możliwe jest rozwiązane problemu aproksymacji w przypadku ogólnym. Tak np. rozwiazując drugie wymienione zadanie dla normy l_2 otrzymujemy

$$\alpha_j = \frac{\int_a^b f_j(t)dt \int_a^b G_j(t)dt - (b-a)\int_a^b f_j(t)G_j(t)dt}{[\int_a^b G_j(t)dt]^2 - (b-a)\int_a^b G_j^2(t)dt} \tag{10.81}$$

gdzie

$$G_j(t) = \int_{t-p_j}^{t} \phi(u)du. \tag{10.82}$$

Poszczególne algorytmy można otrzymać rozwiązując konkretne problemy aproksymacji. Przykładowo przyjmując przedział aproksymacji $[0, p(\mathcal{J})]$ odpowiadający regule statycznej, otrzymamy funkcje priorytetu:

(A1) dla normy l_∞: $\alpha_j = \frac{w_j}{p_j}[1 - \frac{d_j}{p(\mathcal{N})}]$

(A2) dla normy l_2: $\alpha_j = \frac{w_j}{p_j}[1 - 3(\frac{d_j}{p(\mathcal{N})})^2 + 2(\frac{d_j}{p(\mathcal{N})})^3]$

Wynik (A1) jest zgodny z regułą MWSPT. Przyjmując przedził aproksymacji $[p(\mathcal{S}), p(\mathcal{J})]$ odpowiadający regule dynamicznej, otrzymamy funkcje priorytetu

(E1) dla normy l_∞: $\alpha_j = \begin{cases} \frac{w_j}{p_j}[1 - \frac{d_j - p(\mathcal{S})}{p(\mathcal{U})}] & d_j \geqslant p(\mathcal{S}) \\ \frac{w_j}{p_j} & d_j < p(\mathcal{S}) \end{cases}$

(E2) dla normy l_2: $\alpha_j = \begin{cases} \frac{w_j}{p_j}[1 - 3(\frac{d_j - p(\mathcal{S})}{p(\mathcal{U})})^2 + 2(\frac{d_j - p(\mathcal{S})}{p(\mathcal{U})})^3] & d_j \geqslant p(\mathcal{S}) \\ \frac{w_j}{p_j} & d_j < p(\mathcal{S}) \end{cases}$

Spośród wymienionych reguł najlepsze wyniki numeryczne osiąga się dla algorytmu AP. Generalnie, reguły priorytetowe dostarczają rozwiazań ze stosunkowo duzym błędem do 20-30%. Dlatego też, w wielu przypadkach, traktuje się je jako punkt wyjścia do algorytmów poszukiwań lokalnych

10.8 Algorytmy poszukiwań lokalnych

Brak "mocnych" szczególnych własności problemów z kryteriami addytywnymi uzasadnia w pełni celowość stosowania algorytmów typu LS. Powszechnie, jedną z najczęściej implementowanych technik jest poszukiwanie zstępujące DS, dużo bardziej skuteczne niż dla problemów z kryterium $f_{\max}$. Ponieważ każde pojedyncze rozwiązanie większości problemów $1|\beta|\sum f_i$ może być reprezentowane permutacją, lokalne otoczenia można stosunkowo łatwo uzyskać na przykład poprzez wymianę par elementów przyległych w permutacji (API), wymianę par elementów dowolnych (NPI), technikę przenoszenia i wstawiania (INS). Z bardziej zaawansowanych technik generacji sąsiedztwa stosowane są: odwrócenie podciagu (INV), permutacje przyległych k elementów (k-AI), permutacje k dowolnych elementów (k-NI). Liczebności tych otoczeń wynoszą odpowiednio: $O(n)$ (API, 2-AI), $O(n^2)$ (NPI, 2-NI, INS), $O(n^k)$ (k-NI), $O(nk!)$ (k-AI). Techniki API, NPI, K-AI i k-NI są częściej stosowane od INS dla problemów z kryterium addytywnym (odmiennie niż dla probemów z kryterium $f_{\max}$), choć INS potrafi być rónie skuteczne. Szereg wymienionych pomysłów pochodzi od metod przybliżonych rozwiązywania problemu komiwojażera (TSP). W każdym z podanych przypadków, koszt przeszukania pojedynczego otoczenia jest kluczowy dla szybkości działania algorytmu. Przykładowo, pełne otoczenie NPI (tożsame z 2-NI) zawiera $O(n^2)$ permutacji. Jeśli obliczenie pojedynczej wartości funkcji celu ma złożoność $O(n)$, to przeszukanie wyczerpujące tego otoczenia ma złożoność obliczeniową $O(n^3)$, dość kosztowną, bowiem w praktyce co najmniej $O(n)$ różnych otoczeń jest badanych. Z tego powodu poszukiwane są takie teoretyczne własności problemu, które umożliwiają znaczącą redukcję kosztu przeglądania otoczenia.

Jednym z oczywistych pomysłów jest wyznaczenie wartości funkcji celu dowolnego rozwiązania z otoczenia w czasie $O(1)$ poprzez zmodyfikowanie wartości funkcji celu rozwiązania generujacego otoczenie. Jest to na przykład możliwe dla problemu $1||\sum f_i$ i otoczenia API. Oznaczmy przez π permutację reprezentującą rozwiązanie o wartości $F(\pi) = \sum_{j=1}^{n} f_j(S_j + p_j)$, gdzie S_j są terminami rozpoczęcia zadań w permutacji, to znaczy $S_{\pi(j+1)} = S_{\pi(j)} + p_{\pi(j)}$, $j = 1, 2, \ldots, n-1$. Niech $\pi_{(a,b)}$, $b = a + 1$ będzie permuta-

cją otrzymaną przez zamianę pary zadań przyległych $(\pi(a), \pi(b))$. Wówczas zachodzi

$$F(\pi_{(a,b)}) = F(\pi) - f_{\pi(a)}(x - p_{\pi(b)}) - f_{\pi(b)}(x) + f_{\pi(b)}(x - p_{\pi(a)}) + f_{\pi(a)}(x) \tag{10.83}$$

gdzie $x = S_{\pi(b)} + p_{\pi(b)}$. Dla szczególnych postaci funkcji $f_j(t)$ można otrzymać nieco prostsze wzory szczególne. Rozszerzenie tego podejścia na schemat NPI pozostaje niekosztowne, jeśli tylko względna odległość wymieianych elementó $|b-a|$ pozostaje stała i niewielka. Dalej, korzystając z faktu, że każda pojedyncza modyfikacja INS moze być przedstawiona za pomocą ciągu wymian API, rezultat (10.83) może być także użyteczny dla otoczenia INS. Pozwala on zmniejszyć złożoność obliczeniową przeglądania całego otoczenia typu INS z $O(n^3)$ do $O(n^2)$. Zmniejszenie kosztu przeszukiwania otoczenia jest możliwe także poprzez ograniczanie wielkości otoczenia. Przykładowo, jeśli w schemacie NPI lub INS elementarna modyfikacja jest określona parą (a, b), $a, b \in \{1, 2, \ldots, n\}$, to ograniczenie wielkości otoczenia jest możliwe poprzez dopuszczenie tylko par (a, b) takich, że $|b - a| \leqslant \rho$, gdzie ρ jest stałą lub parametrem.

10.8.1 DS z niezależnymi wymianami

Ponieważ złożoność obliczeniowa badania otoczenia k-NI jest $O(n^{k+1})$, w praktyce najczęściej stosowane jest otoczenie 2-NI w celu redukcji kosztu obliczeń. W pracy [67] pokazano metodę umożliwiającą analizowanie sąsiedztwa otrzymanego poprzez aplikację dowolnych sekwencji tak zwanych *par wymian niezależnych*. Metoda ta pozwala analizować sąsiedztwo o wielkości wykładniczej w czasie wielomianowym. Możwe być osadzona w schemacie DS lub TS, dostarczając efektywnego narzędzia do budowy algorytmów LS.

Niech π będzie pewną permutacją na $\mathcal{J}$. Wymianę (swap) definiujemy poprzez parę zadań $\pi(a)$, $\pi(b)$ podlegających zamianie miejscami, dla pewnych $a, b \in \{1, 2, \ldots, \}$. Otoczeniem NPI nazywamy wszystkie rozwiązania, które możemy otrzymać z π poprzez dowolną wymianę jednej pary zadań. Otoczenie z niezależnymi wymianami (DPI, dynasearch swap) jest definiowane jako wszystkie permutacja, które można otrzymać z π stosując dowolny ciąg wymian parami niezaleznych. Pary wymian (a, b) i (c, d) są niezależne, jeśli $\max\{a, b\} \leqslant \min\{c, d\}$ lub $\min\{a, b\} \geqslant \max\{c, d\}$. Sąsiedztwo to zawiera $2^{n-1} - 1$ rozwiązań.

Wyznaczenie najlepszego rozwiązania w sąsiedztwi DPI jest możliwe poprzez zastosowanie schematu PD. Niech σ_j oznacza najlepsze rozwiązanie klasy DPI, które można otrzymać z π dla zadań ze zbioru $\pi(1), \ldots, \pi(j)$.

Oznaczmy odpowiednio przez $F(\sigma_j)$ wartość funkcji celu dla rozwiązania częściowego σ_j. Rozwiązanie σ_j otrzymuje się poprzez analizę wynikających z wprowadzenia zadania $\pi(j)$ (wraz z ewentualnymi jego wymianami typu DPI) do σ_{j-1}. Mogą przytym zajść dwa alternatywne przypadki: (a) zadanie $\pi(j)$ należy dołączyć na koniec σ_{j-1} generując rozwiązanie $\sigma_j^j = \sigma_{j-1}\pi(j)$, (b) należy wykonać wymianę pary zadań $pi(i)$, $\pi(j)$ dla pewnego $0 \leqslant i < j$, generując rozwiązanie $\sigma_j^i = \sigma_i\pi(j)\pi(i = 1)\ldots\pi(j-1)\pi(i)$. W pierwszym przypadku otrzymujemy

$$F(\sigma_j^j) = F(\sigma_{j-1}) + f_{\pi(j)}(P_j(\pi)) \tag{10.84}$$

gdzie $P_s(\pi) = \sum_{i=1}^{s} p_{\pi(i)}$. Ponieważ wszystkie wielkości $P_s(\pi)$, $s = 1, \ldots, n$ można wyznaczyć a priori, pojedyncza wartość (10.84) może być obliczona w czasie $O(1)$. W drugim przypadku, korzystając z optymalności σ_i, mamy

$$F(\sigma_j^i) = F(\sigma_i) + f_{\pi(j)}(P_i(\pi) + p_{\pi(j)})$$

$$+ \sum_{s=i+2}^{j-1} f_{\pi(s)}(P_s(\pi) + p_{\pi(i+1)} + p_{\pi(j)}) + f_{\pi(i+1)}(P_j(\pi)) \tag{10.85}$$

Obliczenie (10.85) wymaga czasu $O(n)$ bowiem $j - i < n$. Ostatecznie otrzymujemy zależność rekurencyjną pozwalającą wyznaczyć σ_j

$$F(\sigma_j^i) = \min_{0 \leqslant i \leqslant j} F(\sigma_j^i). \tag{10.86}$$

Dopełnieniem rekurencji w schemacie PD jest warunek początkowy $\sigma_0 = ()$, $F(\sigma_0) = 0$ oraz proces rozszerzania rozwiązania częściowego dla σ_j, $j = 1, \ldots, n$. Celem jest wyznaczenie σ_n. Postępowanie takie pozwala przeglądnąć otoczenie DPI o liczności $2^{n-1} - 1$ w czasie $O(n^3)$.

Możliwe jest zwiększenie sprawności obliczeniowej metody, przede wszystkim poprzez przyśpieszenie obliczania (10.85) oraz pomijanie niektorych rozwiązań. Ten ostatni pomysł sprowadza się do wstępnego przetestowania, za pomocą niekosztownego dolnego ograniczenia, czy σ_j^i rokuje nadzieję na uzyskanie rozwiązania lepszego niż najlepsze znane do tej pory w otoczeniu. Za początkowe górne ograniczenie przyjmuje się $UB = F(\sigma_j^j)$ zgodnie ze wzorem (10.84). Wielkość ta podlega aktualizacji przy każdym obliczaniu wartośći $F(\sigma_j^i)$. Korzystając z (10.85) otrzymujemy przy założeniu, że $p_{\pi(j)} \geqslant p_{\pi(i+1)}$ dolne ograniczenie

$$F(\sigma_j^i) \geqslant F(\sigma_i) + V_{\pi(j-1)} - V_{\pi(i+1)}$$

$$+f_{\pi(j)}(P_i(\pi)+p_{\pi(j)})+f_{\pi(i+1)}(P_j(\pi)) \stackrel{\text{def}}{=} LB_j^i \tag{10.87}$$

gdzie

$$V_{\pi(j)} = \sum_{s=1}^{j} f_{\pi(s)}(P_s(\pi)). \tag{10.88}$$

Jeśli $LB_j^i \geqslant UB$ to rozwiązanie σ_j^i należy pominąć bez potrzeby obliczania wzoru (10.85). Inne ograniczenia można otrzymać przyjmując szczególną postać funkcji kryterialnej. I tak dla $f_j(t) = w_j[t-d_j]k$, gdzie $[x]^+ = \max\{0, x\}$, mamy

$$\begin{aligned} F(\sigma_j^i) \geqslant\ & F(\sigma_i) + (p_{\pi(i+1)} - p_{\pi(j)})(W_{\pi(j-1)} - W_{\pi(i+1)}) \\ & +f_{\pi(j)}(P_i(\pi)+p_{\pi(j)})+f_{\pi(i+1)}(P_j(\pi)) \stackrel{\text{def}}{=} LB_j^i \end{aligned} \tag{10.89}$$

gdzie

$$W_{\pi(j)} = \sum_{s=1}^{j} w_{\pi(s)} U_s(\pi) \tag{10.90}$$

oraz $U_{\pi(s)} = 1$ jeśli $P_s(\pi) > d_{\pi(s)}$ oraz $U_{\pi(s)} = 0$ w przeciwnym przypadku.

10.8.2 Metody GA, SA, SJ

Dla problemów jednomaszynowych, w których rozwiązania są reprezentowane permutacjami, szereg metod takich jak GA, SA, SJ posiada schemat budowy algorytmu wspólny (podobny) dla wszystkich problemó permutacyjnych. Podejścia te opisane są bardziej szczegółowo w Rozdz. 12.10.

10.9 Uwagi

Powszechnie znana słabość dolnych ograniczeń dla problemó z kryterium addytywnym pociąga za sobą małą skuteczność metod dokładnych opartych na schemacie B&B, uniemożliwiając rozwiązanie przykładów o wielkości powyżej 100 zadań. Cecha ta powoduje także zgrubność oszacownia algorytmów przybliżonych mimi, iż w praktyce niektóre z nich zachowuja się dość dobrze, dostarczając rozwiązań bliskich optymalnemu w krótkim czasie. Spośród metod priorytetowych zaskakująco dobrą jakością charakteryzuje się reguła AU. Wśród algorytmów klasy LS, jak dotychczas najlepsze znane rezultaty otrzymywano dla wielostartowej wersji metody DS z otoczeniem DPI i serią losowych punktów startowych (kicks) generowanych na bazie ekstremów lokalnych zatrzymujacych pojedynczy przebieg DS [67]. Otrzymane rezultaty są lepsze od znanych implementacji TS oraz GS dla tego problemu. Algorytm ten pozwala rozwiązywać na PC, w czasie minut, przykłady

wielkość	zakres
p_j	$1 \dots 100$
w_j	$1 \dots 10$
d_j	$P(1 - TF - RDD/2) \dots P(1 - TF + RDD/2)$

Tabela 10.1: Dane dla problemu $1||\sum T_i$

o rozmiarze do 100 zadań, z dokładnością średnio poniżej 0.5%. Podejście dualne nie jest w tym względzie konkurencyjne, w sensie czasowym, chociaż umożliwia modelowanie innych ograniczeń, trudnych do uwzględnienia w metodach przybliżonych.

Przykłady testowe

Stopień trudności rozwiązywania przykładów problemów z kryterium addytywnym zależy równocześnie od wielu danych. Przykłady posiadajace "większą" wariancję danych lub większy ich zakres są zwykle łatwiejsze do rozwiązywania. Przykładowo, zaróno duży rozrzut wartości

d_j jak i w_j w problemie $1||\sum w_j T_j$ czyni go łatwiejszym. W celu precyzyjnego określenia klasy przykładów testowych, wspólnych dla wszystkich proponowanych algorytmów, opracowano schemat generacji przykładów dla problemu $1||\sum T_j$. Dane są generowane jako losowe liczby całkowite z przedziałów podanych w Tabl. **??**, gdzie $P = \sum_{j=1}^{n} p_j$. Wielkość RDD określa *względny zakres żądanych terminów zakończenia*, zwykle RDD=0.2,0.4,0.6,0.8,1.0, zaś TF *średni współczynnik spóźnienia*, zwykle TF=0.2,0.4,0.6,0.8,1.0. Dla każdej kombinacji RDD i TF jest generowane 5 przykładów, dostarczając 125 przykładów dla ustalonej wartości n. Zakres rozważanych wartości n jest ograniczony do 100. Przykłady te są także dostępne w Internecie [31].

11

Złożone problemy jednomaszynowe

Problemy z kryteriami nieregularnymi są zwykle wymieniane w kontekście systemów JIT, systemów z całościową analizą kosztów wytwarzania i składowania produktu oraz kosztów zaangażowania kapitału, systemów wytwarzania produktów nietrwałych, zmieniających swoje własności fizykochemiczne w czasie. Są one jednymi z najtrudniejszych problemów szeregowania, bowiem rozwiazanie optymalne nie leży w żadnej z klas uszeregowań wymienionych w Rozdz. 4.7. Powoduje to bezużyteczność większości "klasycznych" metod rozwiązywania oraz potrzebę rozwoju zupełnie nowych podejść. Przykładowo, najlepsze reguły priorytetowe pochodzące z problemów klasycznych dostarczają dla wielu zagadnień nieregularnych rozwiązań obarczonych błędem względnym rzędu 100–500%, co w porównaniu z osiąganymi rutynowo wartościami błędu 20–30% jest czymś zaskakujacym. Nowe podejścia zależą w sposób znaczący od typu zastosowanej funkcji kryterialnej. Prawie wszystkie znane problemy tej klasy są NP-trudne, przypadki wielomianowe są nieliczne. Niewiele też jest szczególnych własności problemów, jedną z częściej wymienianych jest własność V-kształtu. Dlatego też liczba problemów szczególnych oraz dedykowanych dla nich metod rozwiazywania jest zdecydowanie większa niż dla problemów klasycznych. Omówimy tylko niektóre z nich, głównie problemy jednomaszynowe, podkreślając różnice oraz związki z problemami znanymi.

11.1 Kary za przyśpieszenia i spóźnienia

Rozpatrzmy następujący problem wyjściowy. Należy wyznaczyć harmonogram pracy stanowiska zawierającego jedną maszynę przy założeniu, że zbiór zadań $\mathcal{J} = \{1, 2, \ldots, n\}$ ma być wykonywany na stanowisku sekwencyjnie, j-te zadanie posiada czas obsługi $p_j > 0$ oraz funkcję kary $h_j(t)$ posiadającą jedno, niekoniecznie jednoznaczne, minimum. Harmonogram jest określony poprzez wektor terminów rozpoczęcia wykonywania zadań $S = (S_1, \ldots, S_n)$, zaś kryterium polega na minimalizacji wartości wskaźnika określonego na wielkościach $h_j(S_j)$. Problemy szczególne otrzymujemy przyjmując pewne założenia o funkcjach $h_j(t)$ oraz o charakterze ich kombinacji w celu sformułowania kryterium optymalizacji.

Zanim przejdziemy do analizy własności problemu, pokażemy związek funkcji $h_j(t)$ z kryteriami klasy (4.3) oraz (4.4). Niech

$$h_j^* = \min_{-\infty<t<\infty} h_j(t) \tag{11.1}$$

będzie minimalną osiąganą wartością funkcji. Dokonajmy wpierw "lokalizacji" ekstremum niewłaściwego. Wielkość

$$a_j = \min\{t : \ h_j(t) = h_j^*\} \tag{11.2}$$

nazywamy pożądanym terminem rozpoczęcia zadania, zaś

$$b_j = \max\{t : \ h_j(t) = h_j^*\} + p_j \tag{11.3}$$

pożądanym terminem zakończenia. Przyśpieszeniem terminu rozpoczęcia zadania względem terminu a_j jest

$$E_j = [a_j - S_j]^+. \tag{11.4}$$

Spóźnieniem terminu zakończenia zadania wzgledem terminu b_j jest

$$T_j = [C_j - b_j]^+, \tag{11.5}$$

gdzie $C_j = S_j + p_j$. Z definicji mamy $E_j T_j = 0$ co znaczy, że zadanie może być dokładnie w jednym z trzech stanów: przyśpieszone ($E_j > 0$, $T_j = 0$), na czas ($E_j = 0 = T_j$), spóźnione ($E_j = 0$, $T_j > 0$). Zauważmy, że jeśli $E_j = 0 = T_j$ oraz $a_j < b_j - p_j$ to nie można określić dokładnie S_j; inaczej taka transformacja z E_j, T_j do S_j jest zawsze jednoznaczna. Kary $g_j()$, $f_j()$ definiujemy jako niemalejące funkcje przyśpieszenia i spóźnienia, odpowiednio, według zależności

$$g_j(E_j) = h_j(a_j - E_j) - h_j^* \tag{11.6}$$

dla $E_j \geqslant 0$ oraz

$$f_j(T_j) = h_j(b_j + T_j - p_j) - h_j^* \tag{11.7}$$

dla $T_j \geqslant 0$. Łatwo sprawdzić, że prawdziwe są związki

$$\sum_{j=1}^{n} h_j(S_j) = \sum_{j=1}^{n}(g_j(E_j) + f_j(T_j)) + \sum_{j=1}^{n} h_j^* \tag{11.8}$$

oraz

$$\max_{1\leqslant j\leqslant n} h_j(S_j) = \max_{1\leqslant j\leqslant n} \max\{g_j(E_j) + h_j^*, f_j(T_j) + h_j^*\}. \tag{11.9}$$

Ponieważ wszystkie kary za przyśpieszenia/spóźnienia można zapisać za pomocą funkcji $h_j(t)$, wprowadzony na początku rozdziału model jest ogólniejszy. W wielu przypadkach praktycznych zachodzi $h_j^* = 0$ co znakomicie upraszcza obliczenia, choć sytuacja $h_j^* > 0$ wcale nie jest trudniejsza do analizy. Mimo iż transformacja kryteriów (11.8)–(11.9) wydaje się zbędnym skomplikowaniem problemu, wiele algorytmów rozwiązywania odwołuje się bezpośrednio do wygodnych interpretacyjnie i intuicyjnie oczywistych pojęć zadanie przyśpieszone/spóźnione.

Niezależnie od przyjetej funkcji celu (4.3) czy (4.4), w literaturze wyróżniane są dwie podklasy problemów: (C) *ograniczone (constrained)*, które posiadaja warunek $S_j \geqslant 0$, $j \in \mathcal{J}$, oraz (U) *nieograniczone (unconstrained)*, które nie posiadają tego wymagania i dopuszczają ujemne terminy S_j. Wprawdzie podklasa C zawiera racjonalne ograniczenie, stosowane domyślnie dla wszystkich klasycznych problemów szeregowania, to podklasa U, dzięki mniejszej liczbie ograniczeń, jest łatwiejsza do analizy, bez jednoczesnej utraty istotnych własności występujących w podklasie C. Formalnie można dokonać transformacji problemu z podklasy C do U poprzez przedefiniowanie funkcji kosztu $h_j(t)$, wprowadzając wysoką barierową wartość kary dla argumentu $t < 0$. Należy jednak pamietać, że zabieg ten nie zawsze jest korzystny. Może on spowodować przesunięcie problemu do wyższej klasy złożoności, skomplikować metodą rozwiązywania, spowodować utratę znaczących własności funkcji (wypukłość, ciągłość, różniczkowalność). Czasami problemy podklasy U pozwalają łatwo "wbudować" ograniczenie $S_j \geqslant 0$ bez istotnej zmiany algorytmu rozwiązywania. Kolejne podklasy problemów, uzasadnione względami praktycznymi, wyróżniono ze względu na charakter uszeregowań. Są to: (Z) *zwarte*, w którym kolejne zadania muszą być wykonywane na maszynie bez przerw pomiędzy nimi, począwszy od pewnego momentu czasowego S_0 (czasami wymaga się $S_0 = 0$), (D) nie wymagające

powyższego warunku, w którym przerwy pomiędzy wykonywaniem zadań wynikają w sposób naturalny z funkcji kary.

Rozwiązaniem problemu jest wektor S dopuszczalnych terminów rozpoczęcia zadań, który implikuje jednoznacznie kolejność π wykonywania zadań, reprezentowaną permutacją na zbiorze $\mathcal{J}$. Odwrotne przejście, to znaczy wyznaczenie S dla danej kolejności π, jest zagadnieniem dużo bardziej skomplikowanym niż w przypadku kryteriów klasycznych. Postrzeganie S w związku z π prowadzi do dwupoziomowej metody rozwiązywania opartej na naturalnej dekompozycji problemu: wyznaczenie optymalnej kolejności π^* wykonywania zadań na poziomie górnym, oraz dla danej kolejności π – wyznaczenie optymalnych terminów rozpoczęcia zadań S na poziomie dolnym. Podejście takie, charakterystyczne dla problemów z regularnymi funkcjami celu, pozwala wykorzystać tutaj większość z podejść omówionych w Rozdz. 9 i 10. Dodatkowo, problemy wynikające z dekompozycji mogą być rozważane niezależnie, co ułatwia analizę problemu.

Poziom dolny. Kryterium addytywne

Rozpatrzmy wpierw problem dolnego poziomu. Przyjmując, że dana jest kolejność π wykonywania zadań na maszynie, optymalne terminy S rozpoczęcia zadań, można wyznaczyć poprzez rozwiązanie następującego problemu

$$\min_{S} \sum_{j=1}^{n} h_j(S_j) \tag{11.10}$$

$$S_{\pi(j)} + p_{\pi(j)} \leqslant S_{\pi(j+1)},\ j = 1, \ldots, n-1, \tag{11.11}$$

$$0 \leqslant S_{\pi(1)} \tag{11.12}$$

przy czym warunek (11.12) należy uwzględnić tylko dla problemu podklasy C. Oznaczmy dalej przez $H(\pi) = \sum_{j=1}^{n} h_j(S_j^*)$, gdzie S^* jest optymalnym rozwiązaniem problemu (11.10)–(11.12). Problem (11.10)–(11.12) jest zadaniem optymalizacji o n ciągłych zmiennych decyzyjnych, z nieliniową funkcją celu i liniowymi ograniczeniami. Czasami, w celu uniknięcia kłopotów z rozwiazaniem problemu (11.10)–(11.12), wprowadza się dodatkowe założenia o analitycznych własnościach funkcji $h_j(t)$, takie jak na przykład wypukłość. Dla pewnych szczególnych klas funkcji $h_j(t)$, przykładowo $h_j(t) = v_j[a_j - t]^+ + w_j[t + p_j - b_j]^+$, problem (11.10)–(11.11) można zapisać w formie zadania PL. W tym celu, należy wprowadzić $2n$ dodatkowych zmiennych E_j, T_j, $j = 1, \ldots, n$. Wzór (11.10) przyjmie wtedy postać formy

liniowej

$$\sum_{j=1}^{n}(v_j E_j + w_j T_j). \tag{11.13}$$

Dodatkowo należy wprowadzić warunki ograniczające

$$E_j \geqslant a_j - S_j,\ T_j \geqslant S_j + p_j - b_j,\ E_j \geqslant 0,\ T_j \geqslant 0,\ j = 1, \ldots, n. \tag{11.14}$$

Dla problemów jednomaszynowych ich związek z PL ma raczej teoretyczne niż praktyczne znaczenie, bowiem istnieją odpowiednie algorytmy wielomianowe o zdecydowanie mniejszej złożoności obliczeniowej. Dla zagadnień ogólniejszych (jak na przykład problem gniazdowy), korzystanie z modelu PL jest niezbędne. Jeśli problem należy do podklasy Z to do warunków (11.10)–(11.12) należy dodać ogranicznie

$$S_{\pi(j+1)} = S_{\pi(j)} + p_{\pi(j)},\ \ j = 1, \ldots, n-1, \tag{11.15}$$

które powoduje, że zagadnienie (11.10)–(11.12), (11.15) staje się *de facto* problemem optymalizacji z jedną zmienną decyzyjną $S_{\pi(1)}$. Mimo, iż problem wydaje się prosty, jego rozwiązywalność jest uzależniona od unimodalności (lub jej braku) przekształconej funkcji celu $\sum_{j=1}^{n} h_j(t + \sum_{i=1}^{j} p_{\pi(i)})$.

W pracy [101] podano wielomianowy algorytm wyznaczania $H(\pi)$ dla problemu z kryterium (4.4) oraz funkcją kary $h_j(t) = |t - b_j|$, nazywaną także funkcją rozbieżności (discrepancy). Wyznaczenie tej wartości jest możliwe poprzez krokową budowę ciągu rozwiązań częściowych. Bez straty ogólności rozważań możemy przyjąć, że $\pi(j) = j$, $j = 1, \ldots, n$. Stąd k-te rozwiązanie częściowe zawiera tylko k pierwszych zadań i jest charakteryzowane przez podanie terminów S_j, $j = 1, \ldots, k$. Rozwiązania te są generowane dla $k = 1, \ldots, n$. Kolejne, $k+1$-sze rozwiązanie częściowe otrzymuje się z rozwiązania k-tego realizując schemat PD implikujacy modyfikację wartości S_j, $j = 1, \ldots, k+1$ według procedury opisanej poniżej. Rozwiązanie k-te jest przedstawiane jako sekwencja bloków $B_{x,y}$, gdzie $x, y \in \{1, \ldots, k\}$. *Blokiem* nazywamy ciąg kolejnych zadań $x, \ldots, y$ taki, że

$$S_{x-1} + p_{x-1} < S_x \tag{11.16}$$

lub $x = 1$,

$$S_j + p_j = S_{j+1},\ j = x, \ldots, y-1, \tag{11.17}$$

$$S_y + p_y < S_{y+1} \tag{11.18}$$

lub $y + 1 = k$. Blok zawiera zadania wykonywane bez przerwy na maszynie. Dla każdego bloku $B_{x,y}$ definujemy zbiór zadań przyśpieszonych $\mathcal{E}_{x,y} = \{j \in$

$B_{x,y}: S_j < b_j\}$, zbiór zadań punktualnych $\mathcal{P}_{x,y} = \{j \in B_{x,y}: S_j = b_j\}$, zbiór zadań spóźnionych $\mathcal{T}_{x,y} = \{j \in B_{x,y}: S_j > b_j\}$. Ze względu na warunki (11.16) oraz (11.18) dla bloku $B_{x,y}$ muszą zachodzić warunki

$$|\mathcal{E}_{x,y} \cup \mathcal{P}_{x,y}| = |\mathcal{T}_{x,y}|\,, \tag{11.19}$$

$$|\mathcal{E}_{x,y}| = |\mathcal{T}_{x,y} \cup \mathcal{P}_{x,y}|\,. \tag{11.20}$$

Istotnie, w przeciwnym przypadku, cały blok można by "przesunąć" odpowiednio w lewo lub prawo na osi czasu, to znaczy podstawić $S_j := S_j + \Delta$, $j \in B_{x,y}$ dla pewnego Δ, zmniejszając jednocześnie wartość składnika $\sum_{j\in B_{x,y}} h_j(S_j)$. Przejdźmy zatem do opisu metody generowania rozwiązania $k+1$-szego. Niech $B_{w,k}$ będzie ostatnim blokiem w rozwiązaniu k-tym, dla pewnego $1 \leqslant w \leqslant k$. Dołączenie zadania $k+1$-szego implikuje jeden z poniższych przypadków:

1. Jeśli $S_k + p_k < b_{k+1}$ to zadanie $k+1$ utworzy samodzielny blok, zatem $S_{k+1} = b_k$, $B_{k+1,k+1} = (k+1)$.

2. Jeśli $S_k + p_k = b_{k+1}$ to zadanie $k+1$ zostanie dołączone do bloku $B_{w,k}$ bez potrzeby jego przesuwania, zatem $S_{k+1} = b_k$, blok $B_{w,k}$ należy zastąpić blokiem $B_{w,k+1} = (w, \ldots, k, k+1)$.

3. Jeśli $S_k + p_k > b_{k+1}$ to zadanie $k+1$ zostanie dołączone do bloku $B_{w,k}$, który następnie należy przesunąć krokowo w lewo. W tym celu wpierw podstawiamy $S_{k+1} = b_k$ zaś blok $B_{w,k}$ zastępujemy blokiem $B_{w,k+1} = (w, \ldots, k, k+1)$. Następnie realizujemy proces przesuwania, rozpoczynając od $x = w$, $y = k+1$ i wykonując nie więcej niż k iteracji. W każdej iteracji sprawdzamy warunek (11.19) dla $B_{x,y}$. Jeśli warunek jest spełniony lub $x = 1$ to koniec. Inaczej blok $B_{x,y}$ przesuwamy w lewo, to znaczy podstawiamy $S_j := S_j - \Delta$, $j \in B_{x,y}$, gdzie $\Delta = \min\{\min_{j\in T_{x,y}}(S_j - b_j), S_u - S_{u-1} - p_{u-1}\}$. Dla problemu podklasy C należy przy tym założyć $S_0 + p_0 = 0$, dla problemu podklasy D podstawiamy $S_0 + p_0 = -\infty$. Jeśli dla obliczonej wartości Δ zajdzie $\Delta = S_u - S_{u-1} - p_{u-1}$, to wówczas blok $B_{x,y}$ należy "skleić" z blokiem bezpośrednio go poprzedzającym zmieniając odpowiednio x. Zarówno przesunięcie bloku o wartość Δ jak i jego sklejenie powodują zmianę zbiorów $\mathcal{E}_{xy}$, $\mathcal{P}_{xy}$, $\mathcal{T}_{xy}$, które następnie wpływają na zachodzenie warunku (11.19). Przesuwanie kontynuowane jest do chwili spełnienia tego warunku.

Opisany algorytm wyznaczania $H(\pi)$ można zrealizować w formie procedury o złożoności obliczeniowej $O(n \log n)$ posługujac się odpowiednią strukturą

danych. Dane dotyczące bloku są zapisywane w stogu binarnym, jeden stóg dla każdego bloku, uporządkowanego według wartości $S_j - b_j$, $j \in \mathcal{T}_{x,y}$. Pozwala to na obliczenie wartości Δ w czasie $O(1)$. Ponieważ zarówno wybór elementu minimalnego, dodanie elementu jak i złączenie stogów można wykonać w czasie $O(\log n)$, złożoność całego algorytmu jest rzędu $O(n \log n)$.

Stosunkowo łatwo można uogólnić opisane podejście na przypadek ważonych funkcji rozbieżności $h_j(t) = v_j\,|t - b_j|$. Wówczas warunek (11.19) należy zastąpić przez

$$\sum_{j \in \mathcal{E}_{x,y} \cup \mathcal{P}_{x,y}} v_j = \sum_{j \in \mathcal{T}_{x,y}} v_j. \tag{11.21}$$

Nie zmienia się przy tym złożoność obliczeniowa odpowiedniego algorytmu wyznaczania $H(\pi)$. Dalsze rozszerzenia są możliwe dla bardziej ogólnych funkcji $h_j(t)$.

Warunki (11.19)–(11.20) oraz (11.21) określają tak zwaną własność V-kształtu dla bloku. Intuicyjnie, oznacza ona, że blok jako całość jest ulokowany optymalnie "w dolinie" funkcji.

Poziom dolny. Kryterium min-max.

Podobnie jak poprzednio, dla danej kolejności π wykonywania zadań na maszynie, terminy S_j, $j \in \mathcal{J}$, można wyznaczyć poprzez rozwiązanie następującego problemu optymalizacji nieliniowej

$$\min_{S} \max_{1 \leqslant j \leqslant n} h_j(S_j) \tag{11.22}$$

przy ograniczeniach (11.11) (oraz (11.12) jeśli wymaga tego problem podklasy C). W odróżnieniu od (11.10)–(11.12) problem (11.22) dla ustalonej π może być rozwiązany już dla ogólnych $h_j(t)$ w sposób efektywny, korzystając z ciągu problemów pomocniczych $1|r_j, q_j|C_{\max}$ oraz pewnych innych własności. Omówimy tą metodę przy założeniu, że $h_j^* = 0$, $j \in \mathcal{J}$, choć możliwe jest uzyskanie analogicznych rezultatów przy braku tych założeń.

Funkcję kary zapisujemy w formie $h_j(S_j) = \max\{g_j(E_j), f_j(T_j)\}$ odpowiednio do (11.9). Niech π będzie permutacją, zaś $H(\pi)$ minimalną wartością kryterium (11.22) dla tej permutacji. W pracy [344] pokazano, że zachodzi równość

$$H(\pi) = \max_{(x,y) \in I_{1,n}} H_{x,y}(\pi) \tag{11.23}$$

gdzie

$$I_{x,y} = \{(i,j) : x \leqslant i \leqslant j \leqslant y\}, \tag{11.24}$$

$$H_{x,y}(\pi) = \min_{0 \leqslant t \leqslant \Delta_{x,y}(\pi)} \max\{g_{\pi(x)}(t), f_{\pi(y)}(\Delta_{x,y}(\pi) - t)\} \tag{11.25}$$

oraz

$$\Delta_{x,y}(\pi) = [\delta_{x,y}(\pi)]^+ = [a_{\pi(x)} - b_{\pi(y)} + \sum_{s=x}^{y-1} p_{\pi(s)}]^+. \tag{11.26}$$

Wektor S^*, który zapewnia minimum (11.23) można znaleźć w następujący sposób. Wpierw wyznaczamy wartości

$$E'_j = \max\{t : g_j(t) \leqslant H(\pi)\} \tag{11.27}$$

gdzie $H(\pi)$ znaleziono zgodnie z (11.23). Następnie stosujemy wzór rekurencyjny

$$S^*_{\pi(1)} = a_{\pi(1)} - E'_{\pi(1)}, \tag{11.28}$$

$$S^*_{\pi(j)} = \max\{S^*_{\pi(j-1)} + p_{\pi(j-1)}, a_{\pi(j)} - E'_{\pi(j)}\}, \;\; j = 2, .., n. \tag{11.29}$$

Metoda ta nie wymaga żadnych dodatkowych założeń, takich jak na przykład ciągłość czy wypukłość, dla funkcji $h_j(t)$. Jeśli funkcje są ciągłe, to rozwiązanie problemu (11.25) można otrzymać wyznaczając rozwiązanie równania $g_{\pi(x)}(t) = f_{\pi(y)}(\Delta_{x,y}(\pi) - t)$. Dla często stosowanych funkcji odcinkami liniowych postaci

$$h_j(t) = v_j[a_j - t]^+ + w_j[t + p_j - b_j]^+, \tag{11.30}$$

rozwiązanie (11.25) przyjmuje postać

$$H_{x,y}(\pi) = \frac{v_{\pi(x)} w_{\pi(y)}}{v_{\pi(x)} + w_{\pi(y)}} \Delta_{x,y}(\pi). \tag{11.31}$$

Ponieważ $|I_{1,n}| = n(n+1)/2$, to $H(\pi)$ może być wyznaczone w czasie $O(n^2\omega)$ gdzie ω jest liczbą iteracji potrzebnych do rozwiązania problemu (11.25) dla danych x, y oraz $\Delta_{x,y}(\pi)$. Wektor S^* może być wyznaczony w czasie $O(n\tau)$ gdzie τ jest liczbą iteracji potrzebną do znalezienia E'_j dla ustalonego j. Dla funkcji kosztu postaci (11.30) zarówno ω jak i τ są $O(1)$.

W szczególności, gdy funkcje kary są postaci $h_j(t) = [t - a_j]^+ + [t + p_j - b_j]^+$ bezpośrednio z warunków (11.31) oraz (11.23) otrzymujemy $H(\pi) = \frac{1}{2}[L_{\max}(\pi)]^+$, gdzie $L_{\max}(\pi)$ jest wartością funkcji celu dla kolejności π w problemie $1|r_j, d_j|L_{\max}$. Ponieważ ten ostatni problem jest równoważny problemowi $1|r_j, q_j|C_{\max}$ analizowanemu szczegółowo w Rozdz. 9), stąd metody tam polecane mogą być zastosowane także do tego problemu.

Dla szczególnych postaci funkcji $h_j(t)$ mozliwe jest uzyskanie bardziej efektywnych algorytmów. I tak dla funkcji rozbieżności $h_j(t) = |t - b_j|$, stosujac podejście podobne do opisanego w rozdziale poprzednim, można skonstruować algorytm wyznaczajacy $H(\pi)$ w czasie $O(n \log n)$ [101].

Poziom górny

Nawet zakładając całkowitoliczbowość danych problemu, terminy S wyznaczone przez rozwiązanie zadania dolnego poziomu nie muszą być całkowite. Nie jest spełniona także zasada zapobiegania bezczynności maszyny ze względu na przerwy pomiędzy wykonywaniem zadań. Te dwa fakty wykluczają stosowanie podejścia dualnego opartego na dyskretyzacji czasu oraz podejścia opartego na PD, patrz Rozdz. 10.5 i 10.3, pozostawiając do rozwiązywania na poziomie górnym schemat B&B, metody priorytetowe oraz metody poszukiwań lokalnych. Pomijając bardzo kosztowny schemat B&B zajmijmy sie bliżej dwoma pozostałymi grupami algorytmów. Zastosowanie reguł priorytetowych jest polecane tylko do wyznaczania kolejności π, w kooperacji z jednokrotnym wywołaniem zadania dolnego poziomu. Podejscie to jest korzystne, na przykład dla funkcji rozbieżności, gdzie π można wyznaczyć poprzez uporządkowanie zadań według niemalejących wartości b_j. Niestety, już wprowadzenie różnych wag v_j do funkcji rozbieżności utrudnia ustalenie równowagi w preferencjach pomiędzy b_j a v_j. Sytuacja komplikuje się wyraźnie dla najczęściej spotykanych funkcji $h_j(t) = v_j[a_j - t]^+ + w_j[t + p_j - b_j]^+$ bowiem pod uwagę należałoby wziąć aż cztery, oddziałowujace na siebie wzajemnie, parametry. Stąd podejście priorytetowe, choć najprostsze, nie wypada zbyt korzystnie w analizie eksperymentalnej [344]. Korzystniejszym podejściem jest krokowa budowa rozwiązania częściowego przy wykorzystaniu techniki INS [344], analogiczna do algorytmu NEH, patrz także Rozdz. 12.5.

W odniesieniu do metod LS, zaskakująco dobre rezultaty uzyskuje się już dla najprostszej metody DS z wymianą par zadań przyległych. Niestety poważnym problemem pozostaje wciaż znaczny koszt przeszukiwania otoczenia, wynikajacy głównie ze złożoności obliczeniowej wyznaczenia wartości funkcji celu pojedynczego rozwiązania. Stąd tendencja do redukcji wielkości otoczeń oraz przyśpieszania obliczeń poprzez wykorzystanie własności V-kształtu. Biorąc pod uwagą to zagadnienie, uzasadnione jest też stosowanie także metody SA.

11.2 Szeregowanie w systemach JIT

Kryteria nieregularne są traktowane jako podstawowe dla systemów wytwarzania w strategii dokładnie-na-czas (JIT), w której zachodzi potrzeba precyzyjnego dostarczania wyrobów. Początkowe prace koncentrowały się głównie na kryteriach klasy (4.3) bądź (4.4), reprezentujących kary za zbyt wczesne lub zbyt późne zakończenie zadań. Metody rozwiązywania tego typu

problemów omówiono bardziej szczegółowo w Rozdz. 11. Przegląd w pracy [26] dostarcza innych postaci kryteriów nieregularnych, opartych na wariancji poziomu produkcji, które reprezentują nie analizowane jeszcze klasy (4.27)–(4.28) funkcji kryterialnych.

Problem PRV. Kryterium addytywne.

Dany jest zbiór n rożnych typów części, wytwarzanych cyklicznie na jednym stanowisku i dostarczanych rytmicznie w ilościach $t_1, \ldots, t_n$ na jeden cykl, odpowiednio. Przygotowanie każdej części odpowiada wykonaniu czynności o jednostkowym czasie wykonywania. Stąd podstawowy cykl stanowiska trwa $T = \sum_{i=1}^{n} t_i$ jednostek czasu, zaś stan stanowiska może być analizowany tylko w dyskretnych chwilach czasowych $1, \ldots, T$.

Oznaczmy przez $s_i = t_i/T$ strumień zapotrzebowanie na części typu j żądany na wyjściu tego stanowiska. Celem jest wytwarzanie przez stanowisko części w sposób rytmiczny, to znaczy tak by w każdym momencie czasu proporcja zakumulowanej produkcji części i do całkowitej produkcji była możliwie bliska s_i. Problem ten można przedstawić w postaci zadania programowania dyskretnego. Niech x_{ik} będzie łączną (zakumulowaną) liczbą części typu i wytworzonych w przedziałach czasu od 1 do k.

$$\min_{x} \sum_{k=1}^{T} \sum_{j=1}^{n} F_j(x_{jk} - kr_j) \tag{11.32}$$

przy ograniczeniach

$$\sum_{j=1}^{n} x_{jk} = k, \; k = 1, \ldots, T, \tag{11.33}$$

$$0 \leqslant x_{jk} - x_{j,k-1} \leqslant 1, \; j = 1, \ldots, n, \; k = 1, \ldots, T, \tag{11.34}$$

$$x_{jk} \text{ całkowite}, \; j = 1, \ldots, n, \; k = 1, \ldots, T. \tag{11.35}$$

Funkcje $F_i()$ są pewnymi funkcjami kary, unimodalnymi, wypukłymi, z warunkiem $F_i(0) = 0$, $i = 1, \ldots, n$. Warunek (11.33) zapewnia, że dokładnie k części będzie uszeregowanych w przedziałach $1, \ldots, k$. Warunki (11.34)–(11.35) zapewniają, że w każdym jednostkowym przedziale czasu jest wykonywana co najwyżej jedna część jakiegoś typu. Warunek $x_{iT} \leqslant t_i$, $i = 1, \ldots, n$ nie został włączony do ograniczeń bowiem każde rozwiązanie optymalne problemu (11.32)–(11.35) zapewnia jego automatyczne spełnienie.

Problem wyjściowy możemy potraktować jako problem szeregowania zbioru jednostkowych zadań $\mathcal{J}$ o liczności $\sum_{i=1}^{n} t_i = T$, przy specyficznej funkcji kryterialnej. Ze względu na jednostkowy czas zadań, problem można

rozwiązywać korzystając z *liniowego zagadnienia przydziału*, jeśli tylko potrafimy efektywnie wyznaczyć współczynniki macierzy kosztów przydziału. Ze względu na powtarzalność części, zbiór zadań przedstawmy w formie $\mathcal{J} = \{(i,j) : \ j = 1,\ldots,t_i,\ i = 1,\ldots,n\}$. Skoncentrujmy się przez chwilę na częściach jednego typu i. Rytmiczność dostarczania pozwala nam określić *idealną pozycję* Z_{ij}^* dla zadania $(i,j) \in \mathcal{J}$. Jest ona równa $Z_{ij}^* = \lceil k_{ij} \rceil$, gdzie

$$k_{ij} = \frac{j - z_i}{s_i},\ j = 1,\ldots,t_i,\ i = 1,\ldots,n \tag{11.36}$$

oraz z_i jest rozwiązaniem równania

$$F_i(z_i) = F_i(z_i - 1),\ i = 1,\ldots,n. \tag{11.37}$$

Jeśli $F_i()$ jest funkcja parzystą to $z_i = 0.5$. Następnie wprowadzamy koszt c_{ijk} przydziału zadania $(i,j) \in \mathcal{J}$ do przedziału k reprezentujący *karę* za zbyt wczesne lub zbyt późne jego zakończenie w stosunku do założonej pozycji idealnej. Kara ta przyjmuje postać

$$c_{ijk} = \begin{cases} \sum_{l=k}^{Z_{ij}^*-1} w_{ijl} & \text{gdy} \quad k < Z_{ij}^* \\ 0 & \text{gdy} \quad k = Z_{ij}^* \\ \sum_{l=Z_{ij}^*}^{k-1} w_{ijl} & \text{gdy} \quad k > Z_{ij}^* \end{cases} \tag{11.38}$$

gdzie

$$w_{ijl} = \left| F_i(j - s_i l) - F_i(j - 1 - s_i l) \right|,\ (i,j) \in \mathcal{J},\ l = 1,\ldots,T. \tag{11.39}$$

Ostatecznie, problem (11.32)–(11.35) możemy przekształcić do następującego liniowego problemu przydziału. Niech y_{ijk} będzie zmienną binarną równą 1 jeśli zadanie (i,j) jest przydzielone do przedziału k, $(i,j) \in \mathcal{J}$, $k = 1,\ldots,T$. Problem (11.34)–(11.35) zapisujemy jako

$$\min_y \sum_{k=1}^{T} \sum_{(i,j)\in\mathcal{J}} c_{ijk} y_{ijk} \tag{11.40}$$

$$\sum_{(i,j)\in\mathcal{J}} y_{ijk} = 1,\ k = 1,\ldots,T, \tag{11.41}$$

$$\sum_{k=1}^{T} y_{ijk} = 1,\ (i,j) \in \mathcal{J}, \tag{11.42}$$

$$y_{ijk} \in \{0,1\},\ (i,j) \in \mathcal{J},\ k = 1,\ldots,T. \tag{11.43}$$

Zauważmy, że $|\mathcal{J}| = T$, stąd $c_{T\times T}$. Problem (11.40)–(11.43) jest problemowi przydziału (AP) zawierającego T zmiennych, dla którego istnieje algorytm wielomianowy o złożoności obliczeniowej $O(T^3)$. Najlepsze znane implementacje równoległe pozwalają rozwiązywać efektywnie przykłady AP z $T \leqslant 30,000$.

Można skonstruować też inne, intuicyjnie naturalne podejście, bazujące na alternatywnych funkcjach kary oraz idealnej pozycji zadania. Nie jest ona jednak równoważne problemowi (11.32)–(11.35). Pozycję *idealną* redefiniujemy jako $k_{ij} = (2j-1)/(2r_i)$. Taka definicja odpowiada poprzedniej, jeśli tylko funkcja $F_i()$ jest parzysta. Dla uproszczenia notacji elementy zbioru $\mathcal{J}$ będziemy oznaczali pojedynczymi literami, podobnie jak ich pozycje. Niech C_v oznacza termin zakończenia zadania $v = (i,j) \in \mathcal{J}$. Wartość C_v zależy od położenie zadania v w uszeregowaniu. Ponieważ $\sum_{i=1}^{n} t_i = T$, wszystkie zadania są wykonywane bez przestoju maszyny w przedziale $[0,T]$. Stąd uszeregowanie może być jednoznacznie reprezentowane permutacją na $\mathcal{J}$ postaci $\pi = (\pi(1),\ldots,\pi(T))$, gdzie $\pi(v) \in \mathcal{J}$. Oczywiście potrzebne jest spełnienie warunku następstwa zdarzeń dla sekwencji kolejno wykonywanych zadań w postaci $C_{\pi(v)} + 1 \leqslant C_{\pi(v+1)}$ wynikające z jednostkowego czasu wykonywania zadań. Jako kryterium przyjmujemy wyrażenie

$$\min_{\pi} \sum_{v\in\mathcal{J}} (C_{\pi(v)} - k_{\pi(v)})^2, \tag{11.44}$$

które należy minimalizować po wszystkich π. Jest to szczególna postać kryterium (4.4) oraz naturalne rozszerzenie miary (4.26) na przypadek różnych terminów dostawy. Można pokazać, że algorytm EDD (earliest due date) wyznacza optymalne rozwiązanie problemu (11.44). Zmiana postaci funkcji kary na $\sum_{v\in\mathcal{J}} \left|C_{\pi(v)} - k_{\pi(v)}\right|$ nie zmienia sposobu rozwiązania problemu – reguła EDD wciąż dostarcza rozwiązanie optymalne, patrz Rozdz. 11.1. Algorytm EDD może być implementowany w formie procedury o złożoności $O(nT)$.

Problem PRV. Kryterium min-max.

Zamiast addytywnej miary rozproszenia wyjścia (11.32) można zastosować miarę opartą na maksymalnym rozproszeniu w postaci

$$\min_{x} \max_{1\leqslant k\leqslant T} \max_{1\leqslant i\leqslant n} |x_{ik} - ks_i| \tag{11.45}$$

przy ograniczeniach (11.33)–(11.35). Problem można rozwiązać sprawdzając, dla danego V, czy istnieje rozwiązanie dopuszczalne x, w którym wartość

funkcji kryterialnej (11.45) jest nie większa od V. Stosując schemat szukania binarnego ze względu na V, pomiędzy jego dolną $\underline{V}$ i górną $\overline{V}$ granicą, znajdziemy optymalne V^* oraz poszukiwane rozwiązanie optymalne x^*. Weryfikacja, czy dla danego V istnieje rozwiązanie dopuszczalne jest równoważna problemowi szeregowania zadań ze zbioru $\mathcal{J}$ na jednej maszynie z terminami gotowości r_{ij}, żądanymi terminami zakończenia d_{ij} (problem oznaczony $1|p_j = 1, r_j, C_j \leqslant d_j|\gamma$), w którym poszukuje się dowolnego rozwiązania dopuszczalnego. Dla danego V wartości r_{ij}, d_{ij} wyznacza się jako liczby całkowite z warunków

$$\frac{j - V}{s_i} \leqslant r_{ij} < \frac{j - V}{s_i} \tag{11.46}$$

$$\frac{j - 1 + V}{s_i} < d_{ij} \leqslant \frac{j - 1 + V}{s_i} + 1 \tag{11.47}$$

Sprawdzenie czy dla danego V istnieje rozwiązanie dopuszczalne może być wykonane w czasie $O(T)$. Granice dla poszukiwania V^* można określić jako

$$\underline{V} = 1, \quad \overline{V} = 1 - \max_{1 \leqslant i \leqslant n} s_i. \tag{11.48}$$

Problem (11.45) ma zawsze rozwiązanie dopuszczalne, w którym, dla każdej chwili czasowej, odchylenie aktualnej wielkości produkcji od wielkości idealnej nie przekracza wartości 1 dla każdej z n części.

Jeszcze innym podejściem do problemu z kryterium min-max jest sformułowanie i rozwiązanie następującego minimaksowego problemu przydziału

$$\min_{x} \max_{1 \leqslant k \leqslant T} \max_{(i,j) \in \mathcal{J}} c_{ijk} y_{ijk} \tag{11.49}$$

przy ograniczeniach (11.41)–(11.43), gdzie znaczenia wielkości c_{ijk} oraz y_{ijk} są takie same jak dla problemu (11.40)–(11.43).

Problem ORV

Rozważmy system jak poprzednio, wytwarzający cyklicznie n produktów w ilościach $t_1, \ldots, t_n$, odpowiednio, w cyklu o długości $T = \sum_{i=1}^{n} t_i$. Załóżmy, że produkty są wytwarzane na linii montażowej, zasilanej przez wejścia podprocesów działających w oparciu o strategię ssania. Zauważmy, że mając określone wielkości zapotrzebowania na produkty wytwarzane przez linię, możemy określić precyzyjnie wielkości zapotrzebowania zgłoszone na wyjsciach wszystkich poprocesów zasilajacych.

Zakładając niezależność podprocesów, możemy dalsze rozważania ograniczyć do jednego z nich, bowiem każdy może być tak samo analizowany. Niech m będzie liczbą wyjść tego podprocesu. Oznaczmy przez u_{ij}, $j = 1, \ldots, m$ liczbe jednostek wyjścia j wymaganą na jednostkę produktu i, $i = 1, \ldots, n$. Oznaczmy przez s_j frakcję zapotrzebowania na wyjście j

$$s_j = \frac{\sum_{i=1}^{n} t_i u_{ij}}{\sum_{j=1}^{m} \sum_{i=1}^{n} t_i u_{ij}}. \tag{11.50}$$

Bieżące zakumulowane zapotrzebowanie na wyjście j w pierwszych k przedziałach czasu jest równe $\sum_{i=1}^{n} u_{ij} x_{ik}$, zaś bieżące zakumulowane zapotrzebowanie na wszystkie wyjścia w tym samym okresie czasu jest równe $\sum_{j=1}^{m} \sum_{i=1}^{n} u_{ij} x_{ik}$. Oczekujemy, że pierwsza z wymienionych wartości będzie tak blika wartości s_j jak to jest tylko możliwe, dla wszystkich przedziałów $1, \ldots, k$. Rozbieżność pomiędzy nimi dla ustalonego $1 \leqslant j \leqslant m$ i $1 \leqslant k \leqslant T$ wynosi

$$\sum_{i=1}^{n} (u_{ij} - s_j \sum_{j=1}^{m} u_{ij}) x_{ik}. \tag{11.51}$$

Sumując rozbieżności oraz korzystając podobnie jak poprzednio z funkcji kary otrzymujemy problem ORV w postaci

$$\min_x \sum_{k=1}^{T} \sum_{j=1}^{m} F_j (\sum_{i=1}^{n} (u_{ij} - s_j \sum_{j=1}^{m} u_{ij}) x_{ik}) \tag{11.52}$$

przy ograniczeniach (11.33)–(11.35). Funkcje $F_j()$ są unimodalne, wypukłe, posiadające własność $\min_x F_j(x) = F(0) = 0$, $j = 1, \ldots, m$.

12

Podstawowe problemy przepływowe

Problemy przepływowe są jednymi z prostszych i częściej analizowanych modeli systemów produkcyjnych. Mają bardzo dobre aplikacje praktyczne, głównie w procesach chemicznych, niektórych procesach przemysłu elektronicznego czy samochodowego, patrz przegląd zastosowań w [344]. Tworzą podstawy do analizy bardziej złożonych struktur szeregowo–równoległych (potokowe linie automatyczne) dominujących we współczesnych systemach wytwarzania. Są także traktowane jako wskaźnik praktycznych możliwości rozwiązywania trudnych numerycznie problemów przy użyciu narzędzi i metod teorii szeregowania zadań. Istnieje wiele odmian problemów przepływowych różniących się wprowadzonymi dodatkowymi ograniczeniami. Wyjąwszy nieliczne przypadki szczególne, problemy te są NP-trudne. Zatem kosztowne obliczeniowo metody dokładne mogą być stosowane tylko dla przykładów o niewielkim rozmiarze lub tych, w których stosunek zysku do nakładu obliczeń jest duży, np. problemy z kryterium czasu cyklu. Dla większych przykładów zalecane są szybkie algorytmy przybliżone o różnych cechach numerycznych, patrz przegląd w pracy [265,344]. Aktualnie najlepsze znane metody pozwalają rozwiązać problemy z kryterium $C_{\max}$ o rozmiarze 500 zadań i 20 maszyn (10,000 operacji) w czasie minut na PC z dokładnością poniżej 1-1.5% [269]. W niniejszym rozdziale będziemy analizować niektóre problemy przepływowe w kolejności wzrastającej stopniowo ogólności.

12.1 Problem podstawowy

Zbiór zadań $\mathcal{J} = \{1, 2, \ldots, n\}$ jest przeznaczony do wykonywania na maszynach $\mathcal{M} = \{1, 2, \ldots, m\}$ o ograniczonej jednostkowej przepustowości, w podanej kolejności. Zadanie $j \in \mathcal{J}$ składa się z ciągu operacji $(O_{1j}, \ldots, O_{mj})$; operacja O_{ij} odpowiada nieprzerywalnemu wykonaniu zadania j na maszynie i czasie p_{ij}. Rozwiązaniem jest harmonogram pracy maszyn reprezentowany przez macierze terminów rozpoczęcia $S = (S_1, \ldots, S_n)$, $S_j = (S_{1j}, \ldots, S_{mj})$ oraz zakończenia zadań $C = (C_1, \ldots, C_n)$, $C_j = (C_{1j}, \ldots, C_{mj})$, spełniające powyższe ograniczenia. W praktyce, ponieważ $C_{ij} = S_{ij} + p_{ij}$, zatem rozwiązanie jest całkowicie charakteryzowane przez jedną z tych macierzy. Jeśli funkcja celu jest regularna, to harmonogram optymalny jest "dosunięty w lewo na osi czasu", zatem można poszukiwać w zbiorze takich rozwiązań. Wtedy też każdy harmonogram może być jednoznacznie reprezentowany kolejnością wykonywania zadań na maszynie i, ta zaś z kolei jest reprezentowana permutacją $\pi_i = (\pi_i(1), \ldots, \pi_i(n))$ na zbiorze $\mathcal{J}$. Jeśli permutacje π_i mogą być różne dla różnych i to odpowiedni problem oznaczamy jako "ogólny" (F), jeśli zaś wszystkie permutacje π_i są takie same to problem określamy jako "permutacyjny" (F*). Rozwiązania problemu F* tworzą podklasę rozwiązań problem F. Mimo, że zagadnienie F pozwala uzyskiwać mniejsze wartości funkcji celu, problem F* jest analizowany częściej. Powodów jest kilka: (1) liczba rozwiązań problem F* jest równa $n!$ i zdecydowanie mniejsza niż $(n!)^m$ dla problemu F, (2) sterowanie systemem F* jest prostsze i wymaga określenia tylko (jednej) kolejności wprowadzania zadań do systemu zakładając, że obsługa na kolejnych stanowiskach odbywa się według reguły FIFO (3) szereg systemów wytwarzania, ze względów technologicznych, zezwala tylko na rozwiązania permutacyjne, (4) dla niektórych klas problemów przepływowych rozwiązania optymalne leżą w klasie rozwiązań permutacyjnych, (5) błąd pomiędzy rozwiązaniami optymalnymi problemu F i F* jest nieznaczny, (6) rozwiązania problemu F* mogą być używane jako rozwiązania początkowe w algorytmach przybliżonych dla problemów F. Niewielki błąd wymieniony w punkcie (5) wynika wyłącznie z analizy eksperymentalnej. W teorii błąd ten może w skrajnych przypadkach osiągać znacznie większe wartości. W pracy [293] pokazano, że stosunek optymalnych wartości $C_{\max}$ problemów permutacyjnego i niepermutacyjnego może być teoretycznie aż rzędu $O(\sqrt{m})$, choć w praktyce różnice są zdecydowanie mniejsze.

W tym rozdziale będziemy się zajmowali tylko przypadkiem permutacyjnym problemu przepływowego; wersja niepermutacyjna jest zwykle analizowana narzędziami charakterystycznymi dla problemów ogólniejszych. Przyj-

mując, że znana jest kolejność wykonywania zadań (określona permutacją π na $\mathcal{J}$) w problemie przepływowym permutacyjnym, terminy zakończenia wykonywania zadań można wyznaczyć z warunków

$$C_{i\pi(j)} \leqslant S_{i\pi(j+1)}, \;\; j = 1, \ldots, n-1, \;\; i = 1, \ldots, m, \tag{12.1}$$

$$C_{ij} \leqslant S_{i+1,j}, \;\; i = 1, \ldots, m-1, \;\; j = 1, \ldots, n, \tag{12.2}$$

które prowadzą do oczywistego wzoru rekurencyjnego

$$C_{i\pi(j)} = \max\{C_{i\pi(j-1)}, C_{i-1,\pi(j)}\} + p_{i\pi(j)}, \;\; j = 1, \ldots, n, \tag{12.3}$$

liczonego dla $i = 1, \ldots, m$, gdzie $\pi(0) = 0$, $C_{i0} = 0$, $i = 1, \ldots, m$, $C_{0j} = 0$, $j = 1, \ldots, n$. Dla bardziej złożonych analiz używa się też równoważnej nierekurencyjnej wersji wzoru (12.3) w postaci

$$C_{i\pi(j)} = \max_{1 \leqslant j_0 \leqslant j_1 \leqslant \ldots \leqslant j_i \leqslant j} \sum_{s=1}^{i} \sum_{t=j_{s-1}}^{j_s} p_{s\pi(t)}. \tag{12.4}$$

Mimo złożonej postaci wzoru (12.4) wszystkie wartości C_{ij} można wyznaczyć dla danej π w czasie $O(nm)$.

Dla problemów F* z kryterium minimalizacji długości uszeregowania poszukiwana jest permutacja $\pi^* \in \Pi$, dla której

$$C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi), \tag{12.5}$$

gdzie $C_{\max}(\pi) = C_{m\pi(n)}$.

12.2 Przypadki wielomianowe.

Historycznie najstarszym i najbardziej znanym rezultatem jest przypadek dla $F^*2||C_{\max}$ posiadający algorytm wielomianowy [185]. Istnieje kilka równoważnych postaci tego algorytmu. Wersja pierwotna buduje permutację poczynając jednocześnie od obu jej końców. Spośród zadań jeszcze nie uszeregowanych wybierane jest to, które ma najmniejszą wartość $\min\{p_{1k}, p_{2k}\}$. Jeśli wartość minimalna odpowiada p_{1k} to zadanie jest szeregowane na pierwszej wolnej pozycji, jeśli zaś odpowiada wartości p_{2k} – to szeregowana jest na ostatniej wolnej pozycji. Alternatywnym algorytmem jest wygodna i szybka technika dwu-krokowa: wpierw wszystkie zadania są dzielone na dwa rozłączne podzbiory $A = \{j : \; p_{1j} \leqslant p_{2j}\}$ oraz $B = \{j : \; p_{1j} > p_{2j}\}$. Zadania z A są ustawiane w kolejności niemalejących

wartości p_{1j} dostarczając permutacji π_A, zaś te z B w kolejności nierosnących wartości p_{2j} dostarczając permutacji π_B. Permutacją optymalną jest konkatenacja $\pi^* = \pi_A\pi_B$. Kolejna równoważna wersja algorytmu tworzy permutację odpowiadającą uporządkowaniu zadań według niemalejących wartości priorytetów

$$v_j = \frac{\mathrm{sign(p_{1j} - p_{2j})}}{\min\{p_{1j}, p_{2j}\}}, \tag{12.6}$$

gdzie $sign()$ jest funkcją znaku. Niezależnie od wybranej wersji algorytmu, złożoność obliczeniowa problemu $F^*2||C_{\max}$ jest $O(n \log n)$.

Wprowadzenie dodatkowych, nawet nieznacznych, ograniczeń powoduje, że problem staje się NP-trudny. I tak, wprowadzenie relacji poprzedzań R pomiędzy zadaniami pozostawia problem w klasie P jedynie wtedy gdy R ma strukturę szeregowo-równoległą; dla ogólnej postaci R problem jest NP-trudny. Podobnie, wprowadzenie niezerowych terminów gotowości implikuje NP-trudność. Odmiennie, wprowadzenie ograniczeń typu *no store* lub *no wait* pozostawia problem w klasie P, jednak już dopuszczenie niezerowych buforów pośredniczących implikuje jego NP-trudność [160, 281].

Pewne szczególne przypadki problemu $F^*3||C_{\max}$ mogą być rozwiązane przez umiejętne sprowadzenie do problemu $F^*2||C_{\max}$.

- Problem $F^*2||C_{\max}$ z czasami p_{ij} ma taką samą optymalną kolejność wykonywania zadań jak problem z czasami $p_{ij} + c$, dla pewnej stałej $c \geqslant 0$. Ten dość oczywisty, w kontekście wzoru (12.6), rezultat można rozszerzyć na inne transformacje czasów p_{ij} nie zmieniające uporządkowania wynikającego z (12.6). Przykładowo, podobną cechę ma mnożenie przez stałą $c > 0$.

- Jeżeli w problemie $F^*3||C_{\max}$ spełniony jest jeden z warunków: (a) $p_{1i} \geqslant p_{2j}$, $i, j \in \mathcal{N}$, lub (b) $p_{2i} \geqslant p_{3j}$, $i, j \in \mathcal{N}$ to optymalna kolejność wykonywania zadań może być wyznaczona poprzez rozwiązanie problemu $F^*2||C_{\max}$ z czasami wykonywania p'_{ij}, gdzie $p'_{1j} = p_{1j} + p_{2j}$, $p'_{2j} = p_{2j} + p_{3j}$.

- Jeżeli w problemie $F^*3||C_{\max}$ spełniony jest jeden z warunków: (a) $p_{1i} \leqslant p_{2j}$, $i, j \in \mathcal{N}$, lub (b) $p_{2i} \geqslant p_{3j}$, $i, j \in \mathcal{N}$ to optymalna kolejność wykonywania zadań może być wyznaczona poprzez rozwiązanie n problemów $F^*2||C_{\max}$ odpowiadających n możliwym wyborom zadania pierwszego lub ostatniego odpowiednio, z czasami wykonywnia p'_{ij}, gdzie $p'_{1j} = p_{1j} + p_{2j}$, $p'_{2j} = p_{2j} + p_{3j}$.

- Problem $F^*3|M_2 - non - bottl.|C_{\max}$ może być rozwiązany za pomocą problemu $F^*2||C_{\max}$. Istnieje optymalna permutacja zadań dla pierwszego z nich taka, że jest ona rozwiązaniem optymalnym problemu pomocniczego $F^*2||C_{\max}$ z czasami wykonywania zadań p'_{ij}, $i = 1, 2$, $j = 1, \ldots, n$ określonymi następująco $p'_{1j} = p_{1j} + p_{2j}$, $p'_{2j} = p_{2j} + p_{3j}$, $j = 1, \ldots, n$. Należy przy tym podkreślić, że problem pomocniczy dostarcza tylko permutacji π, zaś odpowiednie terminy zakończenia wykonywania zadań dla problemu $F^*3|M_2 - non - bottl.|C_{\max}$ należy wyznaczyć posługując się danymi p_{ij}, przy czym dla maszyn i o ograniczonej przepustowości należy skorzystać z zależności (12.3), zaś dla maszyn o nieograniczonej przepustowości należy podstawić

$$C_{i\pi(j)} = C_{i-1,\pi(j)} + p_{i\pi(j)}, \quad j = 1, \ldots, n. \tag{12.7}$$

- Każda kolejność wykonywania zadań w problemie $F^*|pmtn|C_{\max}$ może być transformowana do takiej kolejności realizacji zadań w odpowiednim problemie $F^*||C_{\max}$, że wartości funkcji celu obu problemów są równe oraz zadania na maszynach 1 i m są realizowane w sposób nieprzerywalny. Własność ta gwarantuje, że rozwiązanie optymalne problemu $F^*2||C_{\max}$ jest również rozwiązaniem optymalnym problemu $F^*2|pmtn|C_{\max}$.

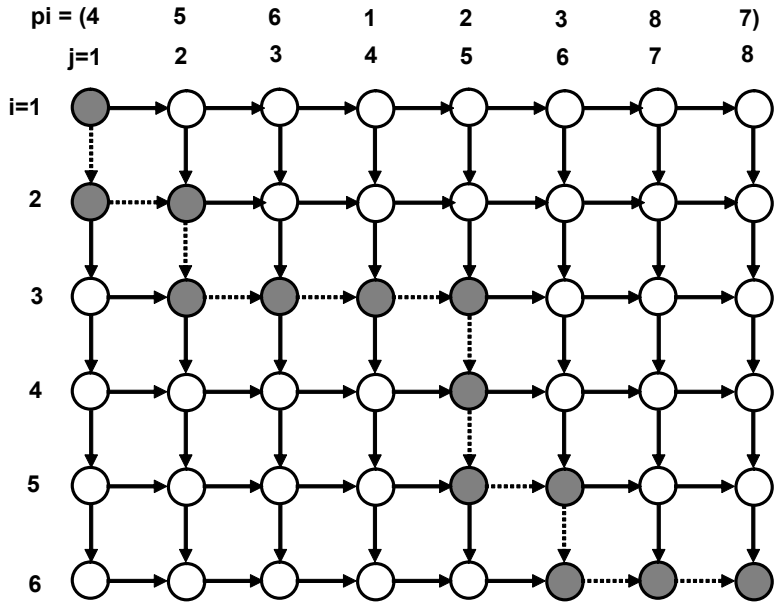
12.3 Pewne własności problemu

Dla $m > 2$ problem $F^*||C_{\max}$ jest NP-trudny nawet, jeśli każde z zadań ma tylko dwie operacje o niezerowym czasie trwania. Z tego względu włożono dużo wysiłku w badania własności problemu. Wykazano, że istnieje optymalne rozwiązanie problemu $F||\gamma$ (niepermutacyjnego) z takim samym porządkiem wykonywania zadań na maszynach 1 i 2 [68]. Podobnie, istnieje optymalne rozwiązanie problemu $F||C_{\max}$ z takim samym porządkiem na maszynach $m-1$ i m. Z własności wynika, że problemy $F2||\gamma$ i $F^*2||\gamma$ oraz $F3||C_{\max}$ i $F^*3||C_{\max}$ są równoważne [185].

Niech π będzie pewną ustaloną permutacją w problemie F^*. Terminy rozpoczęcia zadań S_{ij} muszą spełniać naturalne ograniczenia kolejności przepływu zadań przez maszyny

$$S_{i-1,j} + p_{ij} \leqslant S_{i,j}, \; i = 1, \ldots, m, \; j = 1, \ldots, n, \tag{12.8}$$

$$S_{i\pi(j-1)} + p_{i\pi(j-1)} \leqslant S_{i\pi(j)}, \; i = 1, \ldots, m, \; j = 1, \ldots, n, \tag{12.9}$$



Rysunek 12.1: Graf dla permutacji

gdzie $\pi(0) \stackrel{\text{def}}{=} 0$ oraz $S_{0j} \stackrel{\text{def}}{=} 0$, $j = 1, \ldots, n$, $S_{i0} \stackrel{\text{def}}{=} 0$, $p_{i0} \stackrel{\text{def}}{=} 0$, $i = 1, \ldots, m$. Terminy zakończenia C_{ij} są przy tym określone jednoznacznie $C_{ij} = S_{ij} + p_{ij}$. Ograniczenia (12.8)–(12.9) mają dość oczywistą, wygodną interpretację grafową, patrz na przykład praca [269]. Dla permutacji π tworzymy graf $G(\pi) = (M \times N, E)$, gdzie $M \times N = \{1, \ldots, m\} \times \{1, \ldots, n\}$ jest zbiorem obciążonych wierzchołków reprezentujących operacje; obciążenie wierzchołka (i, j) jest równe $p_{i\pi(j)}$. Zbiór $E = E^0 \cup E^*$ zawiera *technologiczne* krawędzie pionowe

$$E^0 = \bigcup_{j=1}^{n} \bigcup_{i=1}^{m-1} \{((i, j), (i + 1, j))\} \tag{12.10}$$

oraz *sekwencyjne* krawędzie poziome

$$E^* = \bigcup_{i=1}^{m} \bigcup_{j=1}^{n-1} \{((i, j), (i, j + 1))\}. \tag{12.11}$$

Wszystkie krawędzie mają wagę zero. Graf $G(\pi)$ może być traktowany jako regularna prostokątna siatka $m \times n$, w której tylko obciążenia węzłów zależą od π. Wzór (12.4) wyraża długość najdłuższej drogi w $G(\pi)$ prowadzącej z węzła (1,1) do węzła (i, j), włącznie z obciążeniem tego węzła. Odpowiednio

$C_{\max}(\pi)$ jest długością najdłuższej drogi (drogi krytycznej) w $G(\pi)$. Wzór (12.3) odpowiada rekurencyjnej wersji obliczania długości tych dróg. Wartość $S_{i\pi(j)}$ jest długością najdłuższej drogi dochodzącej do węzła (i,j), bez obciążenia węzła (i,j). Każda ścieżka z $(1,1)$ do (m,n) może być reprezentowana ciągiem $(j_0, j_1, \ldots, j_m)$, patrz także wzór (12.4). Ciąg ten określa fragmenty odpowiadające ścieżkom poziomym $(i, j_{i-1}), (i, j_{i-1}+1), \ldots, (i, j_i)$, dla $i = 1, \ldots, m$, oraz krawędziom pionowym $((i, j_i), (i+1, j_i))$, dla $i = 1, \ldots, m-1$. Niech $u = (u_0, u_1, \ldots, u_m)$, gdzie $u_0 = 1$ i $u_m = n$, będzie ciągiem, który definiuje ścieżkę krytyczną w $G(\pi)$, to znaczy ścieżka krytyczna jest postaci

$$(1, u_0), \ldots, (1, u_1), (2, u_1), \ldots, (2, u_2), \ldots, (m, u_{m-1}), \ldots, (m, u_m).$$

Ciąg u jest tym, który maksyalizuje prawą stronę wzoru (12.4) dla $j = n$. Dla każdej maszyny i, podścieżka pozioma $(i, u_{i-1}), (i, u_{i-1}+1), \ldots, (i, u_i)$, odpowiada ciągowi zadań $\pi(u_{i-1}), \pi(u_{i-1}+1), \ldots, \pi(u_i)$ wykonywanych kolejno na tej maszynie, $i = 1, \ldots, m$. Ciąg ten będziemy nazywać *blokiem* [120] i będziemy oznaczali B_i. Liczba zadań w B_i jest $b_i = u_i - u_{i-1} + 1$, $i = 1, \ldots, m$. Z definicji u mamy $\sum_{i=1}^{m} b_i = n + k - 1 \leqslant n + m - 1$. Zadanie ostatnie w B_i jest równocześnie pierwszym w B_{i+1}, $i = 1, \ldots, m-1$. Formalnie, u zależy od π i B_i zależy od u, jednakże, dla dla prostoty notacji, nie będziemy tego uwzględniać bezpośrednio w oznaczeniach. Dodatkowo, dla uproszczenia zapisu, *zbiór* zadań należących do bloku B_i będziemy także oznaczać symbolem B_i. Dla bloku B_i, jego blok wewnętrzny definujemy jako $B_i^* = B_i \setminus \{\pi(u_{i-1}), \pi(u_i\}$.

Pojęcie bloku jest powszechnie używane w kontekście tak zwanych *własności blokowych* i/lub *eliminacyjnych własności blokowych*. Pierwotne sformułowanie własności blokowych [120,121] określa warunki konieczne istnienia rozwiazania lepszego, w sensie kryterium $C_{\max}$ (odpowiednio $L_{\max}$, $f_{\max}$) od pewnego danego rozwiązania. Bardziej dokładnie, niech π będzie pewną permutacją ze ścieżką krytyczną u oraz blokami B_i, $i = 1, \ldots, m$. Jeśli $C_{\max}(\sigma) < C_{\max}(\pi)$ to w σ co najmniej jedno zadanie z pewnego bloku wewnętrznego B_i^* poprzedza zadanie $\pi(u_{i-1})$ lub występuje za zadaniem $\pi(u_i)$. Własność blokowa w podanej postaci jest trudna do bezpośredniej implementacji, stąd w praktyce zdecydowanie częściej używana jest jej forma zanegowana, wykorzystywana wprost do pomijania rozwiązań, nazywana dalej eliminacyjną własnością blokową. Jeśli permutacja σ została otrzymana z π poprzez zmianę kolejności (przemieszanie) zadań należących do B_i^* dla dowolnego i, to zachodzi $C_{\max}(\sigma) \geqslant C_{\max}(\pi)$. Dowód tego faktu opiera się na spostrzeżeniu, że podana modyfikacja prowadzaca z π do σ, zachowuje w σ ścieżkę krytyczną z π. Dla podstawowego problemu $F^*||C_{\max}$ możliwe jest

uzyskanie mocniejszych własności eliminacyjnych dla przypadków szczególnych, to jest gdy $i \in \{1, m\}$. Jeśli $i = 1$ to przemieszanie zadań z $B_1^* \cup \{u_0\}$ nie poprawia wartości $C_{\max}$. Symetrycznie, jeśli $i = m$ to przemieszanie zadań z B_m^* nie poprawia wartości $C_{\max}$.

Blokowa własność eliminacyjna ma w zasadzie charakter *destrukcyjny*, bowiem określa, które rozwiązania należy odrzucić. Niestety nie ma charakteru it konstrukcyjnego, bowiem nie precyzuje *jak* otrzymać lepsze rozwiązanie. Te ostatnie są zwykle poszukiwane w zbiorze dopełniającym te pierwsze. Przykładowo, zgodnie z tą zasadaą, zadanie $j \in B_i^*$ powinno być przeniesione tak by poprzedzało zadanie $\pi(u_{i-1})$ lub występowało za zadaniem $\pi(u_i)$, jednak nie wiadomo dokłądnie gdzie je umieścić by otrrzymać poprawę wartości funkcji celu. Jak dotychczas, bolkowa własność eliminacyjna zostałą wykorzystana w algorytmach opartych na schemacie B&B (implikując bardzo charakterystyczną metodę podziału), w algorytmach TS, GA oraz w tak zwanych akceleratorach dla metod RACS, RAES, NEH oraz technik DS, dostarczając w każdym z tych przypadkó niewątpliwych korzyści.

12.4 Schematy B&B

Mimo, iż przeprowadzono wiele badań algorytmów B&B dla permutacyjnego problemu $F^*||C_{\max}$ oraz problemów pokrewnych $F^*|r_j|L_{\max}$ i $F^*||f_{\max}$, wyniki trudno uznać za zadowalające. O ile z sukcesem udało się rozwiązać dokładnie przykłady o rozmiarze $n \times m$ wielkości 50×3, to trudności napotkano już w przykładach 20×5 oraz innych posiadających większą liczbę maszyn. Powszechnie znane przykłady testowe 50×20 rozwiązywane tym podejściem, potrzebowały kilku tygodni pracy komputera o szybkości 100Mips, często bez sukcesu. Z tego względu, schemat B&B jest traktowany raczej jako kosztowne narzędzie do generowania dobrych rozwiązań referencyjnych wykorzystywanych przy ocenach algorytmów przybliżonych, niż jako konkurencyjna metoda rozwiązywania. Powyższa uwaga nie odnosi się jednak do problemów z wytwarzaniem cyklicznym.

Drzewo rozwiązań

Podstawowymi elementami występującymi w schematach B&B dla problemu przepływowego są: konstrukcja drzewa rozwiązań, dolne i górne ograniczenie oraz dodatkowe kryteria eliminacji. Reguła zamykania jest typowa, oparta na relacji dolnego i górnego ograniczenia. Najczęściej używany jest elegancki schemat budowy drzewa permutacji w oparciu o uszeregowania częściowe, tworzone począwszy od początku i/lub końca, analogicznie do

opisanego w Rozdz. 10.4. Znane są także schematy oparte na relacji poprzedzań pomiędzy zadaniami, wprowadzające podział zbioru rozwiązań poprzez stopniowe powiększanie tej relacji, jak na przykład przez podział dychotomiczny "$i \to j$ oraz "$j \to i$" dla pewnej wybranej pary zadań $i, j \in \mathcal{J}$. Najkorzystniejsze obliczeniowo, ale stosunkowo mało popularne ze względu na skomplikowaną logikę, schematy tego ostatniego typu wykorzystują eliminacyjne własności bloków zadań. Podział jest wprowadzany poprzez relację poprzedzania "$K \to j$" oraz "$j \to K$", gdzie K jest pewnym podzbiorem zadań zaś j pewnym zadaniem z tego bloku [120]. Niestety wymagają one pewnego zaawansowania w technikach implementacji programowych zaś zyski z ich stosowania i tak nie zmieniają w sposób znaczący ostatecznej oceny przydatności schematów B&B.

Kryteria eliminacji

Efektywność metod B&B zależy znacząco od liczby oraz wielkości problemów częściowych przeglądniętych pośrednio czyli faktycznie pominiętych. Podstawowymi narzędziami tego typu redukcji są *kryteria eliminacyjne* oraz *dolne/górne ograniczenia*. Kryteria eliminacji, zwane także *warunkami dominacji*, określają warunki dostateczne dla zadań, przy spełnieniu których potrafimy zawęzić lub wyeliminować pewien podzbiór rozwiązań dopuszczalnych nie tracąc przy tym rozwiązania optymalnego. Czasami kryteria te dzieli się na *globalne* (działające na całym zbiorze rozwiazań) oraz lokalne (odnoszące się do specyficznego podzbioru rozwiązań, zwykle skojarzonego z konkretnym węzłem drzewa rozwiązań).

Jak do tej pory, dla problemu $F^*||C_{\max}$ sformułowane w literaturze kilka klas kryteriów eliminacji, z których niektóre okazały się fałszywe zaś inne równoważne lub zawierająace się, patrz przegląd w pracy [224]. Niech $\mathcal{I}', \mathcal{I}'' \subset \mathcal{J}$ będą pewnymi podzbiorami zadań zaś σ' i σ'' permutacjami częściowymi określonymi na tych zbiorach odpowiednio. Dalej, niech $C(\tau, k)$ oznacza termin zakończenia wykonywania wszystkich zadań w permutacji częściowej τ na maszynie k. Mówimy, że σ'' dominuje σ' jeśli dla każdego dopełnienia $\sigma'\overline{\sigma}'$ istnieje dopełnienie $\sigma''\overline{\sigma}''$ takie, że $C(\sigma''\overline{\sigma}'', m) \leqslant C(\sigma'\overline{\sigma}', m)$. Historycznie najstarszym i jednocześnie podstawowym wynikiem w dominacji jest własność sformułowana dla $\mathcal{I}' = \mathcal{I}''$. Jeśli $C(\sigma'', k) \leqslant C(\sigma', k)$ dla $k = 1, \ldots, m$ to σ'' dominuje σ'. Kryterium to jest najsilniejszym możliwym dla $\mathcal{I}' = \mathcal{I}''$ w tym sensie, że jeśli $C(\sigma'', k) > C(\sigma', k)$ dla pewnego k, to istnieje przykład problemu taki, że $C(\sigma''\overline{\sigma}'', m) > C(\sigma'\overline{\sigma}', m)$ dla każdego dopełnienia. Kolejne kryteria sformułowano dla $\mathcal{I}' \cup \{j\} = \mathcal{I}''$, aby pokazać dominację $\sigma'' = \sigma ji$ nad $\sigma' = \sigma i$. Dla wygody zapisu wprowadzimy

oznaczenie $\Delta_k = C(\sigma ji, k) - C(\sigma i, k)$.

(a) $C(\sigma ji, k) \leqslant C(\sigma ij, k), \quad k = 2, \ldots, m$

(b) $\Delta_{k-1} \leqslant p_{kj}, \quad k = 2, \ldots, m$

(c) $\Delta_{k-1} \leqslant p_{kj}$ and $C(\sigma j, k-1) \leqslant C(\sigma i, k-1), \quad k = 2, \ldots, m$

(d) $\Delta_k \leqslant p_{kj}, \quad k = 2, \ldots, m$

(e) $\max\{\Delta_{k-1}, \Delta_k\} \leqslant p_{kj}, \quad k = 2, \ldots, m$

(f) $\Delta_{k-1} \leqslant \Delta_k \leqslant p_{kj}, \quad k = 2, \ldots, m$

(g) $\max_{1 \leqslant l \leqslant k} \Delta_l \leqslant p_{kj}, \quad k = 2, \ldots, m$

(h) $\Delta_k \leqslant \min_{k \leqslant l \leqslant m} p_{lj}, \quad k = 2, \ldots, m.$

Pokazano, że kryteria (a), (b), (d) są nieprawdziwe. W pracy [224] udowodniono, że kryterium (c) implikuje kryterium (e). Tam też podano referencje do znanych wyników literaturowych wykazujących równoważność kryteriów (e), (f), (g), (h). Pokazano także, że kryteria (e)-(h) są najsilniejszymi możliwymi w wyżej wspomnianym sensie.

Aby sprawdzić zachodzenie kryteriów należy rozpocząć od sprawdzenia warunku (f) dla zadania j. Pociąga on nierówność $p_{1j} \leqslant p_{kj}$, $k = 2 \ldots, m$ żądaną dla zadania j. Warunek (f) należy sprawdzić dla każdej pary zadań i oraz j (spełniającego powyższą nierówność). Otrzymane w ten sposób poprzedzania zadań mogą zawierać cykle, które należy wyeliminować. Stąd, złożoność obliczeniowa badania reguł dominacji jest $O(mn^2)$ i jest porównywalny z kosztem wyznaczania wartości dolnego ograniczenia.

Wynik działania obu rodzajów kryteriów jest zwykle przedstawiany w postaci dodatkowej relacji poprzedzań $\mathcal{R}^e$, pozwalającej z kolei na redukcję wielkości drzewa rozwiązań. Niestety wraz ze wzrostem m częstość zachodzenia warunków maleje co powoduje, że już dla $m > 5$ lub $n > 15$ korzyść z ich stosowania jest wątpliwa.

Górne ograniczenia

Jakość metody B&B zależy w głównej mierze od dokładności budowy oszacowań wartości funkcji celu: górnego (UB) i dolnego (LB). Proces aktualizacji górnego ograniczenia najlepiej przeprowadzać w każdym węźle σ drzewa rozwiązań, reprezentującym zbiór rozwiązań $\Pi_{\mathcal{R}_\sigma}$, według zasady $UB := \min\{UB, C_{\max}(\omega)\}$, gdzie $\omega \in \Pi_{\mathcal{R}_\sigma}$ jest pewnym rozwiązaniem. W

praktyce do wyznaczania ω wykorzystuje się dowolny z algorytmów przybliżonych opisanych w rozdziale następnym, zaadoptowany uprzednio do pracy z relacją $\mathcal{R}_\sigma$.

Dolne ograniczenia

Dolne ograniczenia mogą być budowane przy użyciu wielu różnorodnych relaksacji i są kluczowe dla sukcesu algorytmu B&B. Problemy przepływowe ze względu na swą regularność stanowią klasę zagadnień umożliwiającą wprowadzenie eleganckiego, ogólnego schematu wyznaczania dolnych ograniczeń generującego szereg konkretnych metod. Schemat ten wykorzystuje relaksację przepustowości maszyn prowadząc do jedno- i dwumaszynowych problemów z algorytmami wielomianowymi, patrz np. [121]. Punktem wyjścia do rozważań jest zwykle problem $F^*|prec|C_{\max}$ z relacją poprzedzan $\mathcal{R}_\sigma$ lub $\mathcal{R}^e$ charakterystyczną dla rozpatrywanego węzła w drzewie rozwiązań, wynikającej z przyjetej reguły podziału. W przypadku metody podziału opartej na uszeregowaniu częściowym tworzonym od początku, dolne ograniczenie wynika z odpowiedniej superpozycji fragmentu ustalonego σ oraz dolnego ograniczenia dla zadań jeszcze nieuszeregowanych. Poniżej podano ogólny schemat budowy ograniczeń przy założeniu $\mathcal{R} = \emptyset$. Przypadek $\mathcal{R} \neq \emptyset$ można otrzymać implementując najpierw relację poprzez wprowadzenie terminów gotowości oraz czasów dostarczania dla zadań (operacji), podobnie jak w Rozdz. 9.3, mających sens czasów wykonywania na odpowiednich maszynach o nieograniczonej przepustowości. Maszyny tego ostatniego typu są osadzone w sposób naturalny w opisywanym schemacie dolnych ograniczeń.

Załóżmy, że wszystkie maszyny z wyjątkiem, co najwyżej dwóch u oraz v, $1 \leqslant u \leqslant v \leqslant m$ mają nieograniczoną przepustowość. Ponieważ podzbiór kolejnych maszyn o nieograniczonej przepustowości możemy zastąpić jedną maszyną o takiej własności, otrzymamy w wyniku relaksacji problem przepływowy zawierający co najwyżej 5 maszyn w tym 2 o ograniczonej przepustowości. Niech $\star u$, u, uv, v, $v*$ oznaczają te maszyny. Czasy wykonywania zadań na maszynach u oraz v pozostają niezmienione, zaś pozostałe określamy następująco

$$p_{\star u} = \sum_{i=1}^{u-1} p_{ij} \tag{12.12}$$

$$p_{uv} = \sum_{i=u+1}^{v-1} p_{ij} \tag{12.13}$$

Rysunek 12.2: Graf dominacji dolnych ograniczeń

$$p_{u\star} = \sum_{i=u+1}^{m} p_{ij}. \tag{12.14}$$

Jeśli $u = v$ to problem ma co najwyżej 3 maszyny w tym tylko jedną o ograniczonej przepustowości. Wartość $p_{\star u}$ może być interpretowana jako termin gotowości r_{uj} zadania (w odniesieniu do maszyny u), zaś wartość $p_{v\star}$ jako czas dostarczenia q_{vj} zadania (w odniesieniu do maszyny v). Zastosowanie relacji $\mathcal{R}$ może wprowadzać wartości r_{uj} oraz q_{vj} inne niż wynikające tylko z wzorów (12.12), (12.14), a szacujące dokładniej odpowiednie wartości. Schemat postępowania jest podobny do opisanego w Rozdz. 9, przy czym możliwe jest zastosowanie progresji wynikającej ze zwiększonej liczby maszyn. W najprostszym przypadku r_{uj} może być interpretowane jako najdłuższa droga w grafie, zawierającym relację $\mathcal{R}$ zadań przeniesioną na operacje, dochodząca do węzła (u, j) (bez obciążenia tego węzła), zaś q_{vj} – najdłuższa droga wychodząca z węzła (v, j) (także bez obciążenia tego węzła). Ponieważ tak sformułowane zagadnienie jest NP-trudne, potrzebne sa dalsze relaksacje. Dowolna maszyna $x \in \{\star u, uv, v\star\}$ o nieograniczonej przepustowości może być wyeliminowana z problemu poprzez relaksację czasów trwania operacji na niej wykonywanych. Przykładowo możemy przyjać czas wszystkich operacji na niej wykonywanych równy

$$p_x = \min_{1 \leqslant j \leqslant n} p_{xj},\ x \in \{\star u, uv, v\star\}. \tag{12.15}$$

Wszystkie metody wyznaczania dolnych ograniczeń moga być przedstawione przy użyciu ciągu symboli ze zbioru $\{\square, \triangle, \star\}$, gdzie $\square$ oznacza maszynę o ograniczonej przepustowości, $\triangle$ oznacza maszynę o nieograniczonej przepustowości, $\star$ oznacza wyeliminowaną maszynę o nieograniczonej przepustowości. Posługując się wprowadzonymi pojęciami możliwe jest skonstruowanie wielu problemów, z których każdy stanowi dolne ograniczenie dla $F^*||C_{\max}$. W celu określenia powiązań pomiędzy tak otrzymanymi ograniczeniami wygodnie jest posłużyć się grafem dominacji, Rys. 12.2. Rozpocznijmy od najsłabszych dolnych ograniczeń reprezentowanych przez ciągi trzyelementowe. Ciągi $(\star\square\star)$, $(\triangle\square\star)$, $(\star\square\triangle)$ odpowiadają problemom $1|r_j = r_*, q_j = q_*|C_{\max}$, $1|r_j, q_j = q_*|C_{\max}$, $1|r_j = r_*, q_j|C_{\max}$, odpowiednio, o wielomianowej złożoności obliczeniowej, patrz Rozdz. 9. Problem $(\triangle\square\triangle)$ odpowiada NP-trudnemu problemowi $1|r_j, q_j|C_{\max}$, dla którego zagadnienie wyznaczania efektywnego dolnego ograniczenia było dyskutowane także w Rozdz.

9. Spośród ciągów 5-cio elementowych, ciąg $(\star\square\star\square\star)$ odpowiada problemowi $F^*2|r_j = r_*, t_j = t_*, q_j = q_*|C_{\max}$, gdzie t_j jest czasem transportu pomiędzy maszyną 1 i 2, który ma taką sama kolejność wykonywania jak problem $F^*2||C_{\max}$. Ostatni wielomianowy problem jest związany z ciągiem $(\star\square\triangle\square\star)$, odpowiadajacym problemowi $F^*3|M_2-\text{non bottl.}|C_{\max}$, dla którego algorytm podano w Rozdz. 12.2. Wszystkie pozostałe ciągi odpowiadają problemom NP-trudnym.

12.5 Podstawowe algorytmy przybliżone

Dość duża liczba algorytmów przybliżonych została sformułowana dla problemu $F^*||C_{\max}$, patrz np. przegląd w [344], mimo iż w klasycznej literaturze książkowej wymienia się zaledwie kilka. Bazują one na kilku typowych podejściach (lub ich kombinacjach), a mianowicie: (a) reguły priorytetowe statyczne, (b) reguły priorytetowe dynamiczne, (c) transformacja do prostszych problemów posiadających algorytm wielomianowy, (d) technika wcięć.

Priorytety statyczne

Proste statyczne reguły priorytetowe generują permutację zadań poprzez ich uporządkowanie według niemalejących wartości priorytetu. Odpowiednie funkcje priorytetu dane są wzorami [146,278]

P : $\omega_j = \sum_{i=1}^{m}(m-2i+1)p_{ij}$,

G : $\beta_j = A_j / \min_{1\leqslant i\leqslant m-1}(p_{ij}+p_{i+1,j})$, gdzie $A_j = -1$ jeśli $p_{1j} \leqslant p_{mj}$ oraz $A_j = 1$ w przeciwnym przypadku.

Niektóre nowo proponowane algorytmy poprawiają niedostatki metod zaprojektowanych wcześniej. Przykładowo, Algorytm P ignoruje maszynę $\lceil m/2 \rceil$ jeśli m jest nieparzyste. Zatem Algorytm HR [173] (pochodny od P) zwraca π^P dla m parzystego, zaś dla m nieparzystego zwraca najlepszą spośród permutacji π^P, π', π'', gdzie π' i π'' są generowane przy użyciu zmodyfikowanych funkcji priorytetu $\omega'_j = \sum_{i=1}^{m}(m-2i)p_{ij}$ oraz $\omega''_j = \sum_{i=1}^{m}(m-2i+2)p_{ij}$, odpowiednio.

Funkcje priorytetu mogą pochodzić także z reguł dominacji dostarczających warunków dostatecznych optymalności dla szczególnych przypadków problemów przepływowych. Przykładowa funkcja priorytetu z pracy [149] zależy od pary maszyn k, l $(1 \leqslant k < l \leqslant m)$ oraz indeksu zadania $j \in \mathcal{J}$

$$f_j(k,l) = \frac{\text{sign}\{\alpha_{\text{j}}(\text{k},\text{l}) - \beta_{\text{j}}(\text{k},\text{l})\}}{\min\{\alpha_j(k,l), \beta_j(k,l)\}} \tag{12.16}$$

gdzie

$$\alpha_j(k,l) = \sum_{s=k}^{l-1} p_{sj}, \quad \beta_j(k,l) = \sum_{s=k+1}^{l} p_{sj}. \tag{12.17}$$

Ustalając parę k, l, otrzymujemy zwykły algorytm priorytetowy z priorytetem statycznym $f_j(k,l)$. W pracy [149] zaproponowano następujące oczywiste algorytmy szczególne

F : Znajdź π^F spełniające $C_{\max}(\pi^F) = \min_{1 \leqslant k < m} C_{\max}(\pi^{km})$.

B : Znajdź π^B spełniające $C_{\max}(\pi^B) = \min_{1 < l \leqslant m} C_{\max}(\pi^{1l})$.

T : Znajdź π^T spełniające $C_{\max}(\pi^T) = \min_{1 \leqslant k < l \leqslant m} C_{\max}(\pi^{kl})$.

gdzie π^{kl} jest permutacją otrzymaną przez uporządkowanie zadań według niemalejących wartości $f_j(k,l)$. Złożoność obliczeniowa algorytmów F oraz B jest $O(nm \log n)$, zaś T jest $O(m^2 n \log n)$. Algorytmy z priorytetem (12.16) odwołują się niejawnie do problemu $F^*2||C_{\max}$, patrz priorytetowa wersja algorytmu Johnsona. Istotnie π^{kl} jest optymalnym rozwiązaniem problemu przepływowego, w którym maszyny k i l posiadają ograniczoną (jednostkową) przepustowość, maszyny $k+1, \ldots, l-1$ posiadają nieograniczoną przepustowość, maszyny $1, \ldots, k-1, l+1, \ldots, m$ zostały zredukowane (wyeliminowane). Problem ten jest sprowadzalny do wielomianowego zagadnienia $F^*3|M_2 - non - bottl|C_{\max}$.

Priorytety dynamiczne

Przykładem algorytmów z priorytetem dynamicznym są MINIT, MICOT i MINIMAX z pracy [149]. Algorytm MINIT tworzy permutację częściową σ startując z wybranej dwuelementowej permutacji $\sigma = (a,b)$. Następnie σ jest krokowo rozszerzana o zadanie c, które zapewnia minimalny czas przestoju permutacji rozszerzonej σc, spośród wszystkich możliwych nieuszeregowanych jeszcze zadań $c \in U$.

MINIT: (0) Wyznacz permutację początkową $\sigma = (a,b)$ poprzez przeszukanie par zadań $a, b \in \mathcal{J}$ w celu znalezienia tej z minimalnym czasem przestoju $I_m(\sigma)$; podstaw $U = J \setminus \{a,b\}$. (1) Repeat STEP until $|U| > 2$. (2) STEP: Znajdź zadanie c takie, że $I_m(\sigma c) = \min_{s \in U} I_m(\sigma s)$; podstaw $\sigma := \sigma c$, $U := U \setminus \{c\}$. (3) Sprawdź dwie kompletne permutacje $\pi' = \sigma ab$ oraz $\pi'' = \sigma ba$, gdzie a, b są dwoma pozostałymi jeszcze zadaniami w U, oraz wybierz tą lepszą $\pi^{MINIT} = \min_{\pi \in \{\pi', \pi''\}} C_{\max}(\pi)$.

Algorytm MICOT został zaprojektowany przez analogię do MINIT. Buduje on częściową permutację σ rozpoczynając z permutacji dwuelementowej $\sigma = (a, b)$. Rozszerzanie σ następuje poprzez wybór zadania c, które zapewnia minimalny termin zakończenia permutacji σc spośród wszystkich niuszeregowanych jeszcze zadań $c \in U$.

MICOT: (0) Znajdź permutację początkową $\sigma = (a, b)$ przez przeszukanie par zadań $a, b \in J$ w celu znalezienia tej z najmniejszą długością uszeregowania; podstaw $U = J \setminus \{a, b\}$. (1) Repeat STEP until $|U| > 2$. (2) STEP: Znajdź zadanie c takie, że $C_{\max}(\sigma c) = \min_{s \in U} C_{\max}(\sigma s)$; podstaw $\sigma := \sigma c$, $U := U \setminus \{c\}$. (3) Sprawdź dwie kompletne permutacje $\pi' = \sigma ab$ i $\pi'' = \sigma ba$, gdzie a, b są dwoma pozostałymi zadaniami w U, oraz wybierz lepszą z nich $\pi^{MICOT} = \min_{\pi \in \{\pi', \pi''\}} C_{\max}(\pi)$.

Algorytm MINIMAX buduje permutację z obu końców równocześnie używając pierwotnej idei algorytmu Johnsona (dla problemu $F^*2||C_{\max}$) rozszerzonej do przypadku m-maszynowego. Zadania mające małe czasy wykonywania są preferowane jako początkowe, zaś te z dużymi czasami wykonywania – jako końcowe w permutacji.

MINIMAX: Podstaw $U = J$, $\pi = ()$, $k = 0$, $l = n + 1$. Repeat STEP 1 oraz 2 w naprzemiennej kolejności until $k \geqslant l$. STEP 1: Znajdź zadanie a takie, że $p_{ia} = \min_{1 \leqslant s \leqslant m} \min_{j \in U} p_{sj}$ i podstaw $k := k + 1$, $\pi(k) = a$, $U := U \setminus \{a\}$. STEP 2: Znajdź zadanie b takie, że $p_{ib} = \max_{1 \leqslant s \leqslant m} \max_{j \in U} p_{sj}$ i podstaw $l := l - 1$, $\pi(l) = b$, $U := U \setminus \{b\}$.

Algorytmy priorytetowe powszechnie uważane są za bardzo szybkie, lecz mało efektywne. Opinie te potwierdzają zarówno badania eksperymentalne [365], jak i słabe oceny uzyskiwane w teoretycznej analizie najgorszego przypadku 265, patrz Tabela 12.1.

Zastosowanie problemu $F^*2||C_{\max}$.

Istnieje wiele algorytmów przybliżonych używających problemu $F^*2||C_{\max}$ jako pomocniczego [48, 72, 309]. Różnią się one techniką transformacji oraz liczbą generowanych problemów pomocniczych.

RS : Znajdź permutację, która jest optymalnym rozwiązaniem pomocniczego problemu dwu-maszynowego $F^*2||C_{\max}$ z czasami wykonywania zadań $a_j = \sum_{i=1}^{k} p_{ij}, b_j = \sum_{i=k+1}^{m} p_{ij}$, na maszynach 1 i 2, odpowiednio, gdzie $k = \lfloor m/2 \rfloor$.

RA : Znajdź permutację, która jest optymalnym rozwiązaniem pomocniczego problemu dwu-maszynowego $F^*2||C_{\max}$ z czasami wykonywania zadań $a_j = \sum_{i=1}^{m}(m-i+1)p_{ij}, b_j = \sum_{i=1}^{m} ip_{ij}$, na maszynach 1 i 2, odpowiednio.

CDS: Znajdź π^{CDS} taką, że $C_{\max}(\pi^{CDS}) = \min_{\pi\in\{\pi^1,\ldots,\pi^{m-1}\}} C_{\max}(\pi)$, gdzie π^k jest optymalną permutacją dla problemu $F^*2||C_{\max}$ z czasmi wykonywania $a_j^k = \sum_{i=1}^{i} p_{ij}$, $b_j^k = \sum_{i=m-k+1}^{m} p_{ij}$, $j = 1,\ldots,n$, na maszynach 1 i 2, odpowiednio, $k = 1,\ldots,m-1$.

Algorytmy RS i RA mają złożoność obliczeniową $O(nlogn + nm)$ zaś Algorytm CDS złożoność obliczeniową $(nmlogn + nm^2)$. Możliwe są także odmienne sposoby transformacji prowadzące do algorytmów o podobnych ocenach [344] teoretycznych

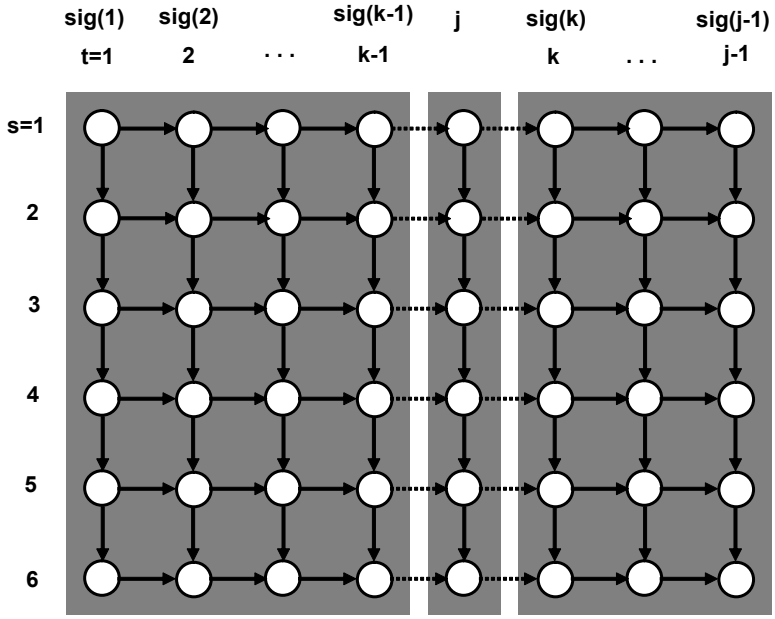
Q/X: Znajdź $\pi^{Q/X}$ taką, że $C_{\max}(\pi^{Q/X}) = \min_{\pi\in\{\pi^1,\ldots,\pi^{m-1}\}} C_{\max}(\pi)$, gdzie π^k jest optymalną permutacją dla problemu $F^*2||C_{\max}$ z czasami wykonywania $a_j^k = \sum_{i=1}^{k} p_{ij}$, $b_j^k = \sum_{i=m-k+1}^{m} p_{ij}$, $j = 1,\ldots,n$, na maszynach 1 i 2, odpowiednio, $k = 1,\ldots,m-1$.

T/Yk Wyznacz $\pi^{T/Yk}$ jako przybliżoną permutację dla problemu k-maszynowego $Fk||C_{\max}$ z czasami wykonywania $p'_{ij} = \sum_{s=l_{i-1}+1}^{l_i} p_{sj}$, gdzie $l_i = \sum_{s=1}^{i} m_s$, $i = 1,\ldots,k$, $l_0 = 0$ oraz $(m_1,\ldots,m_k)$ jest ciągiem takim, że $m_i > 0$, $i = 1,\ldots,k$, $\sum_{s=1}^{k} = m$.

Algorytm T/Y2 jest równoważny Q/X, jednakże dla $k \geqslant 3$ może stanowić konkurencyjną alternatywę dla innych metod, o ile tylko dostępny będzie odpowiednio dobry algorytm przybliżony dla problemu z ustaloną liczbą maszyn. Priorytetowe algorytmy F, B oraz T opisane poprzednio mogą być także traktowane jako algorytmy tej klasy. Opisana klasa algorytmów jest wyraźnie lepsza od klasy algorytmów priorytetowych zarówno w analizie eksperymentalnej jak i teoretycznej, Tabela 12.1. Algorytm CDS jest powszechnie uważany za najlepszy w tej klasie.

Technika wcięć

Algorytm NEH [252], oparty na technice wcięć, pozostaje do chwili obecnej bezspornym championem wśród konstrukcyjnych algorytmów przybliżonych dla problemu $F^*||C_{\max}$. Składa się on z n-krokowej fazy zasadniczej poprzedzonej fazą wstępną. W fazie wstępnej zadania są przenumerowywane zgodnie z nierosnącymi wartościami $\sum_{i=1}^{m} p_{ij}$ (zadanie pierwsze ma



Rysunek 12.3: Ilustracja "szybkiego" wcięcia

największą wartość sumy). W fazie zasadniczej generowany jest ciąg permutacji $\pi_{I_1}, \pi_{I_2}, \ldots, \pi_{I_n}$, gdzie $I_j = \{1, \ldots, j\}$. Permutacja $\pi_{I_n} = \pi^{NEH}$ jest tą generowaną przez Algorytm NEH. W j-tym kroku tej fazy zadanie j-te jest wstawiane bezpośrednio przed $\pi_{I_{j-1}}(1)$ oraz bezpośrednio za zadania $\pi_{I_{j-1}}(s)$, $s = 1, \ldots, j-1$ dostarczając j nowych permutacji na zbiorze I_j. Dla każdej permutacji obliczana jest wartość funkcji celu, zaś permutacja dostarczająca minimum tej wartości jest przyjmowana jako π_{I_j}.

Dobrą intuicyjną analogią dla Algorytmu NEH jest proces pakowania torby turystycznej elementami o różnych rozmiarach i kształtach. Typowo zaczyna się od najkorzystniejszego umieszczenia jedneo lub kilku największych elementów (badajć ich wzajemne dopasowanie), po czym krokowo "dopycha się" aktualnie istniejący stan torby elemntami coraz mniejszymi dopasowujac je metodą prób.

Trywialna implementacja Algorytmu NEH ma złożoność obliczeniową rzędu $O(n^3m)$. Bardziej zaawansowana implementacja o złożoności $O(n^2m)$ dla problemu $F^*||C_{\max}$ została opisana w pracy [357]. Niech $\sigma = (\sigma(1), \ldots, \sigma(j-1))$ będzie permutacją częściową zaś j niech będzie zadaniem wstawianym na pozycję $k = 0, 1, \ldots, j-1$ ($k = 0$ oznacza przed zadanie $\sigma(1)$). Dla σ

obliczamy wpierw wartości

$$r_{st} = \max\{r_{s-1,t}, r_{s,t-1} + p_{s\sigma(t)}\},\ t = 1, \ldots, j-1,\ s = 1, \ldots, m \quad (12.18)$$

$$q_{st} = \max\{q_{s+1,t}, q_{s,t+1} + p_{s\sigma(t)}\},\ t = j-1, \ldots, 1,\ s = m, \ldots, 1 \quad (12.19)$$

gdzie $r_{0t} = 0 = q_{jt}$, $t = 1, \ldots, n$, $r_{s0} = 0 = q_{s,m+1}$, $s = 1, \ldots, m$. Wartość r_{st} jest długością najdłuższej drogi w grafie $G(\sigma)$ dochodzącej do wierzchołka (s,t), wraz z obciążeniem tego wierzchołka. Graf siatkowy $G(\sigma)$ konstruowany jest analogicznie do opisu z Rozdz. 12.3, to znaczy ma postać $G(\sigma) = (M \times I_{j-1}, F^0 \cup F^*)$, gdzie $I_{j-1} = \{1, \ldots, j-1\}$ zawiera numery zadań umieszczonych w σ, $F^0 = \bigcup_{s=1}^{m-1} \bigcup_{t=1}^{j-1} \{((s,t),(s+1,t))\}$ jest zbiorem łuków technologicznych "pionowych", zaś $F^* = \bigcup_{s=1}^{m} \bigcup_{t=1}^{j-2} \{((s,t),(s,t+1))\}$ zbiorem łuków sekwencyjnych "poziomych"; obciążenie węzła (s,t) jest równe $p_{s\sigma(t)}$. Podobnie q_{st} jest długością najdłuższej drogi w $G(\sigma)$ wychodzącej z węzła (s,t), wraz z obciążeniem tego węzła. Niech k będzie pewną ustaloną pozycją. W celu wyliczenia wartości kryterium dla permutacji σ^k otrzymanej z σ poprzez wstawienie zadania j na pozycję k wykonujemy wpierw obliczenie wartości

$$d_i = \max\{r_{i,k}, d_{i-1}\} + p_{ij},\ i = 1, \ldots, m, \quad (12.20)$$

po czym otrzymujemy

$$C_{\max}(\sigma^k) = \max_{1 \leqslant i \leqslant m} (d_i + q_{i,k+1}). \quad (12.21)$$

Ilustracja tego procesu przedstawiona jest w Rys. 12.3.

Dalsza redukcja czasu obliczeń Algorytmu NEH jest możliwa po wykorzystaniu własności blokowych [264]. Niech $u = (u_0, \ldots, u_m)$ będzie ścieżką krytyczną otrzymaną dla wcięcia zadania j na pozycję k w permutacji σ. Jeśli zadanie j należy do pewnego bloku wewnętrznego w σ, to następne wcięcie tego zadania nie powinno nastąpić wcześniej niż na ostatnią pozycję tego bloku. W przeciwnym przypadku wcięcie następuje klasycznie to znaczy na pozycję $k+1$. Pomysł ten umożliwia redukcję obliczeń o 25-50% w zależności od ilorazu n/m. Teoretyczne własności przyśpieszające pozwalają nam np. dla $n = 100$ osiągnać w eksperymentach komputerowych 75-100 -krotnie szybsze działanie algorytmu NEH. Dla porównania, technologia wytwarzania układów scalonych potrzebowała około 13 lat by osiągnąć 100-krotny wzrost mocy obliczeniowej procesorów. W praktyce, algorytm NEH dostarcza rozwiązań problemu $F^*||C_{\max}$ z błędem względnym przeciętnie 5–6% w odniesieniu do rozwiązania optymalnego. Zachowuje się

znacznie lepiej dla problemów z $m > n$ niż w sytuacji $m \ll n$. Dla problemu $F^*2||C_{\max}$ algorytm NEH nie musi dostarczyć rozwiązania optymalnego, odmiennie niż np. CDS, RS, które gwarantują uzyskanie tym przypadku rozwiązania optymalnego. Zaskakującym jest fakt, że rozwiązania generowane przez NEH charakteryzują się także względnie małą wartością kryterium $\sum C_i$ co wskazuje, że metoda wcięć charakteryzyje się dobrymi własnościami dla szerokiej klasy problemów. Przeniesienie idei wcięć jest możliwe zarówno do problemów przepływowych o bardziej złożonych ograniczeniach (patrz rozdział następny) jak i do bardziej złożonych problemów szeregowania (np. problemy gniazdowe), owocujac algorytmami o bardzo korzystnych cechach numerycznych.

Metody kombinowane

Pewną kombinację podejścia priorytetowego połączonej z relaksacją do problemów dwu-maszynowych zaproponowano w pracy [201] w formie algorytmu zwanego dalej HFC. Korzysta on domyślnie z faktu, że dla problemu $F^*3||C_{\max}$, jeśli permutacje otrzymane dla problemów $F^*2||C_{\max}$ zawierających tylko pary maszyn 1-2, 2-3, 1-3 są identyczne, to otrzymane rozwiązanie jest optymalne. Rozszerzenie tej własności na problem ogólny $F^*||C_{\max}$ prowadzi do pewnego algorytmu przybliżonego. Rozpatrywany jest ciąg zawierający $m(m-1)/2$ problemów zrelaksowanych $F^*2||C_{\max}$, z których każdy został otrzymany poprzez przyjęcie ograniczonej (jednostkowej) przepustowości dla maszyn k oraz l $(1 \leqslant k < l \leqslant m)$ oraz wyeliminowanie pozostałych maszyn. Rozwiązanie optymalne każdego z takich problemów oznaczmy przez π_{kl}, odpowiednio. Dla każdego zadania określany jest jego priorytet P_s w oparciu o wszystkie π_{kl}, $1 \leqslant k < l \leqslant m$. Startując z $P_s = 0$, $s = 1, \ldots, n$, dla każdej pary zadań $\pi_{kl}(i)$, $\pi_{kl}(j)$ takiej, że $1 \leqslant i < j \leqslant n$, wykonywane jest zmniejszenie $P_{\pi_{kl}(i)}$ o jeden oraz zwiększenie $P_{\pi_{kl}(j)}$ o jeden, $1 \leqslant k < l \leqslant m$. Ostatecznie, permutacja π^{HFC} jest otrzymywana poprzez uszeregowania zadań według niemalejacych wartości priorytetów P_s. Według sformułowania w pracy [201] Algorytm HFC posiada złożoność obliczeniową $O(n^2m^2)$ czyli większą niż efektywny wariant NEH. W rzeczywistości HFC jest mniej kosztowny niż NEH, co pokazano poniżej. Najbardziej czasochłonną częścią algorytmu jest rozwiązywanie problemów dwu-maszynowych; można to zrobić w czasie $O(nm^2 \log n)$. Oznaczmy przez π_{kl}^{-1} permutację odwrotną do π_{kl}; wszystkie permutacje odwrotne można znaleźć w czasie $O(nm^2)$. Dla zadania j umieszczonego na pozycji $\pi_{kl}^{-1}(j) = t$ wartość priorytetu zostanie $t-1$ razy zwiększona oraz $n-t$ razy zmniejszona. Stąd, analiza pojedynczej permutacji π_{kl} wnosi wkład do P_j równy $2\pi_{kl}^{-1}(j) - n - 1$ co ostatecznie

prowadzi do wartości priorytetu

$$P_j = \sum_{1 \leqslant k < l \leqslant m} (2\pi_{kl}^{-1}(j) - n - 1)$$

$$= 2m(m-1)[-\frac{n+1}{4} + \frac{1}{m(m-1)} \sum_{1 \leqslant k < l \leqslant m} \pi_{kl}^{-1}(j)]. \qquad (12.22)$$

Ponieważ zarówno skalowanie (liczbą dodatnią) jak i jednakowe przesunięcie nie wpływają na porządkowanie priorytetów, wynikowa permutacja odpowiada uporządkowaniu zadań według średniej wartości pozycji

$$\frac{1}{m(m-1)} \sum_{1 \leqslant k < l \leqslant m} \pi_{kl}^{-1}(j),$$

na których były one umieszczane w permutacjach π_{kl}. Ta dość oczywista intuicja potwierdza także ostateczną złożoność obliczeniową algorytmu HFC równą $O(nm^2 \log n)$, mniejszą od NEH. Zauważmy, że algorytm HFC dostarcza rozwiązań optymalnych zarówno dla problemu $F^*2||C_{\max}$ jak i niektórych szczególnych przypadków problemu $F^*3||C_{\max}$. Dodatkowo, badania eksperymentalne sugerują, że HFC konkurencyjny dla NEH pod względem czasu działania, może być także konkurencyjny pod względem jakości dostarczanych rozwiązań.

12.6 Algorytm kulturowy

Algorytmu HFC przedstawiony powyżej być także traktowany jako przypadek uproszczonego *podejścia kulturowego* wymienionego w Rozdz. 8. W takiej interpretacji pozycja zadania j jest ustalana biorąc pod uwagę *stanowiska* zespołu $m(m-1)/2$ niezależnych ekspertów, wyrażonych poprzez wskazanie pozycji dla tego zadania $\pi_{kl}^{-1}(j)$, traktowanej jako forma rankingu. Dobrą analogią działania algorytmu są oceny nadawane przez zespół niezależnych sędziów w zawodach sportowych. Zauważmy, że wynik końcowy jest *wypadkową* wszystkich ocen, czyli czymś kompletnie odmiennym niż zwykły przegląd $m(m-1)/2$ rozwiazań. Demokratyzm głosowania (pluralizm stanowisk), naturalny w wymienionym przypadku, może zostać zmieniony przez wprowadzenie wag różnicujących oceny nadawane przez arbitrów, odzwierciedlających niejednakowe doświadczenie sędziów. Pełny algorytm kulturowy powinien być wyposażony w mechanizm samoadaptacji wag, zrealizowany na przykład w technice GS, umożliwiający doskonalenie doświadczenia zespołu ekspertów.

Wybierając zbiór dowolnych innych heurystyk, spośród opisanych wcześniej, nadając im wagi różnicujące ich rangę oraz realizując proces "głosowania", otrzymamy przykład innego algorytmu kulturowego, nazywanego też czasami *parametryczną* heurystyką złożoną lub *przestrzenią* heurystyk. Poszczególne algorytmy z tego zbioru można otrzymać dobierając odpowiednio zero-jedynkowe wektory wag.

12.7 Algorytmy DS

W systemach wymagających rozwiązań bliskich optymalnym, w których metody B&B są zbyt kosztowne obliczeniowo, rutynowo proponowane są przybliżone algorytmy konstrukcyjne wspomagane dodatkowym modułem poprawy rozwiązania w oparciu o technikę szukania zstępującego (DS). Startując z najlepszego znanego algorytmu konstrukcyjnego możemy tym podejściem otrzymać rozwiązania problemu $F^*||C_{\max}$ z błędem poniżej 5%. Niestety, w przypadku ogólnym technika DS jest dość wrażliwa na dobór punktu startowego co oznacza, że ewentualna współpraca z innymi algorytmami konstrukcyjnymi nie musi gwarantować tak dobrej jakości rozwiązań. Sprawdzono eksperymentalnie, że dość popularna metoda polegająca na wielokrotnym startowaniu DS z rozwiązań przypadkowych jest zwykle zdecydowanie bardziej kosztowna i wolniej zbieżna w porównaniu do algorytmu zaprojektowanego w technikach TS, GA, zatem nie polecana.

Biorąc pod uwagę, że rozwiazanie problemu $F^*||C_{\max}$ może być reprezentowane permutacją, poprawianie polega na analizie serii rozwiązań otrzymanych przez nieznaczne zaburzenie permutacji, nazywanej dalej permutacją bazową, dostarczonej pierwotnie przez pewien algorytm konstrukcyjny. Zgodnie z zasadą DS, każda zaistniała poprawa może być źródłem dalszych popraw, poprzez wymianę permutacji bazowej na otrzymaną lepszą. W literaturze występuje kilka typowych sposobów tworzenia zaburzeń w permutacji, a mianowicie: wymiana par elementów przyległych w permutacji (API), wymiana par elementów dowolnych (NPI), przesunięcia elementów (INS), odwrócenia podciągu (INV), permutacje k wybranych elementów (k-PI). Technika NPI jest równoważna 2-PI. W celu zmniejszenia kosztu obliczeń w technice k-PI używane są niewielkie wartości k, $k = 2, 3, 4$. Wszystkie rozwiązania zaburzone osiągalne z danego rozwiązania bazowego π tworzą *lokalne otoczenie* (sąsiedztwo) π.

Chociaż API posiada poważne wady w zastosowaniu do problemów z kryterium $C_{\max}$, wciąż pozostaje jedną z najczęściej używanych technik ze względu na jej prostotę, głównie dla problemów innych niż $C_{\max}$. Dla proble-

mów z kryterium $C_{\max}$ polecane jest INS lub jej ulepszone warianty, które wykorzystują własności grafowe i blokowe do przyśpieszania obliczeń.

Odpowiednio do opisu algorytmu popraw, możliwe są dwa alternatywne warianty jego pracy: do pierwszej poprawy (pierwsze napotkane lepsze rozwiązanie zastępuje bazowe) oraz do najlepszej poprawy (najlepsze rozwiązanie w otoczeniu zastępuje bazowe jeśli tylko jest lepsze od aktualnego bazowego). Drugi z wymienionych jest stosowany częściej ze względu na lepszą zbieżność do dobrego rozwiązania, jednak jest bardziej kosztowny obliczeniowo. Redukcja tego kosztu jest możliwa poprzez zastosowanie tak zwanych *akceleratorów*.

Rozszerzenia znanych algorytmów

Dołączając do algorytmu RA z Rozdz. 12.5 kompletną procedurę API, sformułowano w literaturze dwa kolejne algorytmy [72]. RACS jest jednoprzebiegowy, do najlepszej poprawy i odpowiada jednej iteracji schematu DS. RACS sprawdza wszystkie $n-1$ permutacji otrzymanych z π^{RA} przez wymiane par przyległych. Ponieważ obliczenie wartości funkcji celu ma złożoność $O(nm)$, ocena ad hoc kosztu obliczeń dla RACS jest $O(n^2m)$. Jednakże, własność blokowa z Rozdz. 12.3 wyraźnie wskazuje, że jedynymi wymianami mogącymi wnieść poprawę wartości kryterium są te, w których jednym z elementów pary jest zadanie $\pi(u_k)$ dla pewnego k, $1 \leqslant k \leqslant m-1$. Par takich, zgodnie z własnością ścieżki krytycznej jest $O(m)$, co sugeruje znaczne ograniczenie liczby niezbędnych do sprawdzenia permutacji zaburzonych do wielkości $O(m)$ oraz daje wynikającą stąd redukcję złożoności obliczeniowej algorytmu RACS do poziomu $O(nm^2)$, szczególnie korzystną dla $m \ll n$. Dalej, wystarczy zauważyć, że dysponując wartościami r_{st}, q_{st}, $s = 1, \ldots, m$, $t = 1, \ldots, n$, wyznaczonymi dla permutacji π^{RA} według wzorów (12.18)–(12.19), wartość $C_{\max}$ dla permutacji po wymianie pojedynczej pary zadań można policzyć w czasie $O(m)$, używając techniki analogicznej do opisanej przy efektywnej implementacji algorytmu NEH. Stąd otrzymujemy ostatecznie złożoność obliczeniową dla RACS równą $O(nm + m^2)$ co dla $m \leqslant n$ sprowadza się do $O(nm)$. Kolejny algorytm RAES, powtarza iteracyjnie RACS jeśli tylko wystąpiła poprawa wartości kryterium po jego zastosowaniu.

Akceleratory dla API w DS

Iteracyjną (wielokrotną) poprawę poprzez API można zastosować do każdego rozwiązania generowanego dowolnym algorytmem. Niestety, obliczanie

explicite wartości $C_{\max}$ dla pełnego otoczenia API w technice DS ma kosztowną złożoność obliczeniową $O(n^2m)$. Istnieją dwie niezależne teoretyczne własności przyśpieszające tą czynność co najmniej n-krotnie: dekompozycja i agregacja obliczeń przy wyznaczaniu wszystkich wartości $C_{\max}$ oraz eliminacyjna własność blokowa. Własności te mogą być stosowane niezależnie lub łącznie. Omówimy je kolejno.

Niech r_{st}, q_{st} będą wielkościami wyznaczonymi z wzorów (12.18)–(12.19) dla n-elementowej permutacji π, zaś $v = (a, a+1)$ niech będzie parą przyległych pozycji, których wymiana w permutacji π prowadzi do nowego rozwiązania $\pi_{(v)}$. Wówczas wartość $C_{\max}(\pi_{(v)})$ może być znaleziona w czasie $O(m)$ ze wzorów

$$C_{\max}(\pi_{(v)}) = \max_{1 \leqslant s \leqslant m} (d'_s + q_{s,a+2}), \tag{12.23}$$

gdzie

$$d'_s = \max\{d'_{s-1}, d_s\} + p_{s\pi(a)},\ s = 1, \ldots, m \tag{12.24}$$

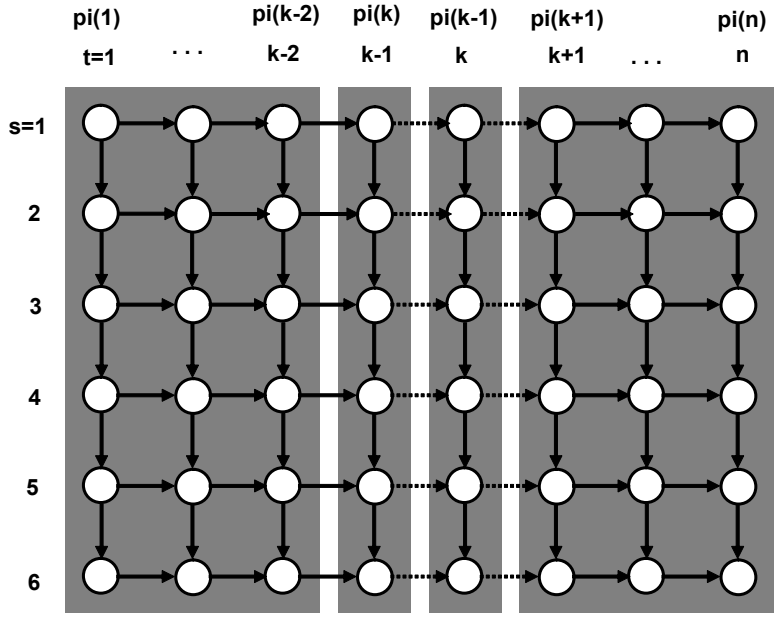
reprezentuje długość najdłuższej drogi dochodzącej do wierzchołka $(s, a+1)$ w grafie $G(\pi_{(v)})$ oraz

$$d_s = \max\{d_{s-1}, r_{s,a-1}\} + p_{s\pi(a+1)},\ s = 1, \ldots, m, \tag{12.25}$$

reprezentuje długość najdłuższej drogi dochodzącej do wierzchołka (s, a) w $G(\pi_{(v)})$ (po wykonaniu zamiany). Odpowiednie warunki brzegowe są zerowe, tzn. $d'_0 = 0 = d_0$, $r_{s0} = 0 = q_{s,n+2}$, $s = 1, \ldots, m$. Teoretycznie wszystkie $n-1$ wymian v może być sprawdzone w czasie $O(nm)$. Stąd dekompozycja i agregacja obliczeń (akcelerator podstawowy) przyśpiesza pracę algorytmu $O(n)$ razy. Dalej, własność blokowa stanowi, że żadna z wymian $v = (a, a+1)$, $u_{i-1} < a < a+1 < u_i$, $i = 1, \ldots, m+1$, nie dostarczy $C_{\max}(\pi_{(v)})$ mniejszego od $C_{\max}(\pi)$. Stąd jedynymi "obiecującymi" wymianami są te, dla których $a = u_i$ bądź $a+1 = u_i$ dla pewnego i. Par takich jest co najwyżej $2(m-1)$, co sugeruje złożoność obliczeniową $O(m^2)$. Jednakże wyznaczenie wartości r_{st}, q_{st} niezbędnych we wzorach (12.23)-(12.25) wymaga czasu $O(nm)$. Zatem własność blokowa wprawdzie dalej zmniejsza realny nakład obliczeń dla API w DS, lecz nie zmniejsza złożoności obliczeniowej akceleratora podstawowego, pozostawiając ją na poziomie $O(nm)$.

Akceleratory dla NPI w DS

Szczególnego znaczenia nabiera problem redukcji kosztu obliczeń dla metod DS krzystających z modyfikacji NPI. Liczba wszystkich rozwiązań zaburzonych osiągalnych z danej permutacji π w jednej iteracj metody DS

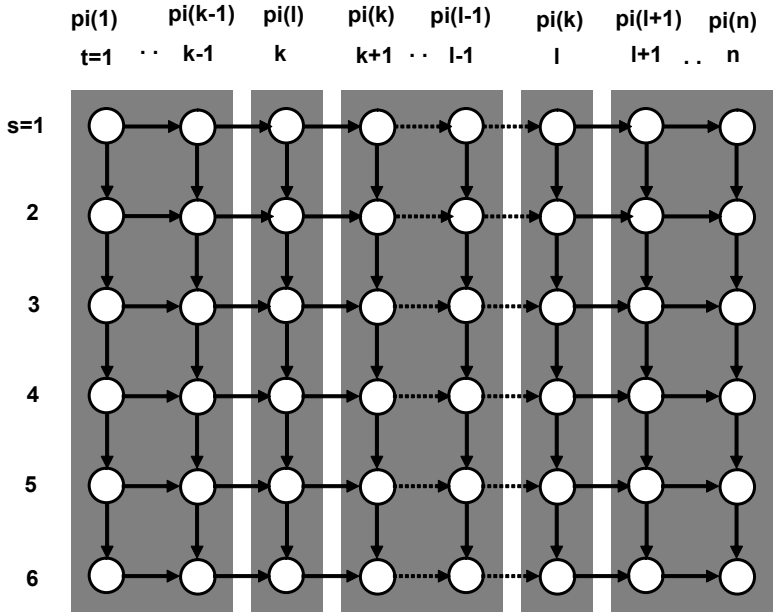


Rysunek 12.4: Akcelerator dla API

jest w tym przypadku równa $n(n-1)/2$, czyli $O(n^2)$. Przyjmując przeszukiwanie lokalnego otoczenia techniką "do najlepszej poprawy", otrzymamy dość wysoką złożoność obliczeniową $O(n^3m)$ wyznaczania wszystkich wartości funkcji w otoczeniu. Umiejętna dekompozycja i agregacja obliczeń oraz właściwe wykorzystanie własności blokowych pozwalają na znaczną redukcję czasu obliczeń.

Niech $v = (a, b)$, $a \neq b$ określa parę zadań $(\pi(a), \pi(b))$, których wymiana generuje nową permutację $\pi_{(v)}$. Bez straty ogólności rozważań możemy przyjąć $a < b$, ze względu na symetrię. Niech dalej r_{st}, q_{st}, $s = 1, \ldots, m$, $t = 1, \ldots, n$ będą wielkościami wyznaczonymi ze wzorów (12.18)–(12.19) dla permutacji n-elementowej π. Oznaczmy przez D_{st}^{xy} długość najdłuższej drogi pomiędzy węzłami (s, t) oraz (x, y) w grafie $G(\pi)$. Korzystając z wprowadzonych oznaczeń możemy wyrazić wartość $C_{\max}(\pi_{(v)})$ poprzez zastosowanie poniższego ciągu wzorów, patrz także Rys. 12.5. Wszystkie wymienione poniżej długości dróg odnoszą się do grafu $G(\pi_{(v)})$. Wpierw w czasie $O(m)$ obliczamy długość najdłuższej drogi dochodzącej do wierzchołka (s, a), dołączając zadanie $\pi(b)$ przestawione na mocy zamiany na pozycję a

$$d_s = \max\{d_{s-1}, r_{s,a-1}\} + p_{s\pi(b)},\ s = 1, \ldots, m, \tag{12.26}$$



Rysunek 12.5: Akcelerator dla NPI

gdzie $d_0 = 0$. Następnie, w czasie $O(m^2)$ obliczamy długość najdłuższej drogi dochodzącej do wierzchołka $(s, b-1)$, dołączając "fragment" grafu zawarty pomiędzy zadaniami na pozycjach od $a+1$ do $b-1$ włącznie, niezmienny względem $G(\pi)$

$$d'_s = \max_{1 \leqslant w \leqslant s} (d_w, D^{s,b-1}_{w,a+1}),\ s = 1, \ldots, m. \tag{12.27}$$

Kolejno, w czasie $O(m)$ obliczamy długość najdłuższej drogi dochodzącej do wierzchołka (s, b), dołączając zadanie $\pi(a)$ przestawione w wyniku zamiany na pozycję b

$$d''_s = \max\{d''_{s-1}, d'_s\} + p_{s\pi(a)},\ s = 1, \ldots, m, \tag{12.28}$$

gdzie $d''_0 = 0$. W końcu otrzymujemy

$$C_{\max}(\pi_{(v)}) = \max_{1 \leqslant s \leqslant m} (d''_s + q_{s\pi(b+1)}). \tag{12.29}$$

Zatem obliczenie pojedynczej wartości $C_{\max}(\pi_v)$ jest możliwe w czasie $O(m^2)$ lecz pod warunkiem, że dysponujemy odpowiednimi wartościami D^{xy}_{st}. Te

ostatnie wielkości możemy obliczać rekurencyjnie, dla ustalonego t oraz y oraz $y = t+1, t+2, \ldots, n$, korzystając z zależności

$$D_{st}^{x,y+1} = \max_{s \leqslant k \leqslant x} (D_{st}^{ky} + \sum_{i=k}^{x} p_{i\pi(y+1)}), \tag{12.30}$$

gdzie $D_{st}^{xt} = \sum_{i=s}^{x} p_{i\pi(t)}$. Metoda ta pozwala wyznaczyć wszystkie potrzebne wartości D_{st}^{xy} w czasie $O(n^2m^2)$. Ostatecznie obliczenie wszystkich $O(n^2)$ wartości $C_{\max}(\pi_{(v)})$ można wykonać w czasie $O(n^2m^2)$, czyli w czasie $O(n/m)$ razy krótszym niż bez użycia akceleratora.

Dalej, analiza otoczenia typu NPI w kontekście własności blokowych wskazuje, że wymiana pary zadań dla $u_{k-1} < i, j < u_k$, to jest zadań pochodzących z tego samego bloku wewnętrznego, nie dostarcza poprawy wartości funkcji celu, zatem może być pominięta. Niestety, liczba pomijanych w ten sposób rozwiązań jest średnio $O(n^2/m)$, co stanowi przeciętnie nieznaczną frakcję $2/m$ wszystkich rozwiązań analizowanych w otoczeniu. Stąd własność blokowa, w tym przypadku, dostarcza niewielkich korzyści w porównaniu z akceleratorem podstawowym.

Akceleratory dla INS w DS

Liczba wszystkich rozwiązań zaburzonych osiągalnych z danej permutacji π w jednej iteracj metody DS z INS jest $O(n^2)$, co implikuje, podobnie jak poprzednio, złożoność obliczeniową $O(n^3m)$ przeglądania otoczenia. Jednakże akcelerator polecany dla tego przypadku ma nieco odmienną budowę, podobną w swej filozofii to efektywnej implementacji Algorytmu NEH.

Niech $v = (a, b)$, $a \neq b$ określa modyfikację INS polegającą na usunięciu (wycięciu) zadania $\pi(a)$ oraz wstawieniu do tak, by zajmował pozycję b w permutacji $\pi_{(v)}$. Niech r_{st}, q_{st}, $s = 1, \ldots, m$, $t = 1, \ldots, n-1$ będą wielkościami wyznaczonymi ze wzorów (12.18)–(12.19) dla permutacji $n-1$-elementowej otrzymanej z π przez wycięcie zadania $\pi(a)$. Wartość $C_{\max}(\pi_{(v)})$ dla $\pi_{(v)}$ otrzymanej przez wstawienia zadania $\pi(a)$ na pozycję $b = 1, \ldots, n$, $b \neq a$, można obliczyć korzystając z zależności (12.20)–(12.21) w czasie $O(m)$. Stąd, obliczenie $C_{\max}(\pi_{(v)})$ dla ustalonego a oraz wszystkich $b = 1, \ldots, n$, $b \neq a$ wymaga czasu $O(nm)$. Odpowiednio, dla rozwiązań zaburzonych $\pi_{(v)}$ generowanych z π w technice INS, sprawdzenie wszystkich $C_{\max}(\pi_{(v)})$ wymaga czasu $O(n^2m)$, zamiast oczekiwanego $O(n^3m)$.

Wykorzystanie własności blokowych w akceleratorze INS jest w pełni analogiczne do ich użycia w efektywnej implementacji NEH.

12.8 Algorytmy TS

Metody TS stanowią silną konkurencję dla DS bowiem oferują dla problemu $F^*||C_{\max}$ błąd przeciętny rzędu 1-1.5%, nieosiągalny zarówno dla DS jak też wielu innych zaawansowanych podejść.

W literaturze występuje niewiele prac wykorzystujących TS dla rozwiązywania problemu przepływowego [7,269,357,383]. Najefektywniejsza aktualnie dostępna implementacja, znana jako algorytm TSAB [269], może być przedstawiona krótko za pomocą elementów składowych, odpowiednio do opisu ogólnego z Rozdz. 8. Dla zapewnienia szybszej zbieżności do dobrego rozwiązania, algorytm TSAB startuje z rozwiązania dostarczonego przez NEH. Zastosowanie innego rozwiązania startowego ogólnie nie ma wpływu na dokładność TSAB lecz tylko na czas jego działania.

Dla rozwiązania bazowego π konstruowane jest otoczenie w oparciu o technikę INS wspartą własnościami blokowymi. Bardziej dokładnie, niech $\pi_{(v)}$, $v = (a, b) \in \mathcal{J} \times \mathcal{J}$, będzie zbiorem wszystkich permutacji, które można otrzymać w wyniku zastosowania modyfikacji INS na π. Pojedynczą taką modyfikację określoną przez parę $v = (a, b)$ będziemy nazywać *ruchem*, zgodnie z nomenklaturą techniki TS. Rozpatrzmy zadanie $\pi(a)$, $u_{i-1} < a < u_i$ należące do wnętrza pewnego bloku B_i w π, to znaczy takie, że $u_{i-1} < a < u_i$. Zgodnie z eliminacyjną własnością blokową zadanie to należy przesunąć na pewną pozycję $b \leqslant u_{i-1}$ lub $b \geqslant u_i$, bowiem dla $u_{i-1} < b < u_i$ nie uzyskamy poprawy wartości funkcji celu (te ostatnie ruchy nazywane są bezużytecznymi). Niestety, własność blokowa nie precyzuje gdzie należy to zadanie przesunąc, aby uzyskać poprawę wartości $C_{\max}$. Stąd dla wymienionego zadania $\pi(a)$ wprowadza się zbiór potencjalnie korzystnych ruchów w lewo na pozycje $L_{\pi(a)} = \{b : \ b \leqslant u_{i-1}\}$ oraz w prawo na pozycje $R_{\pi(a)} = \{b : \ b \geqslant u_i\}$. Odpowiednie zbiory dla zadań $\pi(a)$, $a \in \{u_1, u_2, \ldots, u_m\}$, zawierają wszystkie wlaściwe pozycje w lewo $b < a$ oraz prawo $b > a$, bowiem eliminacyjna wlasność blokowa nie ma zastosowania do nich. Dalej, wspomniana własność pozwala przyjąć dla $a < u_1$ zbiór $L_{\pi(a)}$ pusty, podobnie dla $a > u_m$ zbiór $R_{\pi(a)}$ jest pusty. Tak otrzymany, stosunkowo duży zbiór ruchów jest zbyt kosztowny do analizy drogą bezpośredniego obliczania wartości funkcji celu dla wszystkich sąsiadów w otoczeniu, nawet wbudowując odpowiednie akceleratory. Co więcej, w praktyce tylko niewielka liczba ruchów spośród wymienionych w rzeczywistości poprawia wartość funkcji celu. Obserwacja ta implikuje co najmniej dwa odmienne w swojej koncepcji podejścia do dalszego projektowania algorytmu.

Pierwsze podejście, zastosowane w oryginalnym algorytmie TSAB [269], bazuje na obserwacji, że wybór ruchu do wykonania (niekoniecznie popra-

wiajacego $C_{\max}$) nie ma wpływu na porawność i stabilność metody TS, lecz tylko na szybkość jej zbieżności. Stąd, ogranicza się arbitralnie wielkość zbiorów $L_{\pi(a)}$ oraz $R_{\pi(a)}$ do pewnych zwartych ich fragmentow, używając w tym celu parametru kontrolnego ϵ dobieranego eksperymentalnie. Technika ta nie daje gwarancji selekcji w otoczeniu wyłącznie lepszych rozwiazań, jednak pozwala na wprowadzenie akceleratora oraz zapewnia kompromis pomiędzy ilością obliczeń przy przeglądaniu otoczenia w jednym cyklu iteracyjnym TSAB, a szybkością zbiegania do dobrego rozwiazania, prz jednoczesnej stabilnej pracy metody TS. Dokładniej, dla każdego zadania $\pi(a)$, $u_{i-1} < a < u_i$, rozważane są ruchy $v = (a, b)$ dla których b należy do zbiorów $L_{\pi(a)}(\epsilon) = \{u_{i-1} - \epsilon, \ldots, u_{i-1}\}$ oraz $R_{\pi(a)}(\epsilon) = \{u_i, \ldots, u_i + \epsilon\}$, odpowiednio. Dla symetrii, dla $\pi(a)$, $a \in \{u_1, \ldots, u_m\}$ odpowiednie wzory także posiadają strukturę jak wyżej; słuszność tego postępowania potwierdzono badaniami eksperymentalnymi. Sugeruje się dobór ϵ na bazie doświadczeń, w zależności od ilorazu n/m. Zaleca się *epsilon* $= 0$ dla $n/m \geqslant 3$ oraz liniową zmianę wartości tak by dla $n/m \leqslant 0.5$ osiągnąć wartość $\epsilon = u_{i-1} - u_{i-2}$ dla zbioru $L_{\pi(a)}$ oraz $\epsilon = u_{i+1} - u_i$ dla zbioru $R_{\pi(a)}$.

Drugie podejście, z użyciem tak zwanych rozszerzonych własności blokowych przedstawiono w pracy [135] w formie algorytmu TSGP. Algorytm ten powtarza zasadniczo budowę algorytmu TSAB, wprowadzając różnice jedynie w *zakresie* wykonywanych ruchów. TSGP wykonuje w każdym kroku rozległą serię "celowanych", rozproszonych ruchów typu INS zamiast niewielkiej liczby ruchów obiecujacych stosowanych w TSAB. Dokładniej, TSGP filtruje wstępnie zbior $L_{\pi(a)}$ i $R_{\pi(a)}$ w celu wykrycia najbardziej obiecującej pozycji, używając do tego celu pewnej niekosztownej funkcji oceny wyznaczanej w czasie $O(1)$. Następnie, dla tak wstępnie wyselekcjonowanych rozwiązń wyliczane są explicite wartości funkcji celu dla wyboru ruchu do wykonania. Podejście to pozwala na szybszą zbieżność do dobrego rozwiazania, jednak powiększa znacznie koszt przeglądaniapojedynczego otoczenia, ograniczając także zyski z ewentualnego zastrosowania akceleratora. Ostatecznie, podstawowe własności numeryczne kleczowe dla zastosowań praktycznych pozostają podobne dla TSAB i TSGP.

Dla kompletności opisu algorytmu TSAB należy podać: atrybuty zapisywane na listę tabu T, sposób badania statusu ruchu, strategię wyboru sąsiada oraz wartości parametrów strojących. Jeśli wykonywanym ruchem jest $v = (a, b)$, $a < b$ to na listę T wpisujemy parę $(\pi(a), \pi(a+1))$, jeśli $a > b$ to na listę T wpisujemy parę $(\pi(a-1), \pi(a))$. Lista T pełni funkcję pamięci krótkoterminowej, ma długość ograniczoną parametrem t, jest obsługiwana według reguły FIFO, poczatkowo jest pusta, zaś dodanie nowego elementudo zapełnionej listy powoduje automatyczne usunięcie elementu

najstraszego. Ruch $v = (a, b)$ ma status tabu (zakazany) jeśli: $a < b$ oraz co najmniej jedna para $(\pi(j), \pi(a))$, $a + 1 \leqslant j \leqslant b$ jest w T, lub $a > b$ oraz co najmniej jedna para $(\pi(a), \pi(j))$, $b \leqslant j \leqslant a - 1$ jest w T. Rekomendowaną wartością t jest $6 \ldots 10$. Strategię wyboru ruchu do wykonania zaprojektowano w celu zapewnienia kompromisu pomiędzy dywersyfikacja a intensyfikacją poszukiwań. W tym celu każdy zbiór $L_{\pi(a)}$ i $R_{\pi(a)}$ jest przeszukiwany odzielnie w celu wyłonienia "zwycięskiego" ruchu-kandydata. Wybór ruchu do wykonania jest dokonywny następnie pomiędzy wyłącznie ruchami-kandydatami, kierując się wartością funkcji kryterialnej i statusem ruchu. Najefektywniejsza wersja algorytmu TSAB posiada dodatkowo wbudowany mechanizm dywersyfikacji oparty na koncepcji *skoków powrotnych* [269]. Algorytm ten jest dotychczas najefektywniejszym znanym dla rozwiązywania problemu $F^*||C_{\max}$.

Sąsiedztwo o *sterowanej* wielkości jest dość wygodnym narzędziem dla ustalania kompromisu pomiędzy dwoma przeciwstawnymi tendencjami: nakładem obliczeń przypadających na iterację algorytmu (przeglądnięcie pojedynczego śąsiedztwa), a szybkością zbieżności do dobrego rozwiązania. Koncepcja ta pochodzi pierwotnie z pracy [7], gdzie zastosowano ją do zaburzeń NPI oraz addytywnej funkcji kryterialnej.

12.9 Poszukiwanie mrówkowe

Podamy poniżej dość proste implementacje algorytmu mrówkowego dla problemu $F * ||C_{\max}$, wzorowane na idei pochodzącej z pracy [83]. Ponieważ istniej pewna swoboda w wyborze odwzorowania pojedynczego rozwiązania w trasę mrówek, możliwe są co najmniej dwa odmienne podejścia.

Trasa jako następstwo zadań

Przyjmujemy, że pomiędzy gniazdem a pożywieniem krąży a mrówek. Trasa mrówki s, $s = 1, \ldots, a$ jest reprezentowana pojedynczą permutacją π^s na zbiorze $\mathcal{J}$ i odpowiada przejściu mrówki po krawędziach łączących punkty węzłowe trasy, Rys. **??**. Węzły odpowiadają zadaniom, krawędzie reprezentują kolejność (bezpośrednią) występowania zadań po sobie w permutacji. Punkt o reprezentuje gniazdo, punkt $*$ pożywienie. Trasa mrówki jest *legalna*, jeśl zaczyna się w o, kończy w $*$ oraz każdy inny węzeł występuje w trasie dokładnie raz. Ślad feromonowy jest przypisany wyłącznie krawędziom, przy czym krawędź (i, j), $i, j \in \mathcal{J}$ posiada intensywność śladu o wartości f_{ij}. Odpowiednie intensywności wokół gniazda i jedzenia oznaczono f_{oj}, f_{i*}. Mrówka znajdująca się w punkcie węzłowym i podąża do

punktu węzłowego j wybranego losowo z prawdopodobieństwem zależnym od intensywności śladu na krawędzi (i, j), spośród tych j, które gwarantują legalność trasy. Długość trasy π^s mrówki s jest wartością funkcji kryterialnej $C_{\max}(\pi^s)$. Wielkość feromonu w podłożu jest korygowana raz na cykl, obejmujący zaplanowanie tras wszystkich a mrówek, jako funkcja długości trasy mrówki.

Ustalenie legalnej trasy π^s mrówki s odpowiada generowaniu losowej permutacji częściowej σ o zaburzonym rozkładzie, poczynając od permutacji pustej, aż do kompletnej, to jest zawierajacej wszystkie zadania ze zbioru $\mathcal{J}$. Bardziej dokładnie, niech $\sigma = (\sigma(1), \ldots, \sigma(k))$ będzie k-elementową permutacją częściową, reprezentującą przejście mrówki po trasie $o \to \sigma(1) \to \ldots \to \sigma(k)$. Z punktu węzłowego $\sigma(k)$ mrówka wybierze trasę do punktu $t \in \mathcal{U} \stackrel{\text{def}}{=} \mathcal{J} \setminus \mathcal{S}$, gdzie $\mathcal{S} = \{\sigma(1), \ldots, \sigma(k)\}$, kierując się zasadą ruletki w oparciu o macierz f_{ij}. Losowanie ruletką przypisuje prawdopodobieństwo wyboru punktu węzłowego $t \in \mathcal{U}$ jako następnego po $i = \sigma(k)$ w trasie równe $P_{it} = f_{it}/(\sum_{j\in\mathcal{U}} f_{ij})$.

Gdy trasy wszystkich a mrówek zostaną ustalone, dokonywana jest korekta ilości feromonu w podłożu. Wpierw obliczany jest "świeży ślad" feromonowy pozostawiany przez pojedynczą mrówkę s w tym cyklu równy

$$\Delta^s_{\sigma^s(j-1)\sigma^s(j)} = \frac{A}{C_{\max}(\pi^s)} \tag{12.31}$$

dla krawędzi z trasy tej mrówki, to jest dla $j = 1, \ldots, n+1$, gdzie $pi^s(0) = o$, $pi^s(n+1) = *$, oraz równy zero dla wszystkich pozostałych krawędzi. Wielkość A jest pewną stałą dobieraną eksperymentalnie. Świeży ślad pochodzący od wszystkich mrówek jest następnie kumulowany

$$\Delta_{ij} = \sum_{s=1}^{a} \Delta^s_{ij}, \tag{12.32}$$

przy czym ilość feromonu w podłożu jest korygowana zgodnie z zależnością

$$f_{ij} = e \cdot f_{ij} + \Delta_{ij}, \tag{12.33}$$

gdzie $0 < (1 - e) < 1$ jest pewną stałą reprezentującą szybkość parowania feromonu z podłoża. Po wykonaniu cyklu przejścia gniazdo $\to$ pożywienie, mrówki wykonuja w pełni analogiczny cykl powrotny. Ostatecznie może zachodzić $f_{ij} \neq f_{ji}$, co skłania ku interpretacji, że punkty węzłowe i oraz j są połączone parą przeciwstawnie skierowanych krawędzi o różnym nasyceniu feromonu, zaś mrówka może się poruszać jedynie zgodnie ze skierowaniem krawędzi.

Algorytm mrówkowy wymaga ustalenia na drodze eksperymentalnej kilku parametrów strojących, a mianowicie: liczby mrówek a, początkowego nasycenia feromonem f_{ij} $i, j \in \mathcal{J}$, parametrów e, A, maksymalnej liczby cykli lub innego warunku stopu. Algorytm zwraca jako wynik najlepsze rozwiązanie wygenerowane w trakcie swojej pracy.

Mimo dużej swobody w ustalaniu parametrów pracy, algorytm ten cierpi, podobnie jak wiele innych z powodu zjawiska *stagnacji* (przedwczesnej zbieżności) poszukiwań i/lub powolnej zbieżności do dobrego rozwiazania. Przez stagnację rozumie się przypadek, w którym wszystkie mrówki krążą po jednakowej trasie, blokując mozliwości generowania alternatywnych tras. Dlatego też, proponuje się szereg modyfikacji usprawniających jego pracę.

Metoda nasycenia feromonem tras (12.31) jest dość wrażliwa na zakres wartości $C_{\max}$ wynikających z konkretnych danych problemu. Powoduje to konieczność ustalenia wartości parametru A oddzielnie dla każdej podklasy konkretnych przykładów liczbowych. Niezgodność tą można usunąć wprowadzając w miejsce $C_{\max}(\pi^s)$ oceną bezwzględną korzyści $C_{\max}(\pi^s) - C^{ref}$ lub ocenę względną $(C_{\max}(\pi^s) - C^{ref})/C^{ref}$, gdzie C^{ref} jest wartością referencyjną wyliczaną automatycznie, indywidualnie dla każdego przykładu (może to być na przykład dolne ograniczenie wartości funkcji celu).

Ograniczenie repertuaru punktów węzłowych osiągalnych przez mrówkę s w trakcie budowy jej legalnej trasy częściowej nazywane jest często *pamięcią krótkoterminową* mrówki (short term memory). Podobnie, wprowadzany jest dodatkowy atrybut mrówki nazywany *krótkowzroczną widzialnością* (visibility) lub wzrokiem, pozwalający oddziaływać dodatkowo na wybór kolejnego punktu na trasie. Przyjmując identyczność osobników z populacji, każda mrówk umieszczana w węźle i ocenia wzrokiem, w jednakowy sposób, trasę do punktów sąsiednich $j \in \mathcal{J}$, przypisując krawędziom pewną aprioryczną wartość v_{ij}. Dalej wybór odbywa się metodą ruletki, lecz w oparciu o wartości $[f_{ij}]^{\alpha}[v_{ij}]^{\beta}$ w miejsce f_{ij}.

W celu wzmocnienia wpływu dobrych rozwiązań na ustalanie tras mrówek stosowana jest *strategia elitarna* zwiększająca dodatkowo w każdym cyklu ilość feromonu dla najlepszych dotychczas znalezionych tras, według wzoru (12.31).

Proponowany okresowo-cykliczny schemat zmiany zawartości feromonu (po ustaleniu kompletnych tras mrówek) wydaje sie nieco sztuczny bowiem w naturze zmiana zawartości feromonu w podłożu ma charakter dynamiczny i następuje w trakcie przemieszczania się mrówki. W pracy [83] analizowano również dwa inne schematy zmian dynamicznych, oba oparte na zwiększeniu zawartości feromonu w krawędzi bezpośrednio po przejściu mrówki: (a) *gęstościowy* zwiększający zawsze o stałą wartość, (b) *ilościowy* zwiększający

o wartość odwrotnie proporcjonalna do widzialności. Badania eksperymentalne wykazały, że warianty (a) i (b) są gorsze obliczeniowo od cyklicznego, głównie ze względu na niemożność wykorzystania informacji o ocenie trasy $C_{\max}(\pi^s)$ przed jej zakończeniem.

Proponowane są ustalenia parametrów strojacych następująco: $a = n$, $\alpha = \beta = 1$, $f_{ij} = \epsilon$, gdzie ϵ jest pewną małą liczbą dodatnią, $e = 0.5$. Wartość A występująca we wzorze (12.31) zalecano w pracy [83] równą 100. Korzystając z alternatywnej postaci wyrażenia (12.31) w formie

$$\Delta^s_{\sigma^s(j-1)\sigma^s(j)} = \frac{A \cdot LB}{C_{\max}(\pi^s) - LB} \tag{12.34}$$

proponuje się ustalić wartość $A = fracLBa(C - LB)$, gdzie $C = \max_{1 \leqslant j \leqslant u} C_{\max}(\tau^j)$ oraz $\tau^1, \ldots, \tau^u$ jest serią losowych rozwiązań próbnych generowanych w fazie poprzedzającej właściwą pracę algorytmu ($u \approx n$). Maksymalna liczba cykli jest ustalona na poziomie 5,000. Za kryterium stopu przyjęto przekroczenie maksymalnej liczby cykli lub wystąpienie stagnacji poszukiwań.

Wybór skrajny $\alpha = 0$ oraz $\beta = 1$ odpowiada poszukiwaniu losowemu bez wykorzystania wiedzy globalnej zakumulowanej w feromonie. Miara widzialności v_{ij} może odzwierciedlać dowolne heurystyczne preferencje (priorytet statyczny) w wyborze kolejnego zadania do uszeregowania. Z tego punktu widzenia jest to pewna forma "wspierania" algorytmu mrówkowego za pmoca heurezy. Przeprowadzone przez autorów pracy [83] badania dla problemu komiwojażera (TSP) wykazały konieczność ustalenia $\beta \geqslant \alpha = 1$ w celu zagwarantowania w miarę stabilnej pracy algorytmu, unikającej stagnacji i braku zbieżności. Praktyczny obszar zalecanych wartości α, β jest dość wąski, zaś algorytm wykazuje dużą wrażliwość na zmiany tych parametrów. Ostatecznie polecane ustalenie parametrów $\alpha = 1 = \beta$ odpowiada demokratycznej współpracy dwóch odmiennych technik poszukiwań: (1) losowej z ustalonym rozkładem prawdopodobieństwa, wynikającym z priorytetów statycznych oraz (2) losowej z krokowo modyfikowanym prawdopodobieństwem wynikającym z pamięci poszukiwań (feromon).

Trasa jako przyporządkowanie zadań do pozycji

Podobnie jak poprzednio, trasa mrówki s, $s = 1, \ldots, a$ jest reprezentowana pojedynczą permutacją π^s na zbiorze $\mathcal{J}$, z tym, że budowa permutacji polega na przyporządkowaniu zadań ze zbioru $\mathcal{J}$ do pozycji $1, 2, \ldots, n$, Rys. **??**. W rysunku, węzeł (i, j) odpowiada umieszczeniu zadania i na pozycji j. Węzeł o reprezentuje gniazdo, węzeł $*$ pożywienie. Mrówka wyruszajac z o porusza się po węzłach i krawędziach, kierując się w stronę $*$ co zapewnia,

Rysunek 12.6: SA-A. Bieżąca i najlepsza wartość funkcji celu. Przykład 50|20|3.

Rysunek 12.7: SA-A. Wartości funkcji celu rozwiązań zaburzonych.

że każda pozycja w permutacji zostanie obsadzona jakimś zadaniem. Trasa mrówki jest legalna, jeśli zaczyna się w o, kończy w $*$ oraz każdemu zadaniu przypisano pozycję jednoznacznie. W tym podejściu ślad feromonowy jest przypisany wyłącznie węzłom, węzeł (i, j), $i, j \in \mathcal{J}$ posiada intensywność śladu o wartości f_{ij}. Mrówka znajdująca się w punkcie węzłowym (i, j) podąża do punktu węzłowego $(i', j + 1)$, wybranego losowo z prawdopodobieństwem zależnym od intensywności śladu w punkcie $(i', j + 1)$ spośród tych i', które gwarantuja legalność trasy. Podobnie jak poprzednio, długość trasy π^s mrówki s jest wartości funkcji kryterialnej $C_{\max}(\pi^s)$, zaś wielkość feromonu w podłożu jest korygowana raz na cykl, obejmujący zaplanowanie tras wszystkich a mrówek, jako funkcja długości trasy mrówki.

Podobieństwo techniki mrówkowej do losowych poszukiwań (RS) sugeruje bardzo szybki wzrost czasu obliczeń wraz ze wzrostem rozmiaru przykładów problemu. Potwierdzają to także wyniki badań eksperymentalnych. Jak do tej pory brak jest szczegółowych danych dotyczących konkurencyjności tego podejścia dla problemu $F^*||C_{\max}$ względem innych podejść wymienionych w tym rozdziale. Za korzystne należy uznać uniwersalność podejścia pozwalającą modelować dowolne inne problemy dyskretne, których zbiór rozwiązań oparty jest na paermutacjach. Jednakże błędy względne rzędu 10% osiągane przez autorów pracy [83] dla innych problemów szeregowania o niewielkich rozmiarach nie skłaniają do optymizmu.

12.10 Symulowane wyżarzanie

Istniej wiele algorytmów dla problemów przepływowych opartych na schemacie SA głównie dla zagadnienia $F^*||C_{\max}$, choć ich zastosowanie rozciąga się na inne problemy, w których rozwiązanie jest reprezentowane permutacją, niezależnie od postaci funkcji celu oraz ograniczeń. Ich syntetyczny opis odwołuje się do ogólnego schematu podanego w Rozdz. 8 oraz specyficznych szczegółów podanych poniżej. Rozwiązanie jest reprezentowane permutacją π na $\mathcal{J}$, dla którego funkcję celu określa wartość $C_{\max}(\pi)$ (odpowiednio inna w zależności od rozwiązywanego problemu). Kosztowniejszy obliczeniowo wariant algorytmu z automatycznym strojeniem (oznaczony

Rysunek 12.8: SA-A. Zmiany temperatury.

dalej SA/A) jest w ogólnym rozrachunku korzystniejszy, bowiem unika się żmudnych wstępnych badań eksperymentalnych niezbędnych dla strojenia ręcznego. Rozwiązaniem startowym dla SA/A jest permutacja losowa. Przyjęcie w SA/A rozwiązania startowego dedykowanego, przez zastosowanie pewnego algorytmu konstrukcyjnego, zwykle wymaga "ręcznej" korekty (obniżenia) temperatury początkowej. Stosowane są różne techniki generowania rozwiązania zaburzonego π' względem π, przy czym do najczęściej spotykanych należą: (A) zamiana elementów przyległych $\pi(i), \pi(i+1)$ dla losowego $i \in \{1, \ldots, n-1\}$, (N) zamiana elementów dowolnych $\pi(i), \pi(j)$ dla losowych $i, j \in \mathcal{J}$, (I) przeniesienie elementu $\pi(i)$ tak by znalazł się na nowej pozycji j dla losowych $i, j \in \mathcal{J}$. Wybór techniki określa bezpośrednio wielkość zbioru stanów osiągalnych π. Przykładowo dla (N) jest równy $n(n-1)/2$. W celu uzyskania szybkiej zbieżności algorytmu do rozwiązania dobrego stosuje się bądź zawężanie tych zbiorów (np. implementując własności blokowe lub ograniczając arbitralnie zakres i, j) bądź ich rozszerzanie poprzez kombinację wymienionych technik. Dodatkowym elementem przyśpieszającym pracę algorytmu jest możliwość policzenia w czasie $O(1)$ wartości funkcji celu rozwiązania zaburzonego, niestety dostępna tylko dla wybranych zagadnień kombinatorycznych, np. QAP, TSP. Dobór elementów składowych algorytmu SA (techniki zaburzeń, dodatkowe własności, przyśpieszanie obliczeń, parametry strojące, kryterium stopu) jest zależny od typu rozwiązywanego problemu i zwykle jest dokonywany eksperymentalnie. Ostateczne własności numeryczne otrzymanego algorytmu zależą od umiejętnego połączenia tych elementów.

W Rys. 12.6–12.8 przedstawiono przebieg algorytmu SA-A dla przykładu 50|20|3 z testów Taillard'a dla problemu $F^*||C_{\max}$ o rozmiarze $n = 50$ oraz $m = 20$ przy wartościach parametrów strojących $p = 0.9$, $\delta = 0.1$, $k = n/4$, $K = n^2/4$ oraz zaburzeniach opartych na wymianie par elementów w permutacji. Widać wyraźnie tłumienie wahań wartości kryterim dla rozwiązania bieżącego $C_{\max}$ wraz ze zmniejszaniem się temperatury (patrz Rys. 12.8) oraz stabilizację najlepszej znalezionej wartości $C^*_{\max}$ po okresie szybkiego "wykładniczego" spadku w początkowym okresie poszukiwań. Równocześnie, wartości funkcji celu rozwiazań zaburzonych stale oscylują, Rys. 12.7, potwierdzając, że zasdniczym motorem postępu w technice SA są poszukiwania losowe. Metoda SA dostarcza rozwiązań bliskich optymalnemu, choć koszt obliczeń dla niektórych zagadnień może być nieakceptowalnie długi. Zwykle, w celu osiągnięcia "dobrego" przybliżenia zaleca się ustalić

możliwie dużą wartość powtórzeń w ustalonej temperaturze, rzędu $O(n^2)$ w podanym przykładzie. Pociąga to stosunkowo wolne studzenie oraz długi czas obliczeń.

Schemat SA może być polecany dla problemów, w których koszt obliczenia wartości funkcji celu dla pojedynczego rozwiązania jest znaczący, przy czym brak jest specyficznych własności problemu wspomagających inne klasy algorytmów. Wtedy generowanie losowego rozwiazania zaburzonego jest mniej kosztowne niż bezpośrednie badanie całego otoczenia, występujące w metodach takich jak DS, TS, AMS.

12.11 Poszukiwania ewolucyjne

Istnieje wiele odmian algorytmów GA dla problemów przepływowych. Najlepsze wyniki otrzymuje się kodując rozwiązania w chromosomie nie poprzez ciągi binarne lecz bezpośrednio za pomocą permutacji. To założenie wymaga zastosowania specyficznych operatorów genetycznych krzyżowania i mutacji, działających na permutacjach rodzicielskich i zwracających potomków w formie permutacji, CX, OX, PMX, patrz np. [112]. Istnieją dwie przeciwstawne tendencje w projektowaniu takich operatorów: (1) zachować korzystne cechy rodziców poprzez małe zmiany chromosomu – własność tą posiadają dwupunktowe operatory krzyżowania, (2) wprowadzić nowe cechy poprzez istotną zmianę chromosomu – własność tą posiadają operatory równomiernego krzyżowania. Obie wymienione cechy odpowiadają zachowaniu równowagi pomiędzy intensyfikacją i rozproszeniem poszukiwań w przestrzeni rozwiązań. Większość omawianych operatorów odwołuje się do elementarnej czynności wymiany pary elementów w permutacji. Oznaczmy przez $\pi : \ a \leftrightarrow b$ czynność modyfikacji permutacji π przez wymianę miejscami zadań $a, b \in \mathcal{J}$. Odpowiednio symbol $\pi : \pi(s) \leftrightarrow \pi(t)$ odnosi się do wymiany elementów znajdujących się na pozycjach $1 \leqslant s, t \leqslant n$ w π.

Podstawowymi operatorami genetycznymi działającymi na jednym rodzicu (operator jednoargumentowy) są *operator inwersji* (I) określony regułą $\pi : \pi(s) \leftrightarrow \pi(t)$ oraz *operator inwersji podciągu* (SI) określony regułą $\pi : \pi(s+j) \leftrightarrow \pi(t-j)$, $j = 0, \ldots, \lfloor (t-s-1)/2 \rfloor$, gdzie wartości $1 \leqslant s < t \leqslant n$ wybrano losowo. Operatory te są polecane głownie dla realizacji procesu mutacji. Dalsze operatory są dwuargumentowe.

Jednopunktowy operator krzyżowania (SX) dla dwóch rodziców π, σ oraz losowego punktu krzyżowania $1 \leqslant s \leqslant n$ tworzy, poprzez wymianę odcinka chromosomu, dwóch potomków $\alpha = (\pi(1), \ldots, \pi(s), \sigma(s+1), \ldots, \sigma(n))$ oraz $\beta = (\sigma(1), \ldots, \sigma(s), \pi(s+1), \ldots, \pi(n))$. Każdy potomek jest następnie

poddawani procesowi "korygowania" w celu przywrócenia mu własności permutacji. W tym celu elementy potomka są przeglądane oraz każdy element, który pojawia się dwukrotnie jest zastępowany najmniejszym elementem zbioru $\mathcal{J}$ nie występującym w potomku. Zakładając przeglądanie w sposób uporządkowany po indeksach pozycji $1, \ldots, n$ jak również $n, \ldots, 1$, operator SX dostarcza 1–4 permutacji (niektóre mogą się powtarzać).

Dwupunktowy operator krzyżowania PMX (partially matched crossover) dla dwóch rodziców π i σ oraz sekcji dopasowania określonej losową parą indeksów pozycji s, t, $1 \leqslant s < t \leqslant n$ tworzy dwie permutacje potomne dopasowując π do σ według zasady $\pi : \pi(j) \leftrightarrow \sigma(j)$, $j = s, \ldots, t$ oraz dopasowując σ do π według zasady $\sigma : \sigma(j) \leftrightarrow \pi(j)$, $j = s, \ldots, t$. Podana szczególna zasada jest realizacją pewnej ogólnej filozofii, w której potomek przejmuje odcinek chromosomu leżący poza sekcją dopasowania całkowicie od jednego z rodziców, zaś odcinek leżący w sekcji dopasowania tworzy samodzielnie w oparciu o odpowiednie odcinki sekcji dopasowania rodziców, tak by uzyskać rozwiązanie dopuszczalne. Operator dostarcza dwóch potomków.

Operator krzyżowania porządkowego OX (order crossover) dla dwóch rodziców π i σ oraz sekcji dopasowania określonej losową parą indeksów pozycji s, t, $1 \leqslant s < t \leqslant n$ tworzy dwie permutacje potomne. Potomek przyjmuje geny sekcji dopasowania całkowicie od jednego z rodziców w kolejności i ułożeniu zgodnym z rodzicem, zaś geny leżące poza sekcją dopasowania są dobierane od drugiego rodzica w kolejności ich występowania w chromosomie (drugiego rodzica), tak by uzyskać rozwiązanie dopuszczalne.

Operator krzyżowania pozycyjnego PBX (position-based crossover) został oryginalnie zaproponowany dla chromosomów reprezentowanych ciągami binarnymi. Przeniesienie podejścia na przypadek permutacji prowadzi do filozofii podobnej do OX. Zamiast sekcji dopasowania wybierane są u rodzica losowo, z równomiernym rozkładem prawdopodobieństwa, geny (pozycje w permutacji). Geny te są kopiowane do potomka z zachowaniem ich po położenia w chromosomie rodzica, zaś brakujące geny są uzupełniane od drugiego rodzica w kolejności ich występowania w chromosomie.

Operator krzyżowania opartego na porządku (OBX) (order-based crossover) jest pewną modyfikacją operatora PBX, w której kolejność genów w pozycjach wybranych u rodzica i przekazywanych potomkowi jest dostosowywana do kolejności ich występowania u drugiego rodzica.

Operator krzyżowania cyklicznego CX (cycle crossover) dla dwóch rodziców π i σ tworzy dwóch potomków, w których pozycja zajmowana przez każdy element pochodzi od jednego z rodziców. Rozważmy przypadek dopasowania π do σ (dopasowanie odwrotne jest symetryczne). Tworzymy permutację wynikową ω rozpoczynając od $k = 1$. Następnie powtarzamy ciąg

podstawień $\omega(k) = \pi(k)$, $k = \underline{\pi}(\sigma(k))$, gdzie $\underline{\pi}(\pi(i)) = i$, do chwili zamknięcia cyklu, to znaczy osiągnięcia stanu $k = \pi(1)$. Pozostałe wolne pozycje w ω zapełniamy niewykorzystanymi elementami z σ w kolejności ich występowania.

Operator krzyżowania z porządkiem liniowym LOX (linear order crossover) jest pewną modyfikacją operatora OX. Ponieważ OX zaprojektowany został oryginalnie dla problemu TSP, preferuje on zachowanie względnego zamiast bezwzględnego położenia genów na chromosomie. Dla problemów, w których cykliczność rozwiązania nie występuje bardziej odpowiedni jest operator LOX. Dla dwóch rodziców π i σ oraz losowej sekcji dopasowania s, t, $1 \leqslant s < t \leqslant n$ tworzy on permutacje potomne w dwóch etapach. Rozważmy przypadek dopasowania π do σ (dopasowanie odwrotne jest symetryczne). Wpierw tworzymy pomocniczą permutację częściową ω usuwając z π elementy $\sigma(s), \ldots, \sigma(t)$ oraz pozostawiając powstałe "dziury". Następnie w ω przeprowadzamy proces redukcji "dziur" analizując pozycje w kolejności cyklicznej $t+1, \ldots, n, 1, 2, \ldots, s-1$. Dziurę w pozycji k redukujemy przesuwając cyklicznie elementy $\omega(k+1), \ldots, \omega(n), \omega(1), \ldots, \omega(t)$ w lewo. Ostatecznie otrzymujemy permutację zawierającą dziury tylko w sekcji dopasowania, które zapełniamy elementami $\sigma(s), \ldots, \sigma(t)$ w takiej kolejności.

Mimo iż zaprojektowano szereg "czystych" operatorów działających na permutacjach, korzyść z ich zastosowania silnie zależy od klasy rozwiązywanego problemu. W skrajnych przypadkach zdarza się, że jedynym motorem postępu GA bywa mutacja zamiast krzyżowanie, co jest zwykle traktowane za uchybienie w sztuce. Idealne operatory powinny dostarczać potomków maksymalnie rozproszonych w przestrzeni rozwiązań przy równoczesnym ich dobrym dopasowaniu do środowiska. Badania krajobrazu (landscape) przestrzeni rozwiązań problemów przepływowych, potwierdzają występowanie doliny (valley) ekstremów lokalnych, w której metody lokalnych poszukiwań przejawiają cechy korzystniejsze niż operatory genetyczne. Stąd wniosek, że lepsze wyniki można osiągnąć łącząc klasyczne operatory krzyżowania z technikami poszukiwań lokalnych.

Zręczne połączenie wyżej wymienionych technik oferuje quasi-operator krzyżowania przez *wielokrokową fuzję* [306] (MSXF). Dla danych dwóch permutacji rodzicielskich σ oraz τ dostarcza on rozwiązanie potomne mające pewne cechy rodziców. Jedno z rozwiązań, powiedzmy σ, jest traktowane jako źródłowe do wygenerowania trajektorii zmierzającej stochastycznie w kierunku τ w sensie wprowadzonej miary odległości w przestrzeni rozwiązań $d(\alpha, \beta)$. Bardziej szczegółowo, niech $\pi^0, \ldots, \pi^t$ będzie trajektorią z rozwiązaniem bieżącym π^s w kroku s-tym; przyjmujemy $\pi^0 = \sigma$. Dla każdego rozwiązania bieżącego π^s oraz jego sąsiedztwa $\mathcal{N}(\pi^s)$ tworzona jest lista zawiera-

jąca wszystkie rozwiązania $\rho \in \mathcal{N}(\pi^s)$ uporządkowane zgodnie z rosnącymi wartościami miary $d(\pi^s, \rho)$. Rozwiązanie kolejne na trajektorii π^{s+1} wybiera się z tej listy według następującej zasady. Wpierw, wybierane jest losowe rozwiązanie próbne ρ' metodą ruletki z prawdopodobieństwem odwrotnie proporcjonalnym do indeksu rozwiązania na liście. Następnie rozwiązanie to jest *akceptowane losowo* podobnie jak w schemacie SA, to znaczy z prawdopodobieństwem 1 jeśli $\Delta \leqslant 0$ oraz z prawdopodobieństwem $\min\{1, e^{-\Delta/c}\}$ jeśli $\Delta > 0$, gdzie $\Delta = K(\rho') - K(\pi^s)$ oraz c jest pewnym stałym parametrem. Jeśli rozwiązanie próbne nie zostało zaakceptowane, przesuwa się je na koniec listy i powtarza proces wyboru i akceptacji tak długo aż pewne rozwiązanie zostanie wybrane. Wybrane rozwiązanie staje się kolejnym π^{s+1} na trajektorii. Proces jest kontynuowany przez kolejne t kroków, gdzie t jest parametrem. Ostatecznie, za potomka rodziców σ i τ uznaje się to rozwiązanie z trajektorii $\pi^0, \ldots, \pi^t$, które posiada najmniejszą wartość funkcji celu.

Opisany algorytm GA oprócz "klasycznych" parametrów strojących takich jak początkowa wielkość populacji, schemat selekcji, itp. posiada pewną liczbę parametrów pracy operatora MSXF a mianowicie stałe c, t, sąsiedztwo $\mathcal{N}(\pi)$ oraz miarę odległości rozwiązań $d(\alpha, \beta)$.

12.12 Podejście geometryczne

Podejście to opiera się na twierdzeniu o uporządkowania wektorów, patrz bibliografia w pracy [233], które w zastosowaniu do problemu przepływowego z kryterium $C_{\max}$ dostarcza algorytm aproksymacyjny SEV gwarantujący wygenerowanie rozwiązania π^{SEV} o wartości

$$C_{\max}(\pi^{SEV}) \leqslant C_{\max}(\pi^*) + m(m-1) \max_{1 \leqslant i \leqslant m} \max_{1 \leqslant j \leqslant n} p_{ij} \tag{12.35}$$

Twierdzenie stanowi, że dla zbioru $\mathcal{V} = \{v_j : \ j \in \mathcal{J}\}$ d-wymiarowych wektorów takich, że $\sum_{j=1}^{n} v_j = 0$ można wyznaczyć w czasie wielomianowym permutację π elementów zbioru $\mathcal{J}$ taką, że

$$\| \sum_{j=1}^{k} v_{\pi(j)} \| \leqslant d \cdot \max_{1 \leqslant j \leqslant n} \| v_j \|, \ \ k = 1, \ldots, n \tag{12.36}$$

Jego zastosowanie do problemu $F^*||C_{\max}$ wymaga wstępnej modyfikacji danych. Po pierwsze wymaga się, aby dane przykładu problemu przepływowego spełniały warunek $P_i = P_{\max}$, $i = 1, \ldots, m$, gdzie $P_i = \sum_{j=1}^{n} p_{ij}$,

$P_{\max} = \max_{1 \leqslant i \leqslant m} P_i$. Jeśli przykład nie spełnia tego warunku należy zwiększyć wartości odpowiednich p_{ij}. Modyfikacja ta nie wpływa na gwarancję otrzymanego oszacowania, zaś końcową wartość $C_{\max}(\pi^{SEV})$ należy wyznaczyć dla wyjściowych danych p_{ij}. Drugi krok polega na transformacji p_{ij} do wektorów v_j przez podstawienie $v_j = (p_{1j} - p_{2j}, p_{2j} - p_{3j}, \ldots, p_{m-1,j} - p_{mj})$. Stąd $d = m - 1$.

Algorithm SEV
Krok 1: Jeśli $n \leqslant d$ to wybierz dowolną permutację na zbiorze $\mathcal{J}$ oraz zwróć ją jako π^{SEV}.
Krok 2: Jeśli $n > d$ to wywołaj VECTOR-SUM-ROUTINE, który dostarczy zbiorów $\mathcal{J} \stackrel{\text{def}}{=} \mathcal{I}_n \subseteq \ldots \subseteq \mathcal{I}_d$.
Krok 3: Wybierz dowolną permutację na zbiorze $\mathcal{I}_d$ i jej elementy wstaw na pierwszych d pozycjach w π^{SEV}.
Krok 4: Dla $j = d+1, \ldots, n$ podstaw $\pi^{SEV}(j) = k$, gdzie k jest unikalnym elementem w zbiorze $\mathcal{I}_j \setminus \mathcal{I}_{j-1}$.

Częścią składową Algorytmu SEV jest pomocnicza procedura, wyznaczająca uporządkowanie wektorów w oparciu o wymienione powyżej twierdzenie.

VECTOR-SUM-ROUTINE

Dla każdej iteracji k rozważmy n-wymiarowy wektor λ^k, gdzie każda składowa λ_j^k jest związana z wektorem $v_j \in \mathcal{V}$.

Krok 1. Podstaw $\lambda_j^n = (n-d)/n$, $j \in \mathcal{J} \stackrel{\text{def}}{=} \mathcal{I}_n$.
Krok 2. Dla $k = n-1, n-2, \ldots, d$ wykonaj Krok 3 i 3a.
Krok 3. Jeśli λ^{k+1} ma składową zerową, tzn. $\lambda_{j^*}^{k+1} = 0$ dla pewnego j^*, to podstaw $\mathcal{I}_k = \mathcal{I}_{k+1} \setminus \{j^*\}$, $\lambda_j^k = \lambda_j^{k+1}$, $j \in \mathcal{I}_k$.
Krok 3a. Wykonaj jeśli λ^{k+1} nie posiada składowej zerowej. Wprowadź zmienną $x = \{x_j,\, j \in \mathcal{I}_{k+1}\}$. Znajdź punkt ekstremalny x^* problemu

$$\sum_{j \in \mathcal{I}_{k+1}} x_j v_j = 0 \tag{12.37}$$

$$\sum_{j \in \mathcal{I}_{k+1}} x_j = k - d \tag{12.38}$$

$$0 \leqslant x_j \leqslant 1, \quad j \in \mathcal{I}_{k+1} \tag{12.39}$$

Można pokazać, że co najmniej jedna zmienna x_j^*, $j \in \mathcal{I}_{k+1}$ ma wartość zero i niech $x_{j^*}^* = 0$. Podstaw $\mathcal{I}_k = \mathcal{I}_{k+1} \setminus \{j^*\}$, $\lambda_j^k = x_j^*$, $j \in \mathcal{I}_k$.

Implementacja algorytmu SEV jest nietrywialna i posiada pewną liczbę punktów swobody. Pozwalają one uzyskiwać, przy zachowaniu teoretycznego

oszacowania najgorszego przypadku, różną dokładność i złożoność obliczeniową w analizie eksperymentalnej. Już wstępna modyfikacja danych wejściowych w celu spełnienia warunku $P_i = P_{\max}$, $i = 1, \ldots, m$ może być zrealizowana w różny sposób. Dalej zauważmy, że Krok 1 algorytmu SEV oraz wstępny Krok 2 powoduje wybór pierwszych $\min\{n, d\}$ pozycji w permutacji π^{SEV} całkowicie arbitralnie. Następnie, w kroku procedury VECTROR-SUM-ROUTINE, w którym wybieramy wektor j^* o zerowej składowej zwykle istnieje więcej niż jeden element spełniający podane własności; sposób wyboru takiego wektora ma także znaczenie. Problem pomocniczy może być rozwiązany przy użyciu różnych technik. Proponowane są dwa podejścia: programowanie liniowe (PL) poprzez wprowadzenie "sztucznej" funkcji celu lub specjalizowany algorytm o złożoności $O(d^2m^2)$.

Przekształcenie w zadanie PL wymaga sprecyzowania funkcji celu. Proponowane są m.in.: (1) $z(x) = 0$, tzn. funkcja stała równa zero, (2) $z(x) = \sum_{j \in \mathcal{I}_{k+1}} x_j$, (3) $z(x) = \sum_{j \in \mathcal{I}_{k+1}} R_j x_j$, gdzie R_j jest liczbą losową o rozkładzie $U(0, 1)$, (4) dowolne inne. Zadanie PL można rozwiązać procedurą cplex. Liczba rozwiązywanych zadań PL jest w praktyce dużo mniejsza niż $n - d$ bowiem rozwiązanie x^* pojedynczego zadania PL zwykle dostarcza pewien większy zbiór składowych zerowych $\mathcal{K} = \{j : \ x_j^* = 0\}$, z którego wybierane są elementy j^* w Kroku 3 w kolejnych iteracjach; następne wywołanie procedury PL jest realizowane dopiero po wyczerpaniu elementów tego zbioru. Wybór elementów z $\mathcal{K}$ odpowiada wstępnemu uporządkowaniu zbioru przy użyciu pewnej reguły: (a) malejących wartości norm $\| v_j \|_\infty$, $j \in \mathcal{K}$, (b) malejących wartości norm $\| v_j \|_2$, $j \in \mathcal{K}$, (c) losowo, (d) stosując pomocniczy algorytm przybliżony do oceny wyniku uszeregowania częściowego zadań z $\mathcal{K}$; polecany jest algorytm NEH, (e) optymalnie wśród wszystkich permutacji na zbiorze $\mathcal{K}$ z kryterium pomocniczym $\| \sum_{j=1}^{k} v_j \|$.

Niech q będzie łączną liczbą zadań LP rozwiązywanych w trakcie pracy algorytmu SEV, dostarczających zbiorów $\mathcal{K}_1 \ldots, \mathcal{K}_q$ z zerowymi składowymi w podanej kolejności. Realizacja wariantu (e) oparta na ocenie wartości funkcji celu dla permutacji częściowej odpowiednich zadań jest NP-trudna, zaś wynik silnie zależy od historii, tj. od zbiorów poprzedzających $\mathcal{K}$ i ich uporządkowań. Nieco łatwiejszy do analizy jest wariant bazujący na podanej normie sum częściowych. Sprowadza się on do wyznaczenia permutacji złożonej $\pi = \pi_1 \pi_2 \ldots \pi_q$ takiej, że

$$\min_{\pi} \max_{1 \leqslant k \leqslant n} \| \sum_{j=1}^{k} v_{\pi(j} \|, \tag{12.40}$$

gdzie $\pi_i = (\pi_i(1), \ldots, \pi_i(k_i))$ jest permutacją na zbiorze $\mathcal{K}^i$, $k_i = |\mathcal{K}_i|$. Pro-

blem ten można dekomponować na ciąg problemów pomocniczych postaci

$$\min_{\pi_i} \max_{1 \leqslant k \leqslant k_i} \parallel v_0 + \sum_{j=1}^{k} v_{\pi_i(j)} \parallel \tag{12.41}$$

gdzie $v_0 = \sum_{j=1}^{t} v_{\sigma(j)}$, $\sigma = \pi_1 \pi_2 \ldots \pi_{i-1}$, $t = \sum_{j=1}^{i-1} k_i$. Permutację minimalizującą można otrzymać poprzez złożenie (konkatenację) permutacji częściowych π_i^* otrzymanych przez rozwiązanie problemów pomocniczych dla $i = 1, \ldots, q$. Pomijając techniką przeglądu zupełnego, wymagającą czasu rzędu $O(\sum_{j=1}^{q} k_i!)$, można skonstruować efektywniejszy algorytm programowania dynamicznego o złożoności $O(\sum_{i=1}^{q} k_i^2 2^{k_i})$. Wykorzystuje on zależność rekurencyjną postaci

$$C(L, i) = \max\{\min_{j \in L \setminus \{i\}} C(L \setminus \{i\}, j), \parallel v_0 + \sum_{k \in L} v_k \parallel\}, \tag{12.42}$$

liczoną dla $i \in L$ oraz $L \subseteq \mathcal{K}_i$, z zerowymi warunkami początkowymi.

Analiza eksperymentalna nie potwierdza by algorytm SEV był konkurencyjny do NEH w sensie jakości generowanych rozwiązań, choć dla dużych n $n \geqslant 1000$ oferuje krótszy czas obliczeń. Nie stwierdzono także istotnych różnic pomiędzy implementacjami algorytmu SEV. Mimo iż relacja algorytmu SEV do innych znanych algorytmów nie została dokładnie zbadana, błąd do NEH na poziomie 5-25% sugeruje wynik negatywny tej oceny.

12.13 Podejście sieciowo-neuronowe

Opisana metoda [228] może być stosowana do problemów, w których rozwiązanie jest reprezentowane permutacją (w tym również przepływowych problemów permutacyjnych), niezależnie od postaci funkcji celu oraz ograniczeń. Nie jest ona realizacją sprzętową algorytmu w sensie *stricte* choć odwołuje się do analogii mających swoje źródło w technice sieci neuronowych. Czytelnik łatwo zauważy, że programowa implementacja algorytmu jest możliwa, co więcej bardziej uzasadniona i nieskomplikowana. Algorytm, zwany dalej NN, jest adaptacyjny i dedykowany do pracy w systemach czasu rzeczywistego przy dynamicznym napływie zadań z pewnego ustalonego a priori zbioru zadań produkcyjnych.

Wybór kolejności wprowadzania zadań oczekujących w kolejce do systemu (permutacja indeksów zadań) jest dokonywany on-line przez sztuczną sieć neuronową wstępnie wytrenowaną. Uczenie sieci przeprowadzane jest w trybie off-line oraz w trakcie bezczynności sieci w oparciu o losowe lub

Rysunek 12.9: Architektura systemu szeregowania.

realne przykłady problemów, w szczególności w oparciu o dane zadań charakterystycznych dla systemu wytwórczego, w którym sieć zainstalowano. Celem uczenia jest wytworzenia w pamięci długoterminowej skumulowanej reprezentacji wiedzy o optymalnych decyzjach szeregowania zadań. Pamięć długoterminowa stanowi bazę wiedzy dla operacyjnej pamięci krótkoterminowej uczestniczącej bezpośrednio w podejmowaniu decyzji o postaci uszeregowania. Architektura kompletnego systemu przedstawiona jest na Rys. 12.9.

Uczenie sieci

Uczenie przebiega z udziałem eksperta (E), którego funkcję spełnia algorytm dokładny danego problemu lub algorytm przybliżony o dużej dokładności, np. zaprojektowany w technice B&B, TS, SA, GA. Generowany jest ciąg k przykładów uczących (np. seria losowych instancji) o rozmiarze n zadań m maszyn. Każdy z tych przykładów jest rozwiązywany algorytmem E dostarczając rozwiązania wzorcowego π^s wraz z odpowiadającą mu wartością kryterium, np. $C_{\max}(\pi^s)$, $s = 1, \ldots, k$. W opisanym przypadku uczenie polega na wytworzeniu macierzy wag $W_{n\times n}$ dla połączeń neuronów, mianowicie

$$W = W^1 + W^2 + \ldots + W^k \tag{12.43}$$

gdzie W^s jest macierzą $n \times n$ stworzoną dla s-tego przykładu uczącego. Proponuje się zero-jedynkowe wartości elementów macierzy W^s, to znaczy

$$W^s_{\pi^s(j)\pi^s(j+1)} = 1, \;\; j = 1, \ldots, n-1, \tag{12.44}$$

oraz zero w pozostałych przypadkach. Taka definicja wag odpowiada znanej technice wzmacniania korzystnych połączeń neuronalnych, w tym przypadku wynikających z kolejności bezpośredniego występowania po sobie zadań w rozwiązaniu optymalnym lub bliskim optymalnemu. Jeśli długość ciągu uczącego zmierza do nieskończoności to macierz W powinna raczej reprezentować częstości zamiast wartości bezwzględnych.

Generowanie rozwiązania

Niech $\mathcal{I} \subseteq \mathcal{J}$ będzie zbiorem zadań aktualnie oczekujących w kolejce do systemu (podlegających uszeregowaniu). Naszym celem będzie ustalenie sekwencji wejściowej zadań reprezentowanej permutacją σ elementów z $\mathcal{I}$. Bez

Rysunek 12.10: Budowa sieci neuronowej.

straty ogólności rozważań możemy założyć, że $\mathcal{I} = \mathcal{J}$. Proponuje się zastosowanie sieci neuronowej dwu-warstwowej, dwu-kierunkowej, posiadającej n neuronów w każdej warstwie, przy czym każdy neuron jednej warstwy posiada połączenia z wszystkimi neuronami warstwy drugiej, Rys. 12.10. Warstwy te oznaczono odpowiednio P (poprzednik) oraz S (następnik). Dokładniej, połączenia mają postać łuku skierowanego (i, j), $j \neq i$, $i \in P$, $j \in S$, przy czym połączeniom przypisano wagi u_{ij}; poczatkowo $u_{ij} = W_{ij}$ oraz wagi u_{ij} mogą ulegać zmianie w trakcie pracy sieci. Dodatkowo wprowadzamy zwrotne łuki skierowane (j, i), $j \in S$, $i \in P$, $i = j$. Permutacja σ odpowiada cyklowi połączeń pomiędzy neuronami $(\sigma(j), \sigma(j+1))$, $\sigma(j) \in P$, $\sigma(j+1) \in S$, $j = 1, \ldots, n-1$, $(\sigma(n), \sigma(1)$, w którym połączenia zamykające cykl wykonywane są przez krawędzie zwrotne. Proces generowania rozwiązania wymaga ogólnie kilku iteracji pracy sieci, przy czym każda iteracja obejmuje dwie fazy odpowiadające pobudzeniu warstwy P oraz "odpowiedzi" warstwy S, po których następuje analiza dopuszczalności uzyskanego rozwiazania. Niedopuszczalność uzyskanego rozwiazania implikuje korekcję wag oraz kolejną iterację. W pierwszej fazie pobudzane są wszystkie neurony z P. Pobudzenie neuronu $i \in P$ jest propagowane przez dokładnie jedno połączenie, o największej wadze u_{ij}, spośród tych wychodzących z i i prowadzących do warstwy S (strategia "zwycięzca bierze wszystko"). W konsekwencji pewne neurony w warstwie S (niekoniecznie wszystkie) mogą uzyskać pobudzenie pochodzące od jednego lub kilku neuronów z P. W fazie drugiej każdy pobudzony neuron warstwy $j \in S$ generuje "odpowiedź", wybierając połączenie po tylko jednym z wcześniej aktywowanych połączeń, mianowicie to posiadające największą wagę u_{ij} spośród połączeń aktywnych kończących się w j. Rozstrzyganie niejednoznaczności dokonywane jest losowo (rywalizacja). W końcowym efekcie, dla każdego neuronu z S co najwyżej jedno połączenie pochodzące z pewnego neuronu z P staje się aktywne (pozwala na propagację pobudzenia). Pobudzone neurony w warstwie S skojarzone z odpowiednimi neuronami w warstwie P, do których powróciła odpowiedź, wyznaczają pewną sekwencję poprzedzania zadań. Może ona być kompletną (zawiera wszystkie zadania) lub częściową oraz dopuszczalną lub też niedopuszczalną (tworzącą lokalne cykle poprzedzań). Jeśli sekwencja ta jest częściowa i dopuszczalna to wszyskie połączenia utworzone w tym kroku stają się trwałe. Dla każdego trwałego połączenia (i, j), $i \in P$, $j \in S$ przyjmujemy $u_{is} = 0$, $s = 1, \ldots, n$, $s \neq j$ oraz $u_{sj} = 0$, $s = 1, \ldots, n$, $s \neq i$, aby zapobiec aktywacji innych ich połączeń w kolejnej iteracji. Jeśli natomiast połączenia te two-

rzą niedopuszczalną sekwencję poprzedzania, to wagę dowolnego (jednego), wybranego losowo połączenia ustalamy na pewnym niskim poziomie, aby zapobiec jego powtórnemu wyborowi. Iteracje pracy sieci są powtarzane do chwili uzyskania kompletnej dopuszczalnej sekwencji poprzedzania. Jest ona reprezentowana cyklem połączeń w sieci z Rys. 12.10 obejmujacym wszystkie neurony obu warstw. Ponieważ otrzymana sekwencja poprzedzania odpowiada pojedynczemu cyklowi poprzedzania zawierającemu wszystkie zadania i nie precyzuje, które konkretne zadanie ma byc pierwsze w σ, sprawdzane jest n permutacji otrzymanych przez cykliczne przesuwanie sekwencji poprzedzania w celu wyznaczenia permutacji z najmniejszą wartością funkcji celu.

Istnieje kilka zalet omawianego podejścia: duża szybkość pracy, możliwość "dostrojenia" do specyficznej klasy przykładów praktycznych, do problemów z różnymi kryteriami i ograniczeniami, oraz możliwość implementacji sprzętowej. Jest oczywistym, że jakość tego podejścia zależy od długości i jakości procesu uczenia. Badania eksperymentalne przeprowadzone w pracy [228] dla losowej serii przykładów pokazują dość szybkie uczenie się sieci oraz stabilizację wiedzy dla większych długości ciągów uczących. Niestety zbyt mały zakres wykonanych tam badań dla problemu $F^*||C_{\max}$ nie gwarantuje odpowiedniej wiarygodności wyciągniętych wniosków. Błąd względny na poziomie 1.5-2.5% do rozwiązania optymalnego dla problemu $F^*2||C_{\max}$ był osiągany po wytrenowaniu ciągiem uczącym zawierającym 160 przykładów dla problemów o rozmiarze $n \leqslant 45$. Chociaż pomysł zastosowania NN dla problemu $F^*2||C_{\max}$ wydaje się kontrowersyjny, to jest on całkiem uzasadniony już dla problemu $F^*2||\sum C_i$ oraz jego pochodnych. Dla 10 losowo wybranych przykładów problemu $F^*||C_{\max}$ o większym rozmiarze, $n \leqslant 20$, $m \leqslant 7$, algorytm NN był najszybszy spośród podejść DS, NEH (przy nieefektywnej implementacji NEH), GA oraz najdokładniejszy wśród wielu algorytmów kostrukcyjnych, DS i NEH. Niedoścignione jakością wyniki otrzymywane przez GA skłoniły autorów pracy [228] do zaproponowania algorytmu hybrydowego NN+GA, w którym NN generuje "celowaną" populację początkową dla GA zamiast zwykle używanej poczatkowej populacji losowej. Otrzymane wyniki dla NN+GA są lepsze od czystego GA w tym sensie, że otrzymano nieznacznie szybszą zbieżność w początkowej fazie algorytmu hybrydowego (co wydaje się oczywiste) przy nie zmienionym zasadniczo znacznym czasie działania algorytmu GA.

12.14 Uwagi

Rozległość badań dotyczących problemów przepływowych spowodowała utworzenie biblioteki testów numerycznych zawierających szczególnie trudne przykłady o rozmiarze od 1,000 do 10,000 operacji (500 zadan 20 maszyn) [306,358], dostępnej także w Internecie [31]. Ocenę własności nowo projektowanych algorytmów dla problemów tej klasy przeprowadza się poprzez porównanie wyników otrzymywanych przez różne algorytmy dla tych samych testów. Niezależnie od powyższego, pewne inne ogólne tendencje są obserwowane w badaniach klasycznych problemów przepływowych.

Po pierwsze, schemat B&B należało by raczej traktować jako kosztowne narzędzie do generowania dobrych rozwiązań referencyjnych wykorzystywanych przy ocenach algorytmów przybliżonych, niż jako podstawową metodę rozwiązywania. Graniczną liczbą operacji ($n \times m$, poza którą schemat ten przejawia eksplozję obliczeń, wydaje się być obecnie wielkość 500, próg ten wyznacza racjonalne granice stosowalności B&B do rpoblemu $F^*||C_{\max}$.

Po wtóre, dość liczna klasa metod przybliżonych doczekała się w ostatnich latach teoretycznych analiz dokładności, patrz Tab. 12.1, choć nie wszystkie górne ograniczenia współczynnika najgorszego przypadku są osiągalne. Zwykle, ranga algorytmu uzyskana w teoretycznej ocenie najgorszego przypadku odpowiada randze wynikajacej z analizy eksperymentalnej, choć wartości obserwowanych błędów są krańcowo różne. Przykładowo, Algorytm CDS, posiadający teoretyczną gwarancję błędu S^{CDS} równą $\lceil m/2 \rceil$, w praktyce generuje rozwiazania z przeciętnym błędem względnym S^{CDS} około 1.25 , prawie niezależnie od m. Zarówno analiza eksperymentalna jak i teoretyczna potwierdzaja, że w klasie algorytmów konstrukcyjnych najlepszym jest NEH. W praktyce dostarcza on rozwiązań z przeciętnym błędem względnym S^{NEH} w przybliżeniu 1.05. Niektóre algorytmy poszukiwań lokalnych typu DS uzyskały teoretyczne oceny dokładności, patrz algorytmy RACS, RAES w Tab. (12.1), poprzez "zablokowanie" mechanizmu popraw lokalnych, występującego dla pewnych klas przykładów. Jak do tej pory nie ma analiz teoretycznych dla innych klas algorytmów LS, zaś w analizie eksperymentalnej zachowują się one dobrze lub bardzo dobrze.

Uzyskiwanie rozwiązań bardzo bliskich optymalnym jest możliwe tylko przy użyciu iteracyjnych metod przybliżonych, opartych na technikach LS. Opierając się na wynikach literaturowych i pracach własnych, wyciągnięto następujące wnioski dotyczące problemu $F^*||C_{\max}$ i pokrewnych. Rutynowa technika SA nadaje się dobrze do umiarkowanej poprawy jakości rozwiązania lub do problemów o nieznanych (niezbadanych dotychczas) własnościach szczególnych. Dla SA osiągnięcie wysokiej dokładności zwykle pociąga za so-

γ	A	$\underline{\eta}^A$	$\overline{\eta}^A$
$C_{\max}$	dowolny	m	m
	R	$\lceil m/2 \rceil$	$\lceil m/2 \rceil$
	CDS	$\lceil m/2 \rceil$	$\lceil m/2 \rceil$
	RA	$m/\sqrt{2} + O(1/m)$	$m/\sqrt{2} + O(1/m)$
	RACS,RAES	$m/\sqrt{2} + O(1/m)$	$m/\sqrt{2} + O(1/m)$
	P	$m/\sqrt{2} + O(1/m)$	$m/\sqrt{2} + O(1/m)$
	NEH	$m/\sqrt{2} + O(1/m)$	$(m+1)/2$
	HR	$m/\sqrt{2} + O(1/m)$	$m/\sqrt{2} + O(1/m)$
	G	$m-1$	$m-1$
	TG	$(m+1)/2$	$(m+1)/2$
	IE,M	m	m
	KS1,KS2	m	m
	CDS+HC	$m/2$	$\lceil m/2 \rceil$
	T/Yk	$\lceil m/k \rceil \eta^{Yk}(C_{\max})$	$\lceil m/k \rceil \eta^{Yk}(C_{\max})$
$\frac{1}{n}\sum C_i$	dowolny	n	n
	SPT	m	m
	RCo	$2m/3 + 1/3$	m
	RCo,m=2	1.908	2
	RC1,RC2	n	n
	RC3	$2m/3 + 1/(3m)$	n
	RC3,m=2	1.577	n
	HK,m=2	$2b/(a+b)$	$2b/(a+b)$
	KS1,...,KS5	$n-1$	n
	R,CDS+API	$n-(2-4/(n+2))$	n
	CDS+NPI	$n/3$	n
$\sum w_i C_i$	dowolny	$1+(n-1)(\overline{w}/\underline{w})$	$1+(n-1)(\overline{w}/\underline{w})$
	CDS,G,P,RA, CDS+HC,G+HC, P+HC,RA+HC	$1+(n-1)(\overline{w}/\underline{w})$	$1+(n-1)(\overline{w}/\underline{w})$
	F	m	m
	Q/X	$\lceil m/2 \rceil \eta^X(C_{sum})$	$\lceil m/2 \rceil \eta^X(C_{sum})$
	T/Yk	$\lceil m/k \rceil \eta^{Yk}(C_{sum})$	$\lceil m/k \rceil \eta^{Yk}(C_{sum})$

Tabela 12.1: Dolne $\underline{\eta}^A$ i górne $\overline{\eta}^A$ ograniczenia wartości współczynnika najgorszego przypadku η^A algorytmów A dla różnych kryteriów szeregowania γ

bą nieakceptowalnie dlugi czas obliczeń przebiegu, zwielokratniany przez konieczność powtarzania przebiegów ze względu na losowy charakter metody. Podobną opinię można wystawić dla RS, SJ, GS oraz wielu podobnych. Niektóre inne techniki, jak na przykład AS, NN, AIS, jak dotychczas nie stały się konkurencyjne nawet do wspomnianych przed chwilą. Aktualnie zdecydowanie najkorzystniejsze wyniki otrzymuje się dla metod poszukiwań lokalnych opartych na technice TS z wykorzystaniem własności blokowych. Pozwalają one rozwiązywać testy "biblioteczne" z przeciętnym błędem względnym S^{TS} około 1.1 w czasie minut na PC.

13

Zaawansowane problemy przepływowe

Problemy przepływowe z poprzedniego rozdziału stanowią jedynie niewielką próbkę zagadnień o praktycznej stosowalności. Zdecydowania częściej występują problemy nietypowe, z ogólniejszą funkcją celu oraz dodatkowymi ograniczeniami. W tym rozdziale przedstawimy niektóre wyniki dotyczace problemów z addytywną, regularną funkcją celu, problemów z ograniczeniami NS (brak miejsc składowania), LS (buforowanie), ZW (bez czekania), ograniczona pojemność systemu, transport i przezbrojenia, z kryterium opartym na czasie cyklu, oraz innych.

13.1 Kryterium regularne, addytywne

Problemy przepływowe z addytywną funkcją celu wymagają specyficznych podejść i algorytmów, krańcowo odmiennych od tych dla problemów z kryterium $C_{\max}$. Poniżej podamy kilka z nich polecanych jako metody przybliżone. Wiele z tych metod odwołuje się do *czasów oczekiwania zadań*, *przestojów pomiędzy zadaniami*, *dopasowania zadań*, *skojarzenia zadań*, *przerw pomiędzy zadaniami*. Przykładowym algorytmem popraw jest *HC* proponowany w [344] dla problemu z ogólną funkcją kryterialną i testowany dla kryteriów średni czas przepływu, średnie wykorzystanie maszyn oraz długość uszeregowania Algorytm ten opiera się na filozofii przerw pomiędz zadaniami. Chociaż filozofia przerw pochodzi zasadniczo do problemów z kryterium $C_{\max}$, algorytm HC był formułowany i polecany dla dowolnego kryterium regularnego. Zagregowana przerwa pomiędzy zadaniami jest definiowana jako

$d_{ij}^R = \Sigma_{k=1}^{m-1} d_{ij}^k \delta_{ij}^k$, gdzie $d_{ij}^k = p_{k+1,i} - p_{kj}$ jest przerwą oraz

$$\delta_{ij}^k = \begin{cases} 1 & if \quad d_{ij}^k \geqslant 0 \\ 0.9\frac{m-k-1}{m-2} + 0.1 & if \quad d_{ij}^k < 0 \end{cases} \tag{13.1}$$

jest współczynnikiem dyskonta, $k = 1, \ldots, m-1$, $i, j = 1, \ldots$,n. Dzieląc zbiór $\{1, \ldots, m-1\}$ na podzbiory $M_{ij} = \{1 \leqslant k < m : p_{k+1,i} \geqslant p_{kj}\}$ oraz $L_{ij} = \{1 \leqslant k < m : p_{k+1,i} < p_{kj}\}$ możemy wyrazić d_{ij}^R w innej równoważnej postaci, wygodniejszej dla naszych zastosowań

$$\begin{aligned} d_{ij}^R &= \sum_{k \in M_{ij}} d_{ij}^k + \frac{0.9}{m-2} \sum_{k \in L_{ij}} d_{ij}^k (m-2-k+1) + 0.1 \sum_{k \in L_{ij}} d_{ij}^k \\ &= \sum_{k \in M_{ij} \cup L_{ij}} d_{ij}^k - \frac{0.9}{m-2} \sum_{k \in L_{ij}} d_{ij}^k (k-1) \\ &= \sum_{k=1}^{m-1} p_{k+1,i} - \sum_{k=1}^{m-1} p_{kj} - \frac{0.9}{m-2} \sum_{k \in L_{ij}} d_{ij}^k (k-1) \\ &= b_i^{m-1} - a_j^{m-1} - \frac{0.9}{m-2} \sum_{k \in L_{ij}} (k-1)(p_{k+1,i} - p_{kj}) \end{aligned} \tag{13.2}$$

gdzie $a_t^{m-1} = \sum_{s=1}^{m-1} p_{st}, b_t^{m-1} = \sum_{s=1}^{m-1} p_{m-s+1,t}$, $t = 1, \ldots, n$ są czasami wykonywania dla $(m-1)-$szego dwu-maszynowego problemu pomocniczego żywane w algorytmie CDS. Zauważ, że ostatnia suma w (13.2) może być wykonana dla re-definiowanego zbioru $L_{ij} = \{1 < k < m : p_{k+1,i} < p_{kj}\}$ ponieważ element dla $k = 1$ ma wae zero. Algorytm HC może być zapisany w następującej zwartej postaci. Niech $\pi = (\pi(1), \ldots, \pi(n))$ będzie początkową permutacją na $\mathcal{J}$. Oznaczmy przez $\pi_{(x,y)}$ permutację otrzymaną π poprzez wymianę zadań w pozycjach x oraz y. Podstaw początkowo $a = 1, b = n$.

Algorithm HC

1. Jeśli $b = a + 2$ to podstaw $\pi^{HC} := \pi$ i stop.

2. Wyznacz wartości X, Y oraz indeksy g, h $(a < g < b, a < h < b)$ takie, że $X = d_{\pi(a)\pi(g)}^R = \max\{d_{\pi(a)\pi(j)}^R : a < j < b\}$, $Y = d_{\pi(h)\pi(b)} = \min\{d_{\pi(j)\pi(b)}^R : a < j < b\}$.

3. Jeśli $(((X > 0) \vee (Y < 0)) \wedge (|X| > |Y|)) \vee ((X < 0) \wedge (Y > 0) \wedge (|X| \leqslant |Y|))$ to przejdź do 4, inaczej przejdź do 5.

4. Podstaw $a := a + 1$. Jeśli $K(\pi_{(a,g)}) < K(\pi)$ to podstaw $\pi := \pi_{(a,g)}$. Idź do 1.

5. Podstaw $b := b - 1$. Jeśli $K(\pi_{(h,b)}) < K(\pi)$ to podstaw $\pi := \pi_{(h,b)}$. Idź do 1.

W każdej iteracji HC potrzebujemy $2(b - a - 1)$ wartości $d^R_{\pi(a)\pi(j)}$ oraz $d^R_{\pi(j)\pi(b)}$, $j = a + 1, \ldots, b - 1$. Ponieważ w każdej iteracji dokładnie jeden z dwóch indeksów a lub b zmienia swoją wartość, całkowita liczba wartości d^R_{ij} używanych we wszystkich iteracjach jest równa $(n-2)+(n-3)+\ldots+1 = (n^2 - n - 2)/2$. Wartości te mogą być obliczane na żądanie. Algorytm HC ma złożoność obliczeniową $O(n^2m)$ [1] i wymaga pomocniczego algorytmu przybliżonego do przygotowania permutacji początkowej. Wersja oryginalna HC używa w tym celu CDS.

Każdy algorytm popraw (w tym także HC) może być modfikowany na kilka sposobów. Po pierwsze, używając innego szybkiego algorytmu konstrukcyjnego do generacji rozwiązania początkowego – w tym celu proponoweane są specjalne reguły priorytetowe, znane już funkcje priorytetu, lub inne algorytmy konstrukcyjne opisane w poprzednim rodziale. Po wtóre, wspierające procedury popraw [204] API, NPI, INS są dołączane na końcu algorytmu. Chociaż API przejawia poważne wady dla kryterium C_{max}, [27], wciąż pozostaje najbardziej popularną metodą popraw głównie dla innych niż C_{max} kryteriów. (Najbardziej polecanymi dla C_{max} są INS oraz pewne jego poprawione warianty.)

Inne algorytmy przybliżone, oznaczone tutaj $RC1, RC2, RC3, RCo$, zaproponowano dla problemu z kryterium $\overline{C}_{sum}$. Mogą one być interpretowane jako dynamiczne reguły priorytetowe, które operują na zbiorze zadań już uszeregowanych (S) oraz zadań jeszcze nieuszeregowanych ($U = J \backslash S$). Zadania ze zbioru S tworzą permutację częściową σ, oraz niech $C_{i\sigma(d)}$ oznacza termin zakończenia ostatniego zadania w S na maszynie $i, i = 1, \ldots, m, d = |S|$. Permutacja częściowa σj, rozszerzona o zadanie j, gdzie $j \in U$, dostarcza terminów zakończenia tego zadania równych $C_{1j} = C_{1\sigma(d)} + p_{1j}$, $C_{ij} = \max\{C_{i\sigma(d)}, C_{i-1,j}\} + p_{ij}$, $i = 2, \ldots, m,$. Każdy z poniższych algorytmów wybiera jako kolejne do ustawienia bezpośrednio za σ zadanie $k \in U$ z najmniejszą wartością priorytetu:

$RC1 \ : \ \omega'_k = \sum_{i=2}^{m} \max\{C_{i-1,k} - C_{i,\sigma(d)}, 0\}$,

$RC2 \ : \ \omega''_k = \sum_{i=2}^{m} \left| C_{i-1,k} - C_{i,\sigma(d)} \right|$,

[1] Efektywna implementacja NEH jest także $O(n^2m)$. NEH wykazuje w analizie eksperymentalnej dobrą jakość także dla $\overline{C}_{sum}$.

$RC3$: $\omega_k''' = \sum_{i=2}^{m} \left| C_{i-1,k} - C_{i,\sigma(d)} \right| + \sum_{i=1}^{m} C_{i,k}$.

Niejednoznaczności są rozstrzygane przez wybór zadania mającego najwcześniejszy termin zakończenia na ostatniej maszynie. Algorytm RCo wykorzystuje zbiór priorytetowych reguł statycznych.

RCo : Wybierz π^{RCo} taką, że $\overline{C}_{sum}(\pi^{RCo}) = \min_{\pi \in \{\pi^1, \ldots, \pi^m\}} \overline{C}_{sum}(\pi)$, gdzie π^k jest otrzymany przez uporządkowanie zadań w kolejności niemalejących wartości priorytetu $\omega_{kj}^{o} = \sum_{i=1}^{m} \min\{m-i+1, m-k+1\} p_{ij}$.

Algorytmy $RC1, RC2, RC3$ mają złożoność obliczeniową $O(n^2 m)$, podczas gdy RCo ma $O(nm^2)$.

Algorytm $RC1$ jest szczególnym, uproszczonym przypadkiem algorytmu $KS2$. KS2 należy do grupy reguł dynamicznych opartych na pojęciu opóźnień. Niech σ, j, d, $C_{i\sigma(d)}$, C_{ij} będą wielkościami jak dla algorytmów RC1–RC3. *Najpóźniejszy* termin zakończenia zadania j jest równy $D_{ij} = C_{mj} - \sum_{s=i+1}^{m} p_{sj}$. Jeśli $C_{i\sigma(d)} < D_{ij} - p_{ij}$ to mówimy, że istnieje *opóźnienie* pomiędzy $\sigma(d)$ orz j na maszynie i. Ponieważ termin rozpoczęcia zadania j na maszynie i może być wybrany z przedziału $[C_{ij} - p_{ij}, D_{ij} - p_{ij}]$, zatem jeśli $C_{ij} < D_{ij}$ to można dokonać redukcji pewnych opóźnień. Niech a_j będzie najmniejszym indeksem i, $1 \leqslant i \leqslant m$ takim, że $C_{i\sigma(d)} = D_{ij} - p_{ij}$, zaś b_j niech będzie największym indeksem spełniającym podany warunek. Oczywiście $a_j \leqslant b_j$. Wielkość $\sum_{s=1}^{a_j - 1} (D_{sj} - p_{sj} - C_{s\sigma(d)})$ jest łącznym *opóźnieniem końcowym*, które może być zmniejszone do wartości $\sum_{s=1}^{a_j - 1} (C_{sj} - p_{sj} - C_{s\sigma(d)})$ przez odpowiedni wybór terminów rozpoczęcia zadania j na maszynach $1, \ldots, a_j - 1$. Wartość $\sum_{s=b_j+1}^{m} (C_{sj} - p_{sj} - C_{s\sigma(d)})$ jest łącznym *opóźnieniem początowym*, które jednakże nie może być zredukowane. W literaturze zaproponowano także następujące reguły dynamiczne wybierające zadanie k z minimalną wartością priorytetu:

$KS1$: $\omega_k^{KS1} = \sum_{i=b_k+1}^{m} (C_{ik} - p_{ik} - C_{i\sigma(d)})$ (całkowite opóźnienie początkowe),

$KS2$: $\omega_k^{KS2} = \sum_{i=1}^{m} (C_{ik} - p_{ik} - C_{i\sigma(d)})$ (całkowite opóźnienie pomiędzy zadaniami),

$KS3$: $\omega_k^{KS3} = \sum_{i=1}^{a_k - 1} (D_{ik} - p_{ik} - C_{i\sigma(d)})$ (całkowite opóźnienie końcowe),

$KS4$: $\omega_k^{KS4} = \sum_{i=2}^{m} i(C_{ik} - p_{ik} - C_{i\sigma(d)})$ (całkowite ważone opóźnienie pomiędzy zadaniami),

$KS5$: $\omega_k^{KS5} = -\sum_{i=1}^{m} (D_{ik} - C_{ik})$ (maksymalna oszczędność przy przesunięciu w lewo).

W celu uzyskania lepszej jakości rozwiązania algorytmy są nastepnie modyfikowane tak by każde zadanie było początkowym zadaniem sekwencji, tzn. algorytm jest powtarzany n-razy, startując z permutacji $\sigma = (j)$ dla kolejnych $j = 1, \ldots, n$. Algorytmy te mają złożoność $O(n^3 m)$.

Ostatnio zaproponowano algorytm R dla wielokryterialnej wersji problemu z kryteriami C_{max} oraz $\overline{C}_{sum}$. Algorytm ten składa się zasadniczo z trzech faz: (1) startuje z π^{CDS}, (2) stosuje procedurę API aby poprawić π^{CDS} używając kryterium C_{max}, (3) stosuje ograniczoną roceduręę API, która akceptuje tylko te rozwiązania, które popawiają kryterium C_{max} lub $\overline{C}_{sum}$. Szczegóły fazy (3) nie będą dyskutowane dokładniej, bowiem nie zmieniają one oceny najgorszego przypadku, lecz jedynie ocenę eksperymentalną.

Złożenie algorytmów A oraz B gdzie algorytm B startuje z permutacji generowanej przez algorytm A, będziemy oznaczać $A + B$.

13.2 Modelowanie dodatkowych ograniczeń

W praktyce obok problemów przepływowych odpowiadających modelom postawowym rozpatrywanym w Rozdz. 12 pojawiają się problemy bardziej złożone. Najczęściej są one pochodnymi zagadnienia podstawowego otrzymanymi poprzez wprowadzenie dodatkowych ograniczeń o różnym charakterze. Niekiedy jeden problem posiada równocześnie kilka klas (rodzajów) ograniczeń. Niektóre z nich mogą być modelowane poprzez nieznaczne rozszerzenie modelu podstawowego przy równoczesnym zaadaptowaniu modelu grafowego. Znajomość tych technik jest zwykle pomocna zarówno do badania szczególnych własności problemu jak i do projektowania odpowiedniego algorytmu rozwiązywania. Biorąc pod uwagę, że podane poniżej szczególne problemy szeregowania mają zbiór rozwiązań reprezentowany zbiorem permutacji, szereg opisanych poprzednio metod (np. SA, SJ, TS, GS) może być bezpośrednio zastosowanych do rozwiązywania. Złożoność obliczeniowa odpowiednich problemów szczególnych zawarta została w Tab. **??**.

Relacja poprzedzań

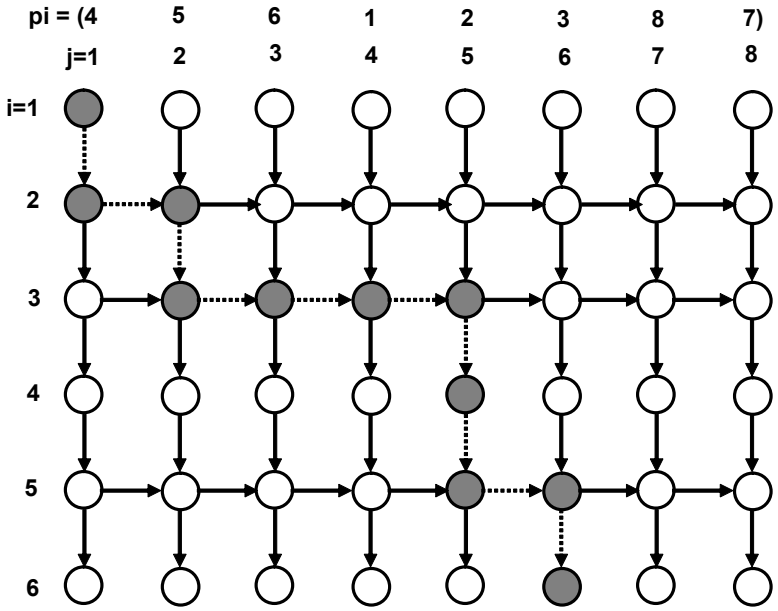
Wprowadzenie technologicznej relacji *poprzedzania* $\mathcal{R}$ zadań na każdej maszynie w formie $C_{ij} \leqslant S_{ik}$, $i = 1, \ldots, m$, $(j, k) \in \mathcal{R}$, implikuje, że permutacje *dopuszczalne* muszą być *porządkami topologicznymi* [69] w grafie skierowanym $(\mathcal{J}, \mathcal{R})$. Wyznaczenie wartości funkcji celu dla danej dopuszczalnej π oraz model grafowy $G(\pi)$ z Rys. 12.1 nie ulegają zmianie. Sprawdzenie dopuszczalności danej permutacji π można wykonać w czasie $O(|\mathcal{R}|)$. Wprowadzenie relacji zwykle poprawia szybkość zbieżności algorytmów (takich

jak na przykład B&B, LS, DS, TS, AMS) pozwalając eliminować znaczne liczby rozwiązań niedopuszczalnych. Z tego też względu pokładano duże, lecz niespełnione, nadzieje w tak zwanych kryteriach eliminacyjnych, jako narzędzia wspomagającego rozwój schematu B&b. Dalej, dla $\mathcal{R} \neq \emptyset$ metody DS, SA, TS, AMS zachowują się istotnie korzystniej, lecz mogą tracić własność *styczności*. W innych metodach, takich jak na przykład GS, pewien problem może sprawiać generowanie permutacji dopuszczalnych wymagające z kolei zaprojektowania specyficznych operatorów genetycznych.

Maszyny o nieograniczonej przepustowości

Pojęcie *przepustowości* maszyny jest wygodnym narzędziem do modelowania i analizowania niektórych procesów rzeczywistych, szczególnie w kontekście systemów sterowania OPT, strategii SQUEZEE, oraz relaksacji prowadzących do konstrukcji dolnych ograniczeń głównie w schemacie B&B. Mówimy, że maszyna ma przepustowość k jeśli w dowolnym momencie czasu może wykonywać nie więcej niż k zadań równocześnie (definicja to odpowiada pojęciu m-procesora z pracy [364]). Pojedyncza maszyna tradycyjnie ma przypisana przepustowość jednostkową. Maszynę o przepustowości k wygodnie jest interpretować jako stanowisko zawierające k identycznych maszyn równoległych. W obu przypadkach wymagane jest szeregowanie zadań do przed stanowiskiem. Z kolei za maszynę o *nieograniczonej przepustowości* można uważać:

- urządzenie, które w sensie fizycznym pozwala obsługiwać jednocześnie dowolnie wiele zadań, np. piec grzewczy,
- maszynę o ograniczonej przepustowości, dla której czas trwania jest nieporównywalnie mały w stosunku do maszyn sąsiednich przez co nie obserwuje się występowania kolejki,
- zbiór urządzeń fizycznych, identycznych funkcjonalnie i o tak dużej liczności, że operacje realizowane na tych urządzeniach nie muszą być szeregowane,
- maszynę otrzymaną poprzez zagregowanie (połączenie) podzbioru kolejnych maszyn o nieograniczonej przepustowości,
- maszynę fikcyjna realizującą proces wymagający upływu czasu lecz nie angażujący urządzenia w sensie fizycznym (operacje starzenia, schnięcia, dojrzewania, chłodzenia, itp.).



Rysunek 13.1: Modelowanie maszyn o nieograniczonej przepustowości

Ostatnia interpretacja pozwala traktować termin gotowości r_j (head) zadania implikujący ograniczenie $r_j \leqslant S_{1j}$, $j \in \mathcal{N}$, jak wykonywanie fikcyjnej operacji zadania na maszynie o nieograniczonej przepustowości, o czasie trwania operacji równym terminowi gotowości. Podobnie żądany terminu zakończenia d_j zadania występujące w kryteriach $L_{\max}$, $T_{\max}$, itp. pozwala poprzez przekształcenie $qj = d_j - K$, gdzie $K = \max_{1 \leqslant j \leqslant n} d_j$ sprowadzić problem do tego z kryterium $C_{\max}$. Istotnie bowiem $L_{\max} = \max_{1 \leqslant j \leqslant n}(C_j + q_j) - K$. Wartość q_j (tail) może być traktowana jako czas wykonywania operacji na maszynie o nieograniczonej przepustowości.

Problemy zawierające maszyny o nieograniczonej przepustowości zachowują swoje własności związane ze ścieżką krytyczną w grafie (dla problemów $C_{\max}$, $L_{\max}$, $f_{\max}$, itp) oraz własności eliminacyjne dotyczące bloków zadań.

Transport

System transportu przemieszcza zadania pomiędzy maszynami. W wielu przypadkach praktycznych przepustowość systemu jest tak duża, że można go traktować jak maszynę o nieograniczonej przepustowości. Jednak zamiast modelować transport techniką opisaną w punkcie poprzednim, co zwiększa niepotrzebnie rozmiar modelu grafowego, wystarczy obciążyć krawędzie

"pionowe" grafu $G(\pi)$ wartościami t_{ij}, $i = 1, \ldots, m-1$, $j = 1, \ldots, n$, gdzie t_{ij} czas transportu zadania j pomiędzy maszynami i oraz $i+1$. Transport przygotowawczy t_{0j} oraz zakończeniowy t_{mj} może być modelowany bądź przy użyciu techniki opisanej poprzednio, bądź jako obciążenie dodatkowej krawędzi $((0,0),(1,j))$ oraz $((m,j),(*,*))$, gdzie $(0,0)$ jest dodatkowym, wspólnym, sztucznym, węzłem początkowym grafu zaś $(*,*)$ odpowiednim węzłem końcowym. Przypadek $t_{ij} \geqslant 0$ ma naturalne uzasadnienie praktyczne. Przypadek $t_{ij} < 0$ odpowiada zezwoleniu na "nakładanie" (overlap) kolejnych operacji zadania, lub też na rozpoczęcie kolejnej operacji zadania z pewnym poślizgiem czasowym w stosunku do rozpoczęcia operacji bieżącej a przed jej ukończeniem (time lag).

Ujemne czasy transportu mogą być usunięte z problemu. Rozważmy przypadek $t_{ij} \neq 0$, $i = 1, \ldots, m-1$, $j \in \mathcal{N}$. Ze zmodyfikowanej wersji wzoru (12.4) mamy

$$C_{i\pi(j)} = \max_{1 \leqslant j_0 \leqslant j_1 \leqslant \ldots \leqslant j_i \leqslant j} \sum_{u=1}^{i} \Big(\sum_{v=j_{u-1}}^{j_u} p_{u\pi(v)} + t_{u\pi(j_u)} \Big). \tag{13.3}$$

gdzie $t_{mj} \stackrel{\text{def}}{=} 0$, $j = 1, \ldots, n$. Niech $T = \min_{1 \leqslant i < m} \min_{1 \leqslant j \leqslant n} t_{ij}$. Stosujac postawienie $t'_{ij} = t_{ij} - T$, otrzymujemy $t'_{ij} > 0$ oraz

$$C_{i\pi(j)} = \max_{1 \leqslant j_0 \leqslant j_1 \leqslant \ldots \leqslant j_i \leqslant n} \sum_{u=1}^{i} \Big(\sum_{v=j_{u-1}}^{j_u} p_{u\pi(v)} + t'_{u\pi(j_u)} - T \Big) = C'_{i\pi(j)} - iT, \tag{13.4}$$

gdzie C'_{ij} jest odpowiednikiem C_{ij} lecz w problemie z czasami transportu t'_{ij}. Ostatecznie otrzymujemy równoważność kryteriów

$$C_{\max}(\pi) = C_{m\pi(n)} = C'_{m\pi(n)} - (m-1)T = C'_{\max}(\pi) - (m-1)T, \tag{13.5}$$

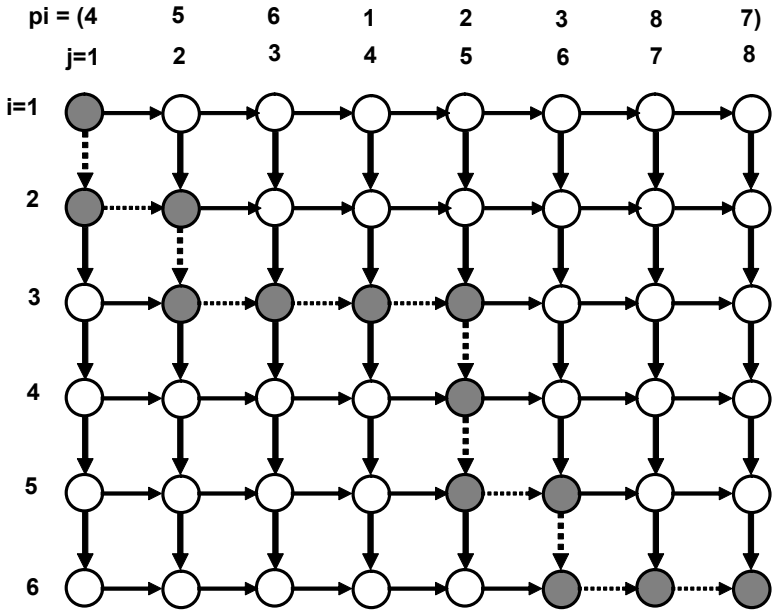
oraz

$$C(\pi) = \sum_{j=1}^{n} C_{m\pi(j)} = \sum_{j=1}^{n} C'_{m\pi(j)} - n(m-1)T = C(\pi)' - n(m-1)T. \tag{13.6}$$

Wartości C_{ij} można policzyć z analogu wzoru (12.3) uwzględniającego czasy transportu

$$C_{i\pi(j)} = \max\{C_{i\pi(j-1)}, C_{i-1,\pi(j)} + t_{i-1,\pi(j)}\} + p_{i\pi(j)}, \quad j = 1, \ldots, n, \tag{13.7}$$

przy czym w przypadku ujemnych t_{ij} należy podstawić $C_{10} = 0$, $C_{i0} = -\infty$, $i = 2, \ldots, m$, $C_{0j} = 0$, $j = 1, \ldots, n$. Trzeba również pamiętać, że warunki te nie gwarantują zachowania nierówności $S_{ij} \geqslant 0$.



Rysunek 13.2: Modelowanie transportu

Problemy zawierające tak rozumiany transport zachowują swoje własności związane ze ścieżką krytyczną w grafie (dla problemów $C_{\max}$, $L_{\max}$, $f_{\max}$, itp) oraz własności eliminacyjne dotyczące bloków zadań.

Buforowanie

Bufor jest miejscem do chwilowego składowania zadania przekazywanego pomiędzy maszynami i jest zwykle związany z maszyną (bufor do maszyny). W obszarze bufora tworzona jest kolejka zadań do obsługi. Pojemność bufora jest rozumiana jako maksymalna liczba zadań, które mogą być składowane równocześnie. Klasyczne problemy szeregowania przyjmują zwykle, że pojemność bufora jest dowolnie duża (nieograniczona). Zadania, które po zakończeniu wykonywania na maszynie nie mogą być przekazane do odpowiedniego bufora pozostają na niej powodując *zablokowanie* tak długo aż pojawi się wolne miejsce w buforze. Logiczne pojęcie bufora może modelować:

- fizyczne urządzenie procesu technologicznego,
- ograniczony fizycznie obszar składowania,

- fikcyjne wymaganie "wymuszajace" przepływ produkcji.

Generalnie, rozpatrywane są dwie kategorie problemów: (1) z buforem o zerowej pojemności (ograniczenie *no store*, strategia NS), (2) z buforami o skończonej, ograniczonej pojemności, różnej dla różnych maszyn (strategia LS). Omówimy te techniki kolejno.

Niech π będzie pewną kolejnością wykonywania zadań. Bufor o zerowej pojemności oznacza, że zadanie $\pi(j)$ po zakończeniu wykonywania na maszynie i pozostanie na niej do chwili C'_{ij}, w której będzie możliwe jego rozpoczęcie na maszynie $i+1$. Stąd oczywiste dodatkowe ograniczenia

$$C_{ij} \leqslant C'_{ij}, \;\; C'_{ij} = S_{i+1,j}, \;\; C'_{i\pi(j)} \leqslant S_{i\pi(j+1)}, \tag{13.8}$$

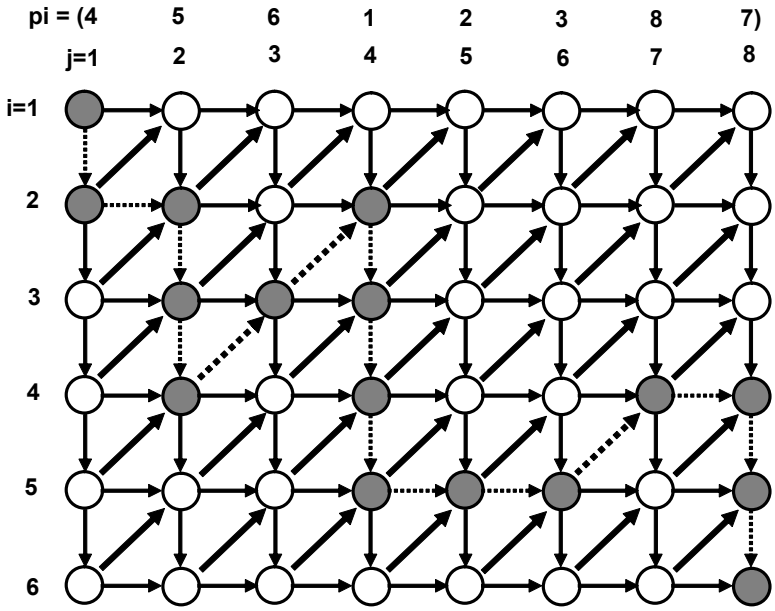
dopełniające znane już warunki (12.1)–(12.2). Eliminując zmienne Cij oraz C'_{ij} z problemu otrzymamy układ warunków koniecznych do wyznaczenia terminów S_{ij} dla danej kolejności π zawierających (12.1)–(12.2) oraz dodatkowo

$$S_{i\pi(j)} + p_{i\pi(j)} - p_{i\pi(j)} \leqslant S_{i+1,\pi(j+1)}, \;\; i = 1, \ldots, m-1, \;\; j = 1, \ldots, n-1. \tag{13.9}$$

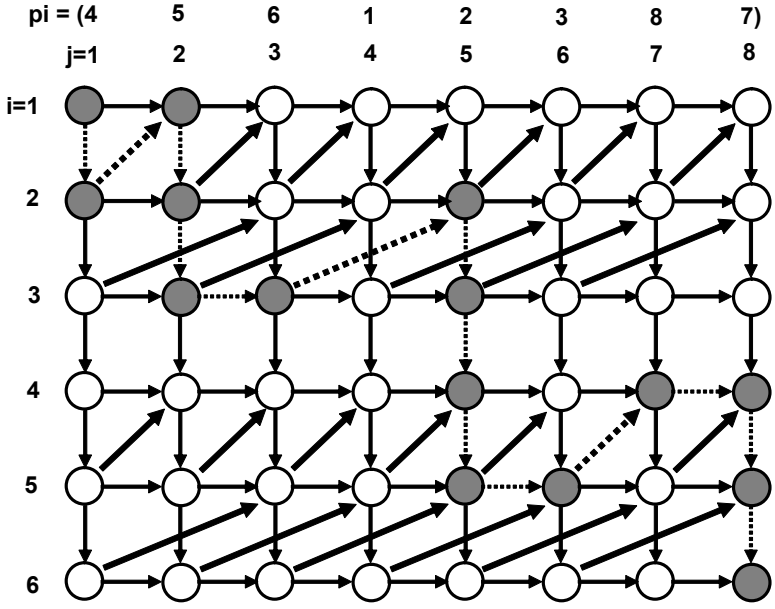
Ograniczenie to jest reprezentowane łukiem ukośnym postaci $((i,j),(i+1,j+1))$ o obciażeniu $-p_{i\pi(j)}$, Rys. **??**. Interpretacja pojęć S_{ij} oraz $C_{\max}(\pi)$ związana z grafem $G(\pi)$ pozostają bez zmian. Można pokazać, że rozwiazanie dopuszczalne problemu przepływowego z buforami o zerowej pojemności musi być permutacyjne.

Dla buforów o pojemności większej niż jeden znaczenia nabiera *reguła obsługi bufora* decydująca, które zadanie jest wybierane do obsługi. Niech $b_i \geqslant 0$ będzie pojemnością bufora przed maszyną i, $i = 2, \ldots, m$. Bufor o pojemności b_i z regułą obsługi FIFO może być modelowany jako ciąg b_i kolejnych fikcyjnych maszyn, o zerowym czasie trwania zadań na nich wykonywanych, pomiędzy którymi nie zezwala się na składowanie. Chociaż ta własność pozwala modelować bufory niezerowe przy pomocy poprzedniego podejścia, otrzymany w ten sposób model ma rozmiar powiększony o $\sum_{i=2}^{m} b_i$ maszyn, co należy uznać za fakt niekorzystny.

Istnieje prostszy sposób modelowania buforów o skończonej pojemności, nie zwiększający rozmiaru problemu. ????? Problemy zawierające maszyny bufory zachowują swoje własności związane ze ścieżką krytyczną w grafie (dla problemów $C_{\max}$, $L_{\max}$, $f_{\max}$, itp) oraz własności eliminacyjne dotyczące bloków zadań. Dodatkowo można sformułować nowe własności zwiazane z istnieniem tzw. bloków ukośnych.



Rysunek 13.3: Modelowanie bufora o zerowej pojemności



Rysunek 13.4: Modelowanie bufora o dowolnej pojemności

Przezbrojenia

Przezbrojeniem nazywamy czas potrzebny na zmianę oprzyrządowania maszyny w związku z wykonywanym zadaniem. Najbardziej ogólny przypadek zakłada, że czas ten zależy od maszyny oraz pary kolejno realizowanych po sobie zadań, to znaczy ma postać s_{ijk}, gdzie $i \in \mathcal{M}$, $j,k \in \mathcal{N}$. Wówczas analogiem wzoru (12.1) jest

$$C_{i\pi(j)} + s_{i\pi(j)\pi(j+1} \leqslant S_{i\pi(j+1)}, \quad j = 1, \ldots, n-1. \tag{13.10}$$

zaś wzór (12.2) pozostaje w mocy. Graf $G(\pi)$ może zostać zaadaptowany poprzez przypisanie do krawędzi $((i,j),(i,j+1))$ wagi $s_{i\pi(j)\pi(j+1)}$. Interpretacja C_{ij} jako długości dróg w $G(\pi)$ pozostaje ważna, podobnie jak interpretacja $C_{\max}$.

Występowanie przezbrojeń w wytwarzaniu pewnej liczby krótkich serii elementów identycznych lub podobnych zmusza do *grupowania* lub *porcjowania* zadań. Przezbrojenia znaczące występują wówczas pomiędzy elementami różnymi, zaś dla elementów identycznych lub podobnych przezbrojenia są nieznaczące lub zerowe.

Dość często w praktyce spotykany jest przypadek przezbrojeń *separowalnych* $s_{ijk} = y_{ij} + x_{ik}$, gdzie y_{ij} jest czasem przestawienia stanowiska i w stan "neutralny" po zakończeniu zadania j, zaś x_{ik} jest czasem przygotowania (od stanu neutralnego) stanowiska i do wykonywania zadania k. Rozpatrywany przypadek jest sprowadzalny do problemu z czasami transportu co świadczy, że przezbrojenia separowalne stanowią jedną z prostszych klas ograniczeń. Istotnie, dla przypadku przezbrojeń terminy rozpoczęcia i zakończenia można wyznaczyć z warunków (13.10) oraz (12.2). Dla przezbrojeń separowalnych mamy $s_{i\pi(j)\pi(j+1)} = y_{i\pi(j)} + x_{i\pi(j+1)}$ co implikuje w oparciu o (13.10)

$$(S_{i\pi(j)} - x_{i\pi(j)}) + (p_{i\pi(j)} + x_{i\pi(j)} + y_{i\pi(j)}) \leqslant (S_{i\pi(j+1)} - x_{i\pi(j+1)}). \tag{13.11}$$

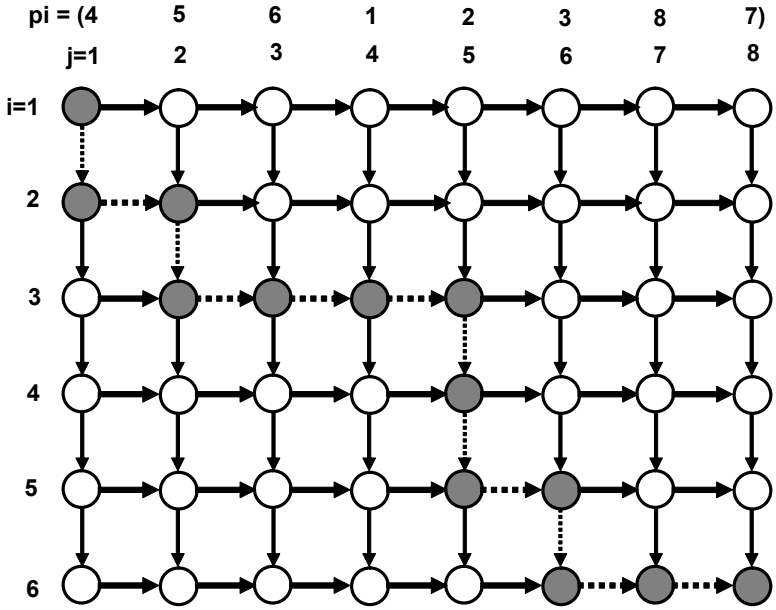
Stosując następnie podstawienia $S'_{ij} \stackrel{\text{def}}{=} S_{ij} - x_{ij}$ oraz $p'_{ij} \stackrel{\text{def}}{=} p_i + x_{ij} + y_{ij}$ z (13.11) otrzymujemy

$$S'_{i\pi(j)} + p'_{i\pi(j)} \leqslant S'_{i\pi(j+1)}, \quad j = 1, \ldots, n. \tag{13.12}$$

Używając tego samego podstawienia we wzorze (12.2) otrzymujemy

$$S'_{ij} + p'_{ij} - (y_{ij} + x_{i+1,j}) \leqslant S'_{i+1,j}, \quad i = 1, \ldots, m-1. \tag{13.13}$$

Warunki (13.12)–(13.13) odpowiadają problemowi z czasami wykonywania zadań powiększonym o przezbrojenia $p_{ij}+x_{ij}+y_{ij}$ oraz z czasami transportu $t_{ij} = -y_{ij} - x_{i+1,j}$.



Rysunek 13.5: Modelowanie przezbrojeń

Problemy zawierające przezbrojenia zachowują niektóre swoje własności związane ze ścieżką krytyczną w grafie (dla problemów $C_{\max}$, $L_{\max}$, $f_{\max}$, itp) pod warunkiem spełnienia własności trójkata. Własności eliminacyjne dotyczące bloków zadań generalnie nie są spełnione. Problemy ze stałymi i separowalnymi przezbrojeniami posiadają własności takie same jak problemy z transportem.

Polityka ZW.

Ograniczenia *bez czekania* (ZW) wynikają bądź z warunków technologicznych prowadzenia procesu bądź są skutkiem celowej polityki zmierzającej do ograniczenia wielkości produkcji w toku oraz wymuszenia jej przepływu. Są pewnym szczególnym przypadkiem ograniczeń czasu oczekiwania zadań, dla kolejnych operacji technologicznych, wprowadzonych w formie

$$a_{ij} \leqslant C_{ij} - S_{i+1,j} \leqslant b_{ij} \tag{13.14}$$

gdzie a_{ij}, b_{ij} są odpowiednio dolnym i górnym ograniczeniem czasu oczekiwania zadania j pomiędzy wykonywaniem w stanowiskach i oraz $i+1$. Ograniczenie ZW odpowiada przypadkowi $a_{ij} = 0 = b_{ij}$ i implikuje warunki $C_{ij} = S_{i+1,j}$, $i = 1, \ldots, m-1$, $j \in \mathcal{N}$. Istnieją co najmniej dwie odmienne

techniki modelowania polityki ZW: (a) przez sprowadzenie do problemu z przezbrojeniami lub problemu komiwojażera (TSP), (b) poprzez jawne modelowanie ograniczeń czasów oczekiwania. Omówimy te techniki kolejno.

Dla dowolnych dwóch zadań j, k, $j \neq k$, $j,k \in \mathcal{N}$, definujemy "wymuszone" opóźnienie zależne od maszyny

$$d_{ijk} = c_{jk} - f_{ij} - g_{ik}, \tag{13.15}$$

gdzie

$$c_{jk} = \max_{1 \leqslant u \leqslant m} (f_{uj} + g_{uk}) \tag{13.16}$$

oraz

$$f_{ij} = \sum_{u=1}^{i} p_{uj}, \;\; g_{ij} = \sum_{u=i}^{m} p_{uj}. \tag{13.17}$$

Wszystkie $n(n-1)(m-1)$ wartości d_{ijk} można wyznaczyc w czasie $O(n^2m)$. Wielkość d_{ijk} określa minimalne opóźnienie terminu rozpoczęcia wykonywania zadania k względem terminu zakończenia zadania j na maszynie i, tak by spełnione zostało wymaganie ZW. Wartości te mogą być traktowane jako szczególny rodzaj przezbrojenia zależnego od sekwencji, Rys. **??**. Dla dowolnej permutacji π istnieje rozwiązanie takie, że

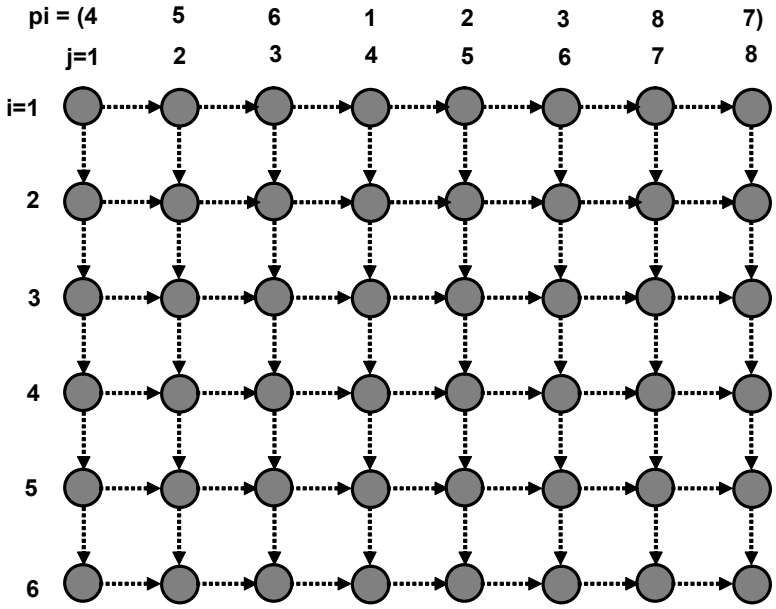
$$C_{i\pi(j)} + d^{i}_{\pi(j)\pi(j+1)} = S_{i\pi(j+1)}, \;\; j = 1,\ldots,n-1, \;\; i = 1,\ldots,m. \tag{13.18}$$

Korzystając z równości (13.18) dla $i = m$ możemy otrzymać szczególny wzór na wartość funkcji kryterialnej

$$\begin{aligned} C_{\max}(\pi) &= \sum_{i=1}^{m} p_{i\pi(1)} + \sum_{j=2}^{n} (d_{m\pi(j-1)\pi(j)} + p_{m\pi(j)}) \\ &= \sum_{j=1}^{n} d_{m\pi(j-1)\pi(j)} + \sum_{j=1}^{n} p_{mj} \end{aligned} \tag{13.19}$$

gdzie $\pi(0) \stackrel{\text{def}}{=} 0$, $d_{0j} \stackrel{\text{def}}{=} \sum_{i=1}^{m-1} p_{ij}$. Ponieważ drugi człon we wzorze (13.19) nie zależy od permutacji π, problem może być sprowadzony do zagadnienia 1-maszynowego z przezbrojeniami d_{mjk} zależnymi od sekwencji. Inaczej, problem można przekształcić także do zadania komiwojażera ze zbiorem miast $\mathcal{N} \cup \{0\}$ oraz zbiorem odległości d_{mjk} przy przejeździe z miasta j do miasta k (m jest stałe). Dla kompletności definujemy $d_{mj0} = 0$, $j \in \mathcal{N}$.

Przedstawione podejście pozwala policzyć wartość funkcj celu dla danej π w czasie $O(n)$, jednak bądź wymaga pomocniczej tablicy d_{mij} o rozmiarze rzędu $O(n^2)$ obliczanej jeden raz. Alternatywnie jeśli wartości d_{mij} są



Rysunek 13.6: Modelowanie polityki ZW jako TSP

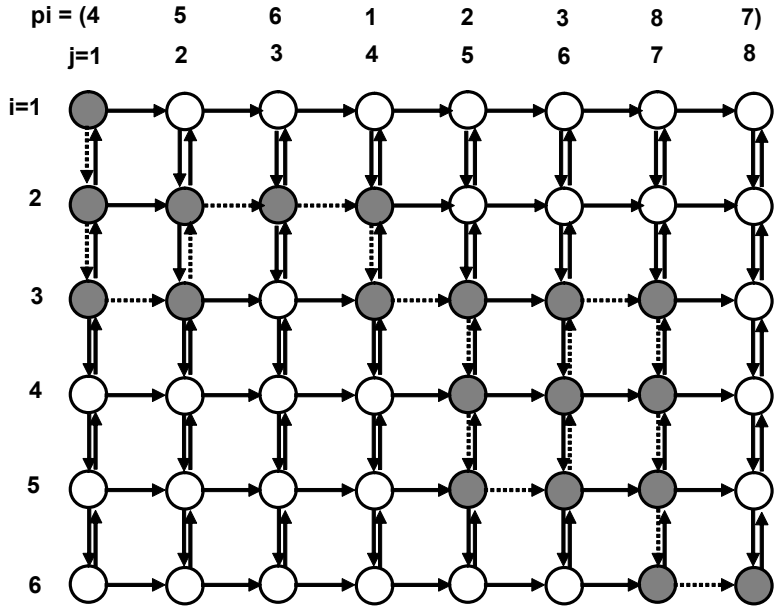
wyliczane na żądanie, koszt wyznaczenia wartości funkcji celu wzrasta do $O(nm)$.

Dla drugiego sposobu modelowania warunek (13.14) zamieniamy na dwie nierówności

$$S_{ij} + p_{ij} - b_{ij} \leqslant S_{i+1,j} \tag{13.20}$$

$$S_{i+1,j} + p_{i+1,j} + (a_{ij} - p_{i,j} - p_{i+1,j}) \leqslant S_{ij} \tag{13.21}$$

Nierówność (13.20) reprezentuje naturalną kolejność technologiczną operacji zadania z ujemnym czasem transportu $-b_{ij}$; dla polityki ZW czas ten jest zerowy. Nierówność (13.20) jest równoważna występowaniu tradycyjnego "pionowego" łuku technologicznego w $G(\pi)$, Rys. (**??**), z obciażeniem $-b_{ij}$. Nierówność (13.21) reprezntuje odwrotną kolejność technologiczną operacji zadania z czasem transportu $a_{ij} - p_{i,j} - p_{i+1,j}$; dla polityki ZW czas ten jest ujemny. Interpretacja grafowa wymaga wprowadzenia odpowiednich dodatkowych łuków w grafie $G(\pi)$, Rys. **??**, z obciażeniem $a_{ij} - p_{i,j} - p_{i+1,j}$. W mocy pozostaje interpretacja terminów C_{ij} jako długości dróg w $G(\pi)$. Ponieważ otrzymany graf posiada cykle o zerowej długości, nie można zastosować algorytmu Belmanna do wyznaczania długości dróg co z kolei powoduje, że nie można zaadoptować prosto wzoru (12.3). Należy w tym przypadku posłużyć się algorytmem Belmanna-Forda [69].



Rysunek 13.7: Modelowanie polityki ZW jako ograniczenia oczekiwania

Ostatnie podejście jest ogólniejsze od poprzedniego bowiem pozwala modelować dowolne ograniczenia czasów oczekiwanania, jednak koszt obliczania wartości funkcji celu jest w tym przypadku zawsze rzędu $O(nm)$.

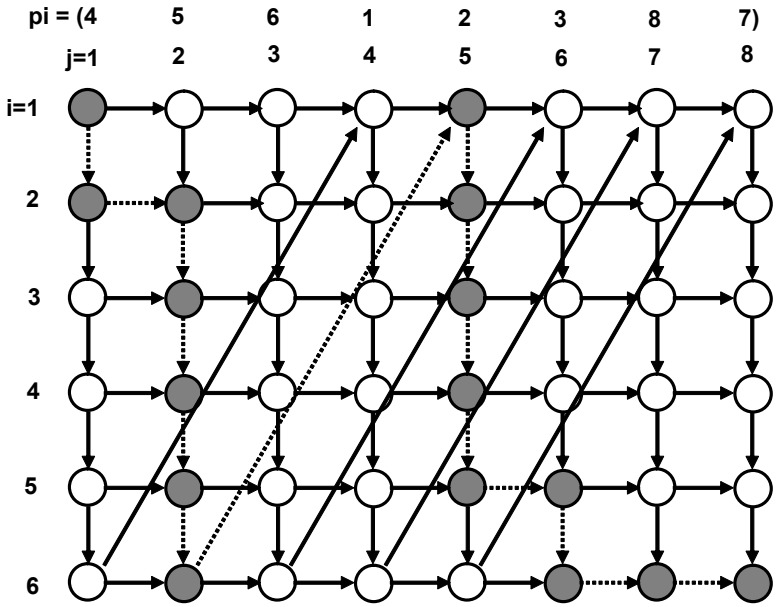
Problemy z polityką ZW posiadają własności takie same jak problemy z przezbrojeniami zależnymi od sekwencji.

Pojemność systemu

Mówimy, że system ma pojemność c jeśli co najwyżej c zadań może jednocześnie przebywać w systemie. Ograniczenie tego typu wynika z praktyki określającej poziom obciążenia systemu (niedociążony, obciążony normalnie, przeciążony), przy którym nie występują zatory w przepływie zadań (*congestion level*). Niech π będzie pewną kolejnością wprowadzania zadań do systemu. Zadanie $\pi(j)$ może zostać wprowadzone do sytemu jeśli zadanie $\pi(j-c)$ system opuściło, co implikuje naturalne ograniczenie

$$C_{m\pi(j-c)} \leqslant S_{1\pi(j)}, \quad j = c+1, \ldots, n, \tag{13.22}$$

modelowane jako specyficzne łuki ukośne $((m, j-c), (1, j))$, $j = c+1, \ldots, n$, dodane do grafu $G(\pi)$, patrz Rys. **??**.



Rysunek 13.8: Modelowanie ograniczonej pojemności systemu

Model ten można zastosować także (bez jakiejkolwiek zmiany) do systemów FMS, w których przetwarzane zadanie polega na obróbce detalu przymocowanego do *palety* (uchwytu transportowo-obróbczego), zaś liczba palet krążących w systemie jest ograniczona liczbą c.

Problemy z ograniczoną pojemnością systemu posiadają własności takie same lub analogiczne jak problemy z buforowaniem.

Ograniczenia geometryczne

Ograniczenie tego typu utrudnia lub blokuje możliwość przetwarzania zadań na stanowiskach sąsiednich lub przyległych. W celu przybliżenia problemu odwołamy się do przykładu praktycznego opisanego w Rozdz. ??. Niech i będzie wybranym stanowiskiem (maszyną), zaś j zadaniem na nim przetwarzanym w czasie p_{ij}. W praktycznym przykładzie zadanie polega na skompletowaniu do pojemnika zestawu części według receptury, operacja polega na pobraniu przez pracownika (lub automatyczny pobierak) wybranych części z i-tego segmentu regału, zaś czas wykonywania operacji zależy od liczby różnych części pobieranych aktualnie z tego segmentu. Do chwili zakończenia procesu pobierania z segmentu, b kolejnych segmentów występujących bezpośrednio za i jest niedostępna ze względu na geometrię układu

pracownik-kontener lub geometrię pobieraka, Rys. ??. Przyjmując, że kolejność realizacji zadań π jest permutacyjna (pracownicy lub kontenery nie mogą "wymijać" się na swojej trasie), otrzymujemy oczywiste ograniczenia

$$C_{i\pi(j)} \leqslant S_{i-b,\pi(j+1)}, \quad i = b+1,\ldots,m, \quad j = 1,\ldots,n-1, \tag{13.23}$$

reprezentowane przez ukośne, nieobciążone łuki grafu $((i,j),(i-b,j+1))$.

Tak skonstruowany warunek domyślnie zezwala na tworzenie nieograniczonej kolejki pomiędzy stanowiskami $i-b$ i $i-b-1$, co jednakże intuicyjnie nie jest dopuszczalne w omawianym przykładzie praktycznym. Istotnie, wstrzymane zadania, ze względu na brak miejsc składowania, tworzą kolejkę w obszarze stanowisk powodując ich sukcesywną blokadę. Warunki zapewniające w tym przypadku dopuszczalność są podobne do ograniczeń buforowania.

Zadanie $\pi(j)$ może zostać rozpoczęte na stanowisku i pod warunkiem, że zadanie je blokujące $\pi(j-1)$ i wykonywane wcześniej zostało zakończone i *opuściło* stanowisko $i+b$. Ponieważ opuszczenie stanowiska $i+b$ przez zadanie $\pi(j-1)$ następuje w chwili jego rozpoczęciem w stanowisku $i+b+1$, stąd otrzymujemy oczywiste warunki

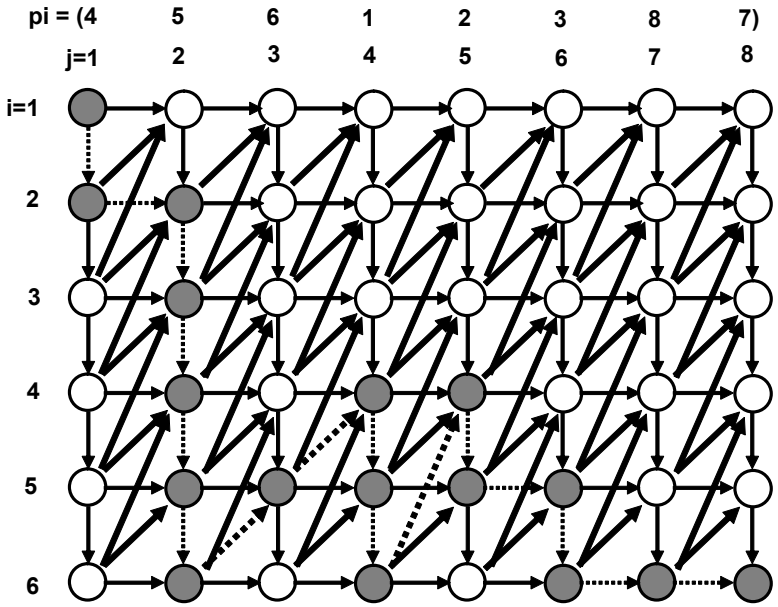
$$S_{i\pi(j)} \geqslant S_{i+b+1,\pi(j-1)}, \quad i = 1,\ldots,n-b-1, \quad j = 2,\ldots,n, \tag{13.24}$$

które następnie przekształcone do wygodniejszej postaci

$$C_{i\pi(j)} - p_{i\pi(j)} \leqslant S_{i-b-1,\pi(j+1)}, \quad i = b+2,\ldots,n, \quad j = 1,\ldots,n-1, \tag{13.25}$$

oraz mogą być reprezentowane jako ukośne łuki $((i,j),(i-b-1,j+1))$ z obciązeniem minus $p_{i\pi(j)}$ w grafie $G(\pi)$, Rys. ??. Zauważmy dalej, że niektóre łuki ukośne pochodzace z (13.23) w połączeniu z (13.25) są nadmiarowe i mogą być usunięte, w Rys. ?? zaznaczono je symbolem kasowania. Istotnie, rozważmy węzły (i,j) oraz $(i-b-1,j+1)$, pomiędzy którymi istnieją dwie drogi. Pierwsza o długości zerowej zawiera jeden łuk $((i,j),(i-b,j+1))$ pochodzacy z (13.23). Druga zawiera łuk $((i,j),(i+1,j))$, obciążony węzeł $(i+1,j)$ (obciążenie $p_{i+1,\pi(j)}$) oraz łuk $((i+1,j),(i-b,j+1))$ (obciążenie $-p_{i+1,\pi(j)}$). Długośc tej ostatniej drogi jest większa lub równa zero jeśli tylko łuk $((i,j),(i+1,j))$ posiada obciążenie nieujemne (czas transportu). Zatem, łuk $((i,j),(i-b,j+1))$ może zostać pominięty dla $i = b+1,\ldots,m-1$, $j = 1,\ldots,n-1$.

Dla kompletności modelu problemu praktycznego określimy sposób uwzględnienia dekopmozycji segmentów na regały (rozłączne ciągi segmentów). Niech i będzie ostatnim segmentem w pewnym regale, zaś $i+1$ pierwszym segmentem w regale następnym. Przypadek gdy pomiędzy i i $i+1$ można



Rysunek 13.9: Modelowanie ograniczeń geometrycznych i polityki NS

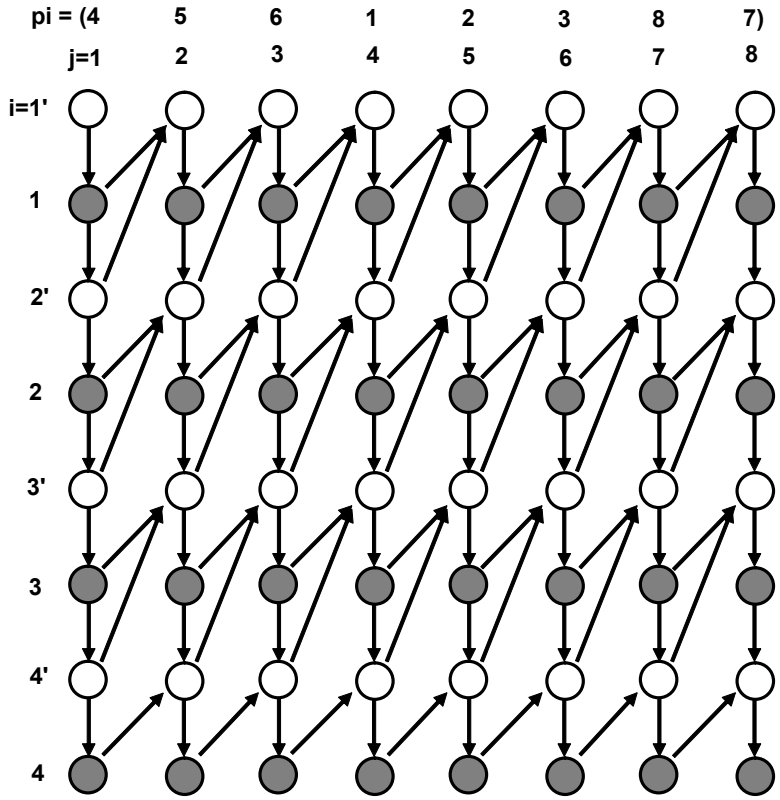
utworzyć nieograniczoną kolejkę jest oczywisty, nie potrzeba wprowadzać żadnych ukośnych łuków w $G(\pi)$ pomiędzy węzłami w kolumnach i i $i+1$, Rys. ??. Z kolei ograniczenia geometryczne blokujące $' < b$ stanowisk można modelować opisaną wcześniej techniką. Dopełnieniem jest czas transportu niezależny od zadań i stanowisk oraz pewne ustalone kryterium optymalności.

Zauważmy, że dla $b = m$ ograniczenia geometryczne odpowiadają trywialnemu systemowi z pojemnością $c = 1$, dla którego wartość kryterium $C_{\max}(\pi)$ nie zależy od permutacji (jest stała), kryterium $\sum C_j$ jest minimalizowane przez zmodyfikowaną regułę SPT (uszereguj według niemalejących wartości $\sum_{i=1}^{m} p_{ij}$), zaś kryterium $\sum w_j C_j$ jest minimalizowane przez zmodyfikowaną regułę WSPT (uszereguj według niemalejących wartości $\sum_{i=1}^{m} p_{ij}/w_j$).

Problemy z ograniczeniami geometrycznymi posiadają własności takie same lub analogiczne jak problemy z buforowaniem.

13.3 Kryterium czasu cyklu

Systemy produkcyjne realizujące wytwarzanie wieloasortymentowe, wielkoseryjne przy wolno-zmiennym w czasie asortymencie wyrobów mogą za-



Rysunek 13.10: Modelowanie mobilnego realizatora i polityki NS

spokajać zapotrzebowanie rynku bądź poprzez: (1) nieokresową zmianę jednolitego asortymentu produkcji, bądź też poprzez (2) cykliczne dostarczanie na wyjście mieszanki asortymentów. Skład ilościowy i jakościowy tej mieszanki zależy od średnio i długoterminowych zamówień klientów. Zakładając techniczną realizowalność obu powyższych rozwiązań, to drugie podejście jest bardziej atrakcyjne bowiem eliminuje lub ogranicza magazyn wyrobów gotowych. Dodatkowo minimalizując czas cyklu, dla ustalonego zestawu zadań w cyklu zwiększamy wydajność systemu oraz stopień wykorzystania maszyn. Dalszą poprawę efektywności funkcjonowania systemu uzyskuje się poprzez eliminację lub ograniczenie magazynowania półproduktów w magazynach pośrednich lub pomiędzy stanowiskami. Prowadzi to do warunków znanych w literaturze jako "no store" (zakaz składowania międzystanowiskowego) lub "limited store" (ograniczony obszar składowania międzystanowiskowego, tzw. bufor), dość często wymienianych w kontekście systemów JIT oraz częstokroć będących koniecznością wynikającą z pracy ciągłej sytemu. Systemy z ograniczonymi pojemnościami buforów lub ich brakiem są obiektem zainteresowań wielu badaczy ze względu na ich silne znaczenie praktyczne oraz problemy z uzyskaniem skutecznych algorytmów rozwiązywania. Zdecydowana większość otrzymanych dotychczas wyników dotyczy niecyklicznych systemów tego typu. Deterministyczne problemy cykliczne z ograniczeniami składowania należą do jednych z trudniejszych zagadnień optymalizacji dyskretnej. Świadczy o tym stosunkowo niewielka liczba pozycji literaturowych oraz kłopotliwe numerycznie procedury wyznaczania długości czasu cyklu i/lub harmonogramu.

Silna NP-trudność już wielu najprostszych badanych wersji problemu, ogranicza zakres stosowa-nia algorytmów dokładnych do instancji o małej liczbie zadań. Z tego powodu, do wyznaczania satysfakcjonujących rozwiązań, stosuje się powszechnie szybkie algorytmy przybliżone oparte na technikach przeszukiwań lokalnych. Prawie wszystkie metody tego typu opierają się na dwu-poziomowej dekompozycji problemu: wyznaczenie optymalnej kolejności zadań (poziom górny) oraz wielokrotne wyznaczenie minimalnej wartości kryterium dla danej kolejności zadań (poziom dolny). O ile dla "klasycznych" problemów szeregowania rozwiązanie problemu dolnego poziomu sprowadza się do analizy specyficznego grafu możliwej do wykonania w sposób efektywny czasowo, to w przypadku postawionego zagadnienia rozwiązanie (wielokrotne) zagadnienia dolnego poziomu jest stosunkowo czasochłonne bowiem, w ogólności, wymaga rozwiązania pewnego zagadnienia PL. Stąd wszelkie własności szczególne, w tym pozwalające na bardziej efektywne wyliczanie czasu cyklu i poszukiwanie harmonogramu oraz ograniczenie liczności lokalnie przeglądanego sąsiedztwa i/lub przyspieszenie szybkości

jego przeglądania są bardzo pożądane.

13.3.1 Sformułowanie problemu

Rozważmy system wytwórczy o strukturze szeregowej (przepływowy) złożony z m stanowisk (maszyn o jednostkowej przepustowości) indeksowanych kolejno $1, 2, \ldots, m$. W systemie tym należy wykonywać cyklicznie (w sposób powtarzalny) n, niekoniecznie różnych, zadań danych zbiorem $\mathcal{J}$. Pojedyncze zadanie odpowiada wytworzeniu jednej sztuki produktu finalnego (lub półproduktu). Wykonanie zadania odbywa się w m etapach, w sposób jednakowy dla wszystkich zadań, lecz w różnym czasie zależnym od zadania. Etap i wykonywany jest przez maszynę o numerze i, $i = 1, \ldots, m$. Ze względu na strategię wytwarzania, tj. brak możliwości składowania wyrobów podczas produkcji w magazynach i buforach międzystadialnych, zadanie zakończone na maszynie i, którego nie można przekazać na maszynę $i + 1$ ze względu na jej zajętość, musi pozostawać na i (blokować ją) aż do chwili zwolnienia $i+1$. Zadanie $j \in \mathcal{J}$ jest interpretowane jako sekwencja m operacji $O_{1j}, O_{2j}, \ldots, O_{mj}$ wykonywanych na maszynach $1, 2, \ldots, m$ w takiej kolejności. Operacja O_{ij} odpowiada wykonywaniu zadania j na i-tej maszynie w czasie $p_{ij} > 0$. W danej chwili maszyna może wykonywać tylko jedno zadanie, oraz zadanie może być wykonywane tylko na jednej maszynie. Zestaw zadań wykonywanych w cyklu może być dany arbitralnie lub wynikać z polityki jednostajnego i równoczesnego dostarczania wszystkich zamówionych partii produkcyjnych (tzw. mieszanki produktów wyjściowych). W tym drugim przypadku przyjmuje się, że złożone zostało zamówienie na produkty pochodzące ze zbioru $\mathcal{J}^* = \{1, 2, \ldots, t\}$ w ilościach $r_1, \ldots, r_t$, odpowiednio. Zatem zachodzi potrzeba zrealizowania w systemie zadań niekoniecznie różnych. Niech $c > 1$ będzie wspólnym dzielnikiem liczb $r_1, \ldots, r_t$. Zbiór J* dzielimy na c identycznych podzbiorów nazywanych MPS (ang. minimal part set), z których każdy zawiera n=r1/c+...+rt/c zadań. MPS-y są przetwarzane jeden po drugim w sposób cykliczny, dostarczając mieszankę produktów w ilościach odpowiednio r1/c,...,rt/c na cykl. Dalej będziemy się kolejny zajmować wyłącznie pojedynczym MPS-em. Przyjmujemy, że zadania ze zbioru J wykonywane są w określonej kolejności, takiej samej dla wszystkich MPS-ów, co zasadniczo implikuje cykliczność sekwencji, ale niekoniecznie harmonogra-mów czasowych. W pracy [1] pokazano, że w systemie przepływowym z ograniczeniami "bez magazynowania" dopuszczalna kolejność wykonywania zadań na maszynach musi być identyczna na każdej maszynie. Zatem, kolejność wykonywania zadań w każdym MPS-ie możemy opisać przy pomocy jednej permutacji =((1), (2),..., (n)) elementów

zbioru 1,. . . ,n. Harmonogram czasowy wykonywania zadań k-tego MPS-a (dla danej kolejności ich realizacji) opisujemy przy pomocy macierzy [Sk]m n, gdzie oznacza termin rozpoczęcia wykonywania zadania j na maszynie i, patrz także [5]. Harmonogram ten musi spełniać następujące ograniczenia (1) (2) (3) (4) (5) dla k=1,2,. . . Takie postawienie problemu, choć dopuszcza pewną swobodę w konstrukcji harmono-gramu, równocześnie komplikuje jego wyznaczenie, bowiem warunki (4) oraz (5) wymuszają zależną analizę MPS-ów. Załóżmy zatem dalej, że nie tylko kolejność zadań jest cyklicznie powtarzana ale także, że harmonogram czasowy pracy systemu jest w pełni cykliczny. Oznacza to istnieje stała T (okres) taka, że (6) Okres T zależy od i jest nazywany czasem cyklu systemu. Minimalną wartość T, dla ustalonej , będziemy nazywać minimalnym czasem cyklu i oznaczać przez T( ). Zauważmy, że podstawiając (6) do (1) - (5) otrzymamy warunki odnoszące się tylko do harmonogramu k-tego MPS-a oraz, że wszystkie MPS-y dla k=1,2,. . . mają taką samą postać. Zatem wystarczy skonstruować harmonogram dla jednego MPS-a i dokonać jego translacji o wielkość kT, k=1,2,. . . na osi czasu. Ponieważ ogranicze-nia (1) - (5) są identyczne dla wszystkich k, to możemy pominąć górny indeks k w oznaczeniach. Wartość T( ) oraz harmonogram Sij, i=1,...,m, j=1,...,n każdego cyklu można wyznaczyć poprzez rozwiązanie zadania postaci

(8) (9) (10) (11) (12) (13) Bez straty ogólności możemy przyjąć, że punktem zakotwiczenia harmonogramu jest Wprowadzone oznaczenia pozwalają na eleganckie sformułowanie obu poziomów dekompono-wanego problemu optymalizacji. Na poziomie górnym należy wyznaczyć permutację * , gdzie jest zbiorem wszystkich permutacji, taką że . (14) Na poziomie dolnym należy, dla danej , wyznaczyć T( ). Poszukiwanie * można zrealizować w technice SA, TS czy GA, poprzez analogię do innych problemów posiadających zbiór rozwiązań opartych na permutacjach zbioru n-elementowego.

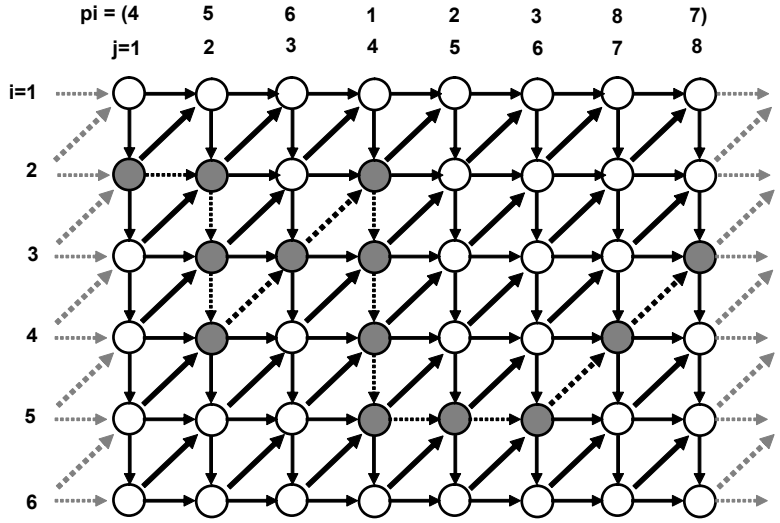
13.3.2 Problem dolnego poziomu

13.3.3 Problem górnego poziomu

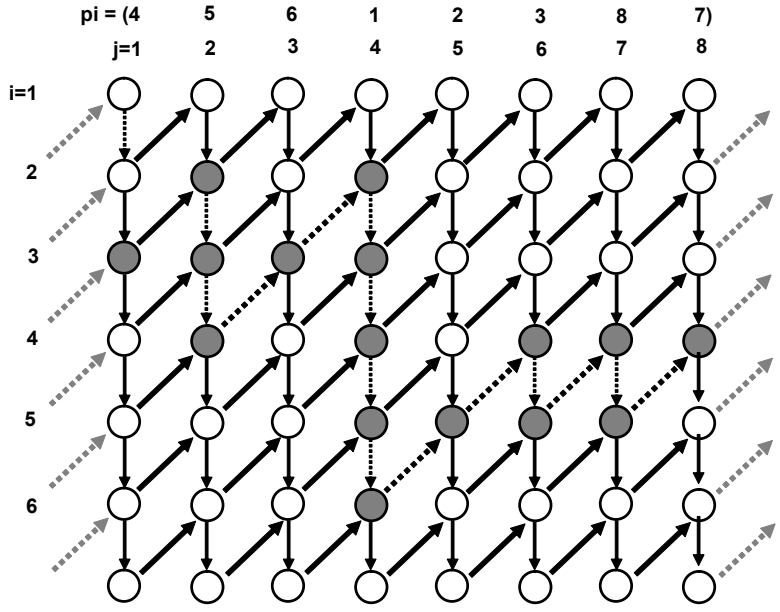
13.3.4 Alternatywne techniki rozwiązywania

13.3.5 Uwagi i wnioski

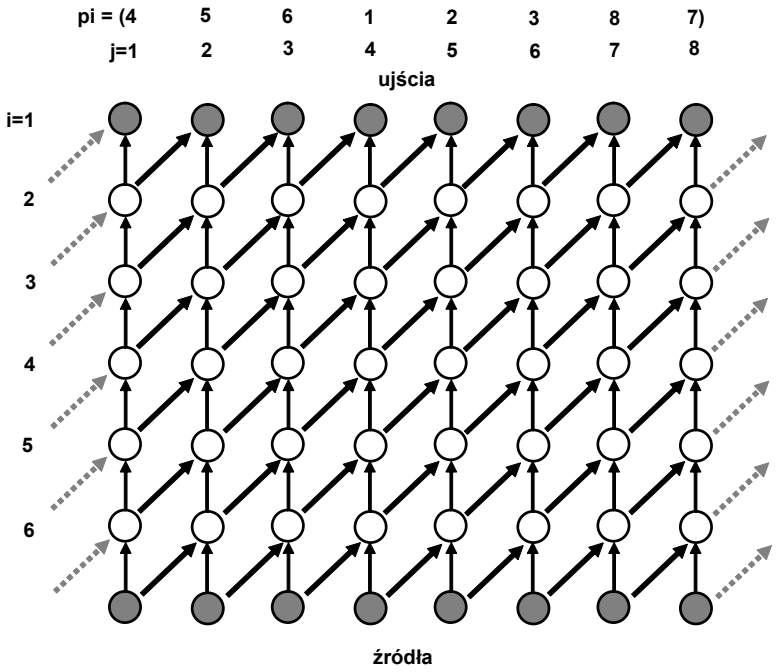
Konsekwencje wprowadzonych i wykazanych własności są wielorakie. Własność 1 pozwala znaleźć wartość minimalnego czasu cyklu $T(\pi)$ bez konieczności wyznaczania dokładnego harmonogramu S_{ij}, $i = 1, ..., m$, $j = 1, ..., n$, co może być skutecznie wykorzystywane w metodach SA, TS, GA.



Rysunek 13.11: Modelowanie czasu cyklu



Rysunek 13.12: Przekształcony graf dla permutacji



Rysunek 13.13: Sieć przepływowa

Faktycznie, wyznaczenie szczegółowego harmonogramu jest potrzebne tylko dla kolejności generowanej na wyjściu algorytmu lokalnej optymalizacji. Własność 3 mówi, że $T(\pi)$ można wyznaczyć efektywniej niż kiedykolwiek dotychczas. Po drugie, własność 2, poprzez swój związek z długości drogi w grafie pozwala na wprowadzenie własności blokowych znanych dla problemów z buforowaniem, wyjątkowo korzystnych dla konstrukcji efektywnego otoczenia w metodzie TS oraz przy budowie tzw. akceleratora przyśpieszającego proces obliczeniowy. Własność 4 pozwala znaleźć szczegółowy harmonogram cyklu efektywniej niż kiedykolwiek dotychczas. W końcu, uporządkowanie i usystematyzowania podejścia pozwala na skuteczne zaatakowanie trudniejszych problemów tego typu, takich jak np. problem o strukturze zadaniowej (job-shop).

Z analizy wyników prezentowanych w tabeli 1 wynika, że dla rozważanego problemu algorytm SA dostarcza najlepszych wyników spośród testowanych GA, SA, TS. Algorytm TS, głównie ze względu na znaczny czas działania, nie jest w stanie osiągnąć właściwych warunków pracy, bowiem w porównywalnym do SA i GA czasie jest w stanie wykonać zaledwie 50 (lub 10) iteracji, a powinien co najmniej 1000...5000. Zastosowanie w tym przypadku własności blokowych i akceleratora w TS powinno dostarczyć znaczących korzyści. Oddzielnym problemem pozostaje adekwatność testów Taillard'a dla zastosowań w systemach bez składowania. Jak już pokazano w testach numerycznych z prac [10,4], jakość otrzymywanych wyników silnie zależy od cech statystycznych realnych przykładów występujących w praktyce.

13.4 Uwagi

Internetowa biblioteka przykładów testowych [31], dedykowana dla problemu przepływowego $F^*||C_{\max}$ jest także używana jako zbiór przykładów referencyjnych dla problemów klasy $F||\sum f_i$. Niestety, brak odpowiednio mocnych własności szczególnych analogicznych do własności blokowych dla problemu $F||C_{\max}$ pokazuje, że rozwiązanie problemu szeregowania z kryterium addytywnym jest zdecydowanie trudniejsze od jego odpowiednika z kryterium minimaksowym. W szczególności, zastosowanie metod LS, wobec braku odpowiednich akceleratorów, implikuje znaczny koszt obliczeń i długi czas działania algorytmów. Stąd wniosek, że metody SA, SJ, GA będą wykazywały korzystniejsze własności numeryczne od innych podejść, przy umiarkowanym koszcie obliczeń. Tak jaest istotnie. Dodatkowo, brak odpowiednio skutecznych kryteriów eliminacyjnych oraz powszechnie znana zgrubność oszacowania dolnych ograniczeń, pociąga wyraźnie mniej sukce-

sów osiągalnych w schemacie B&B. Stąd też pesymizm w ocenie przydatności schematu B&B dla rozwiązywania problemów szczególnie tej klasy.

Techniki rozwiązywania problemów postaci $F^*||f_{\max}$ oraz $F||f_{\max}$ są analogiczne do tych dedykowanych dla $F||C_{\max}$. Większość korzystnych własności blokowych można uogólnić na wymienione klasy zagadnień. Niestety, mimo iż odpowiednie uogólnienia można zrobić także dla dolnych ograniczeń, wykazuja one słabość ze względu na potrzebę stosowania głębokich relaksacji. Stąd także obserwowana słabość odpowiednich algorytmów B&B, równiez i w tym przypadku.

W ostatnch latach szczególnym zainteresowaniem cieszą się problemy przepływowe z dodatkowymi ograniczeniami buforowania, ograniczeniami oczekiwania [160], ograniczoną pojemnością systemu, ograniczeniami geometrycznymi, ruchomego realizatora oraz innymi. Nie tylko modeluja one bardziej precyzyjnie istniejącą rzeczywistość, ale także umożliwiają zaprojektowanie bardziej "agresywnych" (lepszych ekonomicznie) strategii sterowania systemami tego typu. Faktycznie dość duża liczba współczesnych systemów wytwarzania ma przepływowo-równoległą strukturę potokową, dla której algorytmy sterowania wykorzystuja doświadczenia zdobyte w projektowaniu algorytmów dla podstawowych systemów przepływowych.

Nielicznie w literaturze rozważane są problemy przepływowe z kryteriami nieregularnymi. Wyłączywszy przypadki szczególne, algorytmy są rozszerzeniami podejść z Rozdz. **??**.

13.5 Warsztat

1. Zaadaptuj algorytm NEH dla problemu $F^*|r_j, q_j|C_{\max}$.
2. Zaprojektuj algorytm SAa dla problemu $F^*|setup|\overline{\mathcal{C}}$.
3. Zaprojektuj algorytm GA dla problemu

14

Problemy gniazdowe

Problem gniazdowy jest powszechnie uważany za szczególnie trudne zagadnienie optymalizacji kombinatorycznej. Ze względu na bardzo duże znaczenie praktyczne problem z kryterium $C_{\max}$ był atakowany przez wielu badaczy i doczekał się szeregu algorytmów dokładnych i przybliżonych, patrz przegląd w pracy [36]. Najlepsze znane algorytmy dokłądne dla problemu gniazdowego oparte są na schemacie B&B. Mimo iż w ostatnich latach dokonano znacznego postępu w rozwoju tego podejścia, otrzymane tą drogą algorytmy są wciąż nieatrakcyjne dla praktyków. Są czasochłonne zaś rozmiar przykładów, które są w stanie rozwiązać w rozsądnym limicie czas jest poniżej 225 operacji. W praktyce, taki algorytm jest dość złożóną konstrukcją zawierającą kilka współdziałających ze sobą algorytmów składowych. Z tego względu wykonanie implementacji wymaga znajomości teorii szeregowania, struktur danych i algorytmów, jak i pewnego doświadczenia programistycznego. W przeciwieństwie do słabej atrakcyjności algorytmów dokładnych, stosunkowo dużo badań poświęcono algorytmom przybliżonym dla tego problemu. Stosują one szereg różnych podejść opisanych bardziej szczegółowo w kolejnych podrozdziałach. Aktualnie najlepsze algorytmy pozwalają rozwiązać problemy do 2,000 operacji z kryterium $C_{\max}$ z błędem względnym, przeciętnie poniżej 4-5%, co należy uznać za rezultat bardzo dobry. Co więcej implementacja najlepszych algorytmó tego typu jest nieskomplikowana. Dla porownania, rozwiązania losowe mają błąd względny rzędu 130%, zaś konwencjonalne algorytmy priorytetowe około 30%.

14.1 Problem i jego modele

Dane są zbiór zadań $J = \{1, \ldots, n\}$, zbiór maszyn $M = \{1, \ldots, m\}$ i zbiór operacji $O = \{1, \ldots, o\}$. Zbiór O jest dekomponowany na podzbiory odpowiadające zadaniom. Zadanie j składa się z sekwencji o_j operacji indeksowanych kolejno przez $(l_{j-1} + 1, \ldots, l_j)$, które powinny być wykonane w podanej kolejności, gdzie $l_j = \sum_{i=1}^{j} o_i$, jest całkowitą liczbą operacji pierwszych j zadań, $j = 1, \ldots, n$ $(l_0 = 0)$, oraz $\sum_{i=1}^{n} o_i = o$. Operacja i musi być wykonana na maszynie $\nu_i \in M$ w nieprzerywalnym czasie $p_i > 0$, $i \in O$. Zwykle zakłada się, że każde dwie kolejne operacje są wykonywane na różnych maszynach [1]. Każda maszyna może wykonywać, co najwyżej jedną operację, w dowolnej chwili czasu. Rozwiązaniem dopuszczalnym jest wektor terminów rozpoczęcia operacji $S = (S_1, \ldots, S_o)$, taki, że powyższe ograniczenia są spełnione, tzn.

$$S_{l_{j-1}+1} \geqslant 0, \quad j = 1, \ldots, n \tag{14.1}$$

$$S_i + p_i \leqslant S_{i+1}, \quad i = l_{j-1} + 1, \ldots, l_j - 1, \quad j = 1, \ldots, n, \tag{14.2}$$

$$(S_i + p_i \leqslant S_j) \,\dot{\vee}\, (S_j + p_j \leqslant S_i), \; i, j \in \mathcal{O}, \; \nu_i = \nu_j, \; i \neq j. \tag{14.3}$$

Warunek dysjunktywny (14.3) wymaga aby każde dwie operacje wykonywane na tej samej maszynie posiadały jednoznacznie określoną kolejność ich wykonywania. Do zbioru ograniczeń należy dołączyć funkcję celu odpowiednią do rozwiązywanego problemu. Jednym z najczęściej używanych, ze względu na zastosowania praktyczne oraz możliwości algorytmiczne, jest kryterium terminu zakończenia wykonywania wszystkich zadań $\mathcal{C}_{\max}$

$$\mathcal{C}_{\max}(S) = \max_{1 \leqslant j \leqslant n} (S_{l_j} + p_{l_j}). \tag{14.4}$$

Innym, stosunkowo często wymienianym kryterium jest suma terminów zakończenia zadań

$$\mathcal{C}(S) = \sum_{j=1}^{n} (S_{l_j} + p_{l_j}), \tag{14.5}$$

mająca bezpośredni związek z minimalizacją czasu przepływu zadań oraz ich przestojami w systemie. Należy przy tym zaznaczyć, że mimo iż oba problemy są silnie NP-trudne, są podobnie modelowane, to drugi z wymienionych

[1] Założenie to jest wprowadzane dla prostoty opisu modelu podstawowego, nie jest restryktywne i może być usunięte przez wprowadzenie fikcyjnej operacji separującej o nieskończenie małym czasie trwania, wykonywanej na fikcyjnej maszynie.

jest powszechnie uznawany za trudniejszy, głównie z powodu braku szczególnych własności korzystnych dla konstrukcji algorytmu rozwiązywania.

Aktualnie, problem gniazdowy posiada kilka odmiennych technik modelowania, w zależności od proponowanych algorytmów rozwiązywania. Ważniejsze podamy poniżej.

Model PLC

Warunek dysjunktywny (14.3) może być zamieniony podobnie jak w Rozdz. ?? na dwa warunki ze zmiennymi binarnymi, prowadząc do zadania programowania liniowego, binarno-ciągłego. Otrzymane zadanie może być następnie rozwiązywane metodami ogólnymi podanymi w Rozdz. ??. Niestety, to oczywiste postępowanie uważane jest za jedno z najmniej efektywnych.

Model dysjunktywny

Najstarszy historycznie model odwołuje się do pojęcia grafu dysjunktywnego. Graf ten $G = (\mathcal{O}, \mathcal{U}, \mathcal{V})$ ma zbiór węzłów $\mathcal{O}$ reprezentujących operacje, zbiór łuków koniunktywnych (skierowanych) przedstawiających kolejność technologiczną wykonywania operacji

$$\mathcal{U} = \bigcup_{j=1}^{n} \bigcup_{i=l_{j-1}+1}^{l_j-1} \{(i, i+1)\} \tag{14.6}$$

oraz zbiór łuków dysjunktywnych (nieskierowanych) przedstawiających wszystkie możliwe kolejności realizacji operacji na maszynach

$$\mathcal{V} = \bigcup_{i,j \in \mathcal{O};\ i \neq j;\ \nu_i = \nu_j} \mathcal{V}_{ij} = \bigcup_{i,j \in \mathcal{O};\ i \neq j;\ \nu_i = \nu_j} \{(i,j),(j,i)\}. \tag{14.7}$$

Zauważmy, że łuk dysjunktywny $\{(i,j),(j,i)\}$ może być skojarzony jednoznacznie z odpowiednim warunkiem dysjunktywnym (14.3). Węzeł i posiada obciażenie p_i, zaś wszystkie łuki mają obciążenie zerowe. Wybór dokładnie jednego łuku ze zbioru $\{(i,j),(j,i)\}$ nazywamy *skierowaniem* łuku dysjunktywnego. Wybór ten odpowiada ustaleniu kolejności wykonywania operacji $i \to j$ albo $j \to i$. Podzbiór $\mathcal{W} \subset \mathcal{V}$ zawierający dokładnie jeden łuk z każdego zbioru $\mathcal{V}_{ij}$ nazywamy *reprezentacją* łuków dysjunktywnych. Reprezentacja może być *kompletna* (wszystkie łuki dysjunktywne posiadają określoną orientację) lub *częściowa* (tylko część z nich ma tą własność). Każda reprezentacja kompletna generuje jedno rozwiązanie, niekoniecznie dopuszczalne.

Rozwiązanie dopuszczalne jest generowane przez reprezentację kompletną $\mathcal{W}$ taką, że graf $G(\mathcal{W}) = (\mathcal{O}, \mathcal{U} \cup \mathcal{W})$ jest acykliczny. Każda reprezentacja kompletna dopuszczalna generuje unikalne uszeregowanie dosunięte w lewo. Dla uszeregowania dopuszczalnego, terminy rozpoczęcia wykonywania operacji S_i można wyznaczyć jako najdłuższe drogi dochodzące do wierzchołków grafu, oraz na ich bazie – wartość funkcji celu. Do wyznaczania długości dróg w grafie można zastosować standardowy algorytm Bellmana, patrz np. praca [69], prowadzący do wzoru rekurencyjnego

$$S_j = \max_{i \in \mathcal{B}_j}(S_i + p_i),\ j \in \mathcal{O}, \tag{14.8}$$

gdzie

$$\mathcal{B}_j = \{i :\ (i,j) \in \mathcal{U} \cup \mathcal{W}\} \tag{14.9}$$

z warunkiem początkowym $S_j = 0,\ \mathcal{B}_j = \emptyset$. W przypadku gdy numeracja wierzchołków grafu jest zgodna z ich *porządkiem topologicznym*, rekurencja we wzorze (14.8) może przebiegać dla $j = 1, 2, \ldots, o$. Jeśli numeracja wierzchołków nie posiada podanej własności, można zaadoptować algorytm szukania porządku topologicznego do równoczesnego liczenia długości dróg w grafie. Niezależnie od przyjetej techniki postępowania, algorytm wyliczania długości wymaganych dróg w grafie ma złożóność obliczeniową $O(|\mathcal{U} \cup \mathcal{W}|$. Ponieważ $G(\mathcal{W})$ posiada o wierzchołków i $O(o^2)$ krawędzi, zatem wyznaczenie wartości funkcji celu dla danej reprezentacji $\mathcal{W}$ wymaga czasu rzędu $O(o^2)$. W szczególności, wartość kryterium $C_{\max}$ dla rozwiązania reprezentowanego przez $\mathcal{W}$ jest równa długości najdłuższej drogi (drogi krytycznej) w $G(\mathcal{W})$. Technika grafu dysjunktywnego umożliwia jedynie modelowanie zagadnienia, nie generuje natomiast żadnej konkretnej metody rozwiązywania.

Wyznaczenie najlepszego dosuniętego w lewo uszeregowania jest równoważne wyznaczeniu dopuszczalnej orientacji kompletnej, która minimalizuje przyjęte kryterium optymalności. Wprawdzie poszukiwania mogłyby zostać ograniczone do uszeregowań aktywnych, jednakże dla danej kommpletnej reprezentacji, aktywnośc na pierwszy rzut oka nie jest oczywista i wymaga pewnych działań w celu jej zweryfikowania. Aktywność zależy bowiem od wielkości czasó trwania operacji, natomiast dosunięcie w lewo – nie. Dlatego też łatwiej jest rozważać nieco większy zbiór rozwiązań dosuniętych w lewo zamiast aktywnych.

W Rys. ??–?? przedstawiono graf $G = (\mathcal{O}, \mathcal{U}, \mathcal{V})$ dla przykładu z Tab. 14.1, stosując odmienne techniki rysowania. W Rys. ?? zadania umieszczono w wierszach, zachowując naturalną kolejność operacji w zadaniu (łuki

j	i	p_i	ν_i	
	0			$\leftarrow l_0$
1	1	10	4	
1	2	20	2	
1	3	15	3	
1	4	25	1	$\leftarrow l_1$
2	5	30	2	
2	6	10	3	$\leftarrow l_2$
3	7	20	2	
3	8	40	1	
3	9	10	3	$\leftarrow l_3$
4	10	15	3	$\leftarrow l_4$

Tabela 14.1: Dane dla przykładu problemu gniazdowego

ciągłe), a następnie dorysowano łuki dysjunktywne (przerywane) dla każdej pary operacji wykonywanej na tej samej maszynie. Podobnie postąpiono w Rys. ?? z tym, że położenia operacji dostosowano do liniowego porządku maszyn, wyszczególnionego w górnej części rysunku. W konsekwencji uzyskano czytelniejsze zgrupowanie łuków dysjunktywnych jednak kosztem przejrzystości powiązań w całym grafie. Zbiór $\mathcal{U}$ zawiera $o - n$ łuków, zaś zbiór $\mathcal{V}$ zawiera $\frac{1}{2}\sum_{k=1}^{m} m_k(m_k - 1)$ łuków nieskierowanych, gdzie m_k jest liczbą operacji z $\mathcal{O}$ wykonywanych na maszynie k. Ponieważ reprezentacja $\mathcal{W}$ odpowiada wyborowi orientacji dla każdego łuku dysjunktywnego z $\mathcal{V}$, liczba wszystkich rozwiązań (łącznie z niedopuszczalnymi) jest rzędu $O(2^{o^2})$, dokładniej jest równa $2^{\sum_{k=1}^{m} m_k(m_k-1)/2}$. Model dysjunktywny jest wciąż popularny, mimo niedoskonałości, prawdopodobnie głównie ze względu na łatwość programowania.

Model kombinatoryczny

Dla wielu zastosowań lepszą techniką modelowania jest kombinatoryczna reprezentacja rozwiązań, pozbawiona redundantności charakterystycznej dla grafu dysjunktywnego. Zbiór operacji $\mathcal{O}$ może być w naturalny sposób dekomponowany na podzbiory operacji wykonywanych na jednej ustalonej maszynie $k \in \mathcal{M}$, $\mathcal{M}_k = \{i \in \mathcal{O} : \nu_i = k\}$ i niech $m_k = |\mathcal{M}_k|$.

Kolejność wykonywania operacji na maszyne k jest określony permutacją $\pi_k = (\pi_k(1), \ldots, \pi_k(m_k))$ n zbiorze $\mathcal{M}_k$, $k \in \mathcal{M}$; $\pi_k(i)$ oznacza ten element z $\mathcal{M}_k$, który jest w pozycji i w π_k. Niech $\Pi(\mathcal{M}_k)$ będzie zbiorem wszystkich permutacji na $\mathcal{M}_k$. Kolejność wykonywana na wszystkich maszynach jest definiowana jako m-tka $\pi = (\pi_1, \ldots, \pi_m)$, gdzie $\pi \in \Pi = \Pi(\mathcal{M}_1) \times \Pi(\mathcal{M}_2) \times \ldots \times \Pi(\mathcal{M}_m)$. Dla kolejności π, tworzymy graf skierowany (digraf) $G(\pi) = (\mathcal{O}, \mathcal{U} \cup E(\pi))$ ze zbiorem węzłów $\mathcal{O}$ i zbiorem łuków $\mathcal{U} \cup \mathcal{E}(\pi)$, gdzie $\mathcal{U}$ jest zbiorem łuków stałych reprezentujących kolejność wykonywania operacji w zadaniu, zaś zbiorem łuków reprezentujących kolejność wykonywania operacji na maszynach

$$E(\pi) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{m_k - 1} \{(\pi_k(i), \pi_k(i+1))\}. \tag{14.10}$$

Każdy węzeł $i \in \mathcal{O}$ ma wagę p_i i każdy łuk ma wagę zero. Zauważmy, że każdy węzeł ma co najwyżej dwa bezpośrednie następniki i co najwyżej dwa bezpośrednie następniki. Oznaczmy przez $\underline{s}_i$ oraz $\overline{s}_i$ bezpośredni oporzednik i następnik sekwencyjny operacji i (wynikający z $E(\pi)$), wstawiając symbol $\circ$ jeśli odpowiedni byt nie istnieje. Podobnie, niech $\underline{t}_i$ oraz $\overline{t}_i$ oznacza bezpośredni poprzednik i następnik technologiczny operacji i (wynikający z $\mathcal{U}$). Kolejność wykonywania π jest dopuszczalna jeśli tylko graf $G(\pi)$ nie zawiera cyklu. W Rys. **??** pokazano graf $G(\pi)$ dla danych z przykładu w Tabl. 14.1 oraz kolejności $\pi = (\pi_1, \pi_2, \pi_3, \pi_4)$, $\pi_1 = (8, 4)$, $\pi_2 = (5, 7, 2)$, $\pi_3 = (3, 6, 10, 9)$, $\pi_4 = (1)$. Każda dopuszczalna kolejność wykonywania generuje rozwiązanie dopuszczalne S, w którym termin rozpoczęcia S_i równa się długości najdłuższej drogi dochodzącej do wierzchołka i (bez p_i) w $G(\pi)$, $i \in \mathcal{O}$; jeśli wierzchołek nie ma poprzednika długość jest zero. Dla danej dopuszczalnej π wyznaczenie wartości funkcji celu wymaga czasu rzędu $O(o)$, zatem mniejszego niż w przypadku reprezentacji dysjunktywnej. Co więcej, Rys. ?? pokazuje, że także graf $G(\pi)$ posiada bardziej przejrzystą strukturę od odpowiedniego grafu z reprezentacją dysjunktywną (łuki sekwencyjne, należące do zbioru $E(\pi)$, zaznaczono linią przerywaną). Istotnie reprezentację $\mathcal{W}$ w grafie z Rys. ?? można otrzymać dodając w grafie z Rys. ?? do $E(\pi)$ wszystkie łuki kolejnościowe (przerywane) wynikające z przechodniości relacji poprzedzania operacji na maszynie. Ponieważ dla każdej π możemy w ten sposób utworzyć odpowiednią reprezentację łuków dysjunktywnych $\mathcal{W}$, zaś nie dla wszystkich reprezentacji możemy skonstruować π, stąd wniosek, że liczba wszystkich rozwiązań w grafie dysjunktywnym jest większa niż dla przedstawienia permutacyjnego. Faktycznie, rozbieżnośc dotyczy tylko rozwiązań niedopuszczalnych, bowiem rozwiązania dopuszczalne są określone

jednoznacznie w obu modelach. Podobnie jak poprzednio, wartość $C_{\max}(\pi)$ dla kolejności π równa się długości najdłuższej drogi w $G(\pi)$. Liczba wszystkich rozwiązań (łącznie z niedopuszczalnymi) jest równa $\prod_{k=1}^{m}(m_k!)$.

Model listowy

Oba wymienione powyżej modele rozwiazań (reprezentacja dysjunktywna, ciąg permutacji) wymagaja kontroli dopuszczalności poprzez detekcję cyklu w odpowiednim grafie. (W przypadku bardziej złożónych problemów badanych jest fakt występowania cyklu o dodatniej długości.) Ponieważ koszt badania acykliczności grafu jest porównywalny z kosztem algorytmu wyznaczania wartości S przyuzyciu grafu, efektywna eliminacja rozwiązań niedopuszczalnych jest kluczowa dla szybkości pracy wielu algorytmów, głównie tych modyfikujących rozwiązanie w sposób krokowy. Możliwe są przy tym rózne alternatywne podejścia: (1) bezpośrednie kosztowne badanie acykliczności grafu, (2) wyznaczanie długości dróg w grafie z równoczesną detekcją cyklu, (3) predykcja wystąpienia cykle, (4) generowanie rozwiązań gwarantujących dopuszczalność, (5) używanie reprezentacji rozwiązań nie wymagającej badania dopuszczalności. Omówimy te podejścia kolejno.

Do sprawdzenia czy graf posiada cykl można wykorzystać algorytm wyznaczania porządku topologicznego wierzchołków [69]. Biorąc pod uwagę, że algorytm ten można zmodyfikować w celu wyznaczania długości dróg w grafie, podejście (2) należy uznać za korzystniejsze niż (1); tak też jest w praktyce. Metoda predykcji wystąpienia cyklu (3) jst stosowana w metodach poszukiwań lokalnych, gdzie bada się czy proponowana nieznaczna modyfikacja rozwiązania bieżącego naruszy jego dopuszczalność. Wymaga tozasadniczo odpowiedzi na pytanie czy w grafie reprezentującym bieżące rozwiązanie istnieje droga $i \to j$, $j \in K \subseteq \mathcal{O}$ (lub $j \to i$, $j \in K \subseteq \mathcal{O}$), gdzie $i \in \mathcal{O}$ oraz K są pewnym ustalonym wierzchołkiem i podzbiorem wierzchołków, odpowiednio. Detekcję wspomnianej drogi można przeprowadzić poprzez przeszukanie grafu wszerz, startujac z K oraz posuwając się po bezpośrednich poprzednikach (lub następnikach, odpowiednio) aż do zakończenia przeszukania lub osiągnięcia węzła i. Mimo iż podany algorytm jest w praktyce stosunkowo szybki, jego złożoność obliczeniowa jest porównywalna z badaniem cycklu explicite. Z tego też powodu dużo wysiłku włożono w rozwinięcia podejścia (4). Jak do tej pory znane są pewne warunki dostateczne, przy spełnieniu których modyfikacja rozwiązania dopuszczalnego gwarantuje otrzymanie nowego rozwiązania dopuszczalnego, patrz Rozdz. 14.2. Niestety, warunki te odnoszą się tylko do pewnej podklasy problemów. głónie z kryterium $C_{\max}$ oraz jego pochodnymi $L_{\max}$, $f_{\max}$.

Ostatnie podejście wprowadza inny sposób reprezentacji rozwiązania problemu gniazdowego za pomocą jednej wspólnej listy operacji. Oznaczmy przez ω permutację o-elementową na zbiorze $\mathcal{O}$. Permutacja ω reprezentuje pewne rozwiązanie dopuszczalne problemu gniazdowego jeśli: (a) operacje wchodzące w skład jednego zadania występują w poprawnej kolejności w ω, (b) kolejność wykonywania operacji na maszynie jest zgodna z kolejnością pojawiania się ich w ω. Istnieje związek pomiędzy ω (w reprezentacji listowej) oraz π (w reprezentacji kombinatorycznej). Niech π będzie rozwiązaniem dopuszczalnym: ω może być otrzymane jako porządek topologiczny w $G(\pi)$. Ponieważ dla każdego π może istnieć wiele porządków topologicznych, transformacja z π do ω jest niejednoznaczna, natomiast z ω do π jest jednoznaczna. Zauważmy, że dowolna permutacja ω spełniająca warunek (a) generuje zawsze rozwiązanie dopuszczalne, zatem nie ma potrzeby sprawdzania dopuszczalności. Permutację ω można wykorzystać, w co najmniej trojaki sposób: (1) przekształcając ją do π w celu skorzystania z własności grafu $G(\pi)$, (2) formułując algorytm konstrukcji wektora $S = (S_1, \ldots, S_o)$ bezpośrednio na bazie ω, (3) wykorzystując ją jako priorytetową listę rozstrzygania konflitów w schemacie GT opisanym dalej.00

Niekiedy pewien kłopot może sprawiać generowanie permutacji ω spełniającej warunek (a). Problemu tego można uniknąć operując ciągiem $\omega^* = (\omega^*(1), \ldots, \omega^*(o))$, $\omega^*(i) \in \mathcal{J}$, kodującym położenie zadań zamiast operacji. Permutacja ωjest przy tym jednoznacznie wyznaczana z ω^* poprzez przyporządkowanie kolejnym występieniom tego samego zadania w ω^* kolejnych numerów operacji tego zadania, zgodnie z warunkiem (a). Przykładowo, zadania występujące w Rys. **??** w kolejności $\omega^* = (1, 2, 3, 1, 3, 1, 2, 4, 3, 1)$ są tłumaczone na kolejność operacji $\omega = (1, 5, 7, 2, 8, 3, 6, 10, 9, 4)$ w sposób jednoznaczny.

Wybór modelu

Wybór modelu jest kompromisem pomiędzy dopussczalnością, redundantnością, dostępnymi własnościami oraz rodzajem projektowanego algorytmu. Przykładowo, mimo iż każda kolejność listowa spełniająca warunek (a) generuje zawsze rozwiązanie dopuszczalne, liczba tak otrzymanych różnych kolejności jest dużo większa niż wszystkich dopuszczalnych π. Dla przykłądu testowego FT10 zawierającego 10 zadań wykonywanych na 10 maszynach (100 operacji) liczba wszystkich rozwiązań dla odpowiednich reprezentacji wynosi: model dysjunktywny - $2^{10 \cdot 10 \cdot 9/2} \approx 2.9 \cdot 10^{135}$ (łącznie z niedopuszczalnymi), model kombinatoryczny – $(10!)^{10} \approx 4 \cdot 10^{65}$ (łącznie z niedopuszczalnymi), model listowy – $(100!)/(10!)^{10} \approx 2.3 \cdot 10^{92}$ (wszystkie

są dopuszczalne). Jak widać model listowy posiada znaczącą redundancję, zaś kombinatoryczny – najmniejszą. We wszystkich modelach liczba różnych rozwiązań dopuszczalnych jest identyczna.

14.2 Pewne własności problemu

Podstawowe własności są wymieniane w związku z reprezentacją grafową problemu. Niezależnie od przyjętej metody modelowania, punktem wyjścia do rozważań jest acykliczny graf $G(\mathcal{W})$ lub $G(\pi)$. Opisane poniżej własności powstały głównie w kontekście problemów z kryterium $C_{\max}$, choć niektóre z nich mogą być adaptowane do innych funkcji kryterialnych. Wartość funkcji celu $C_{\max}$ odpowiada długośći najdłuższej drogi (krytycznej) w grafie $G(\mathcal{W})$ (odpowiednio $G(\pi)$). Niech $u = (u_1, \ldots, u_t)$ będzie ciągiem węzłów występujących w tej drodze. Ogólnie może istnieć wiele dróg krytycznych, wprowadzone pojęcia odnoszą się do każdej z nich. Droga w naturalny sposób może być dekomponowana (pocięta) na rozłączne odcinki odpowiadające operacjom wykonywanym kolejno na tej samej maszynie. Dokładniej interesują nas wszystkie zwarte podciągi kolejnych elementów z u postaci $(u_a, u_{a+1}, \ldots, u_b)$, gdzie $1 \leqslant a \leqslant b \leqslant t$, $\nu_{u_j} = k$, $j = a, \ldots, b$, $k = \nu_{u_a}$, $\nu_{u_{a-1}} \neq k \neq \nu_{u_{b+1}}$. Podciąg taki odpowiada kolejnym operacjom ze ścieżki krytycznej wykonywanej bez przerwy na tej samej maszynie. Ostatni warunek w definicji żąda maksymalnej rozpiętości podciągu co implikuje, że dekompozycji dokonano na najmniejszą liczbę odcinków posiadających podaną własność. Tak określony odcinek drogi krytycznej będzie dalej nazywany *blokiem operacji na maszynie* lub krótko *blokiem*. Interesują nas tylko bloki zawierające więcej niż jedną operację, $b-a > 1$. *Blokiem wewnętrznym* nazywamy ciąg operacji bloku bez pierwszej u_a i ostatniej u_b operacji tego bloku. Jeśli blok ma mniej niż trzy operacje, to jego blok wewnetrzny jest pusty.

Korzystając z pojęcia drogi krytycznej oraz bloków można pokazać pewne własności rozwiązań problemu z kryterium $C_{\max}$, będące podsumowaniem prac wielu autorów (patrz ich zestawienie w [3, 371]). Niech $\mathcal{W}$ będzie kompletną reprezentacją dopuszczalną, zaś u dowolna drogą krytyczną w $G(\mathcal{W})$. Zachodzą następujące własności.

A. Odwrócenie skierowania jednego (dowolnego) łuku dysjunktywnego leżącego na drodze krytycznej u generuje inną (kompletną) reprezentację dopuszczalną.

B. Odwrócenie skierowania jednego (dowolnego) łuku dysjunktywnego nie leżącego na drodze krytycznej u generuje reprezentację niedopuszczalną

lub dopuszczalną $\mathcal{W}'$ lecz nie lepszą, to znaczy długość drogi krytycznej w $G(\mathcal{W}')$ jest nie mniejsza niż w $G(\mathcal{W})$.

C. Odwrócenie skierowania jednego (dowolnego) łuku dysjunktywnego pomiędzy dwoma operacjami wewnętrznymi bloku generuje reprezentację dopuszczalną $\mathcal{W}'$ lecz nie lepszą wyżej podanym sensie.

D. Niech i, j będą dwoma kolejnymi opracjami pierwszego bloku na drodze krytycznej oraz j jest operacją wewnętrzną tego bloku. Odwrócenie skierowania łuku dysjunktywnego łączącego te operacje generuje reprezentację dopuszczalną $\mathcal{W}'$ lecz nie lepszą wyżej podanym sensie. Symetryczna własność zachodzi dla dwóch kolejnych operacji ostatniego bloku.

Niestety, brak jest analogicznych lub chociaż podobnych własności problemu gniazdowego w przypadku innych funkcji kryterialnych.

14.3 Schemat B&B

Mimo, iż w ostatnich latach osiagnięto znaczny postęp w budowie algorytmów B&B dla problemu gniazdowego [43,52], otrzymane rezultaty trudno uznać za zadowalające. Miarą dokonanego postępu jest fakt że problem praktyczny [94] FT10 o rozmiarze 10 zadań, 100 operacji, 10 maszyn, postawiony w roku 1963 doczekał się rozwiązania optymalnego po 26 latach właśnie za pomocą schematu B&B. Najlepszy znany algorytm dla problemu gniazdowego, wykorzystujący schemat, B&B, używa wiele specyficznych cech problemu, w tym tak zwane własności blokowe, charakteryzuje się znacznym stopniem skomplikowania, nietrywialna implementacją komputerową i jest w stanie rozwiązać w rozsądnym czasie problemy o rozmiarze poniżej 200 operacji [43]. Niestety, podany rozmiar problemu wyznacza aktualnie przybliżoną granicę praktycznej stosowalności algorytmów dokładnych, poza która objawiaja one niekontrolowana eksplozję obliczeń. Przesunięcie tej granicy będzie możliwe po znaczącym rozwoju teorii, zaś za główna przyczynę niedostatków powszechnie uważa się słabość metod wyznaczania dolnych ograniczeń oraz brak skutecznych własności eliminacyjnych. Stad, algorytmy dokładne tracą powoli swoje znaczenie w rozwiązywaniu przykładów o większym rozmiarze, na rzecz szybkich i dość dokładnych algorytmów przybliżonych. Tym niemniej, cykliczne systemy wytwarzania, charakteryzujące się niewielkimi rozmiarami przykładów praktycznych, a wymagające dokładnego (optymalnego) rozwiązania, są doskonałym obszarem zastosowania schematu B&B. Niestety, w chwili obecnej stosunkowo mało prac poświęcono takiemu podejściu.

Schematy B&B dla problemu gniazdowego z kryterium $C_{\max}$ są oparte

w zasadzie na czterech odmiennych podejściach: (1) historycznie najstarsze, oparte na eliminacyjnych własnościach drogi krytycznej, z podziałem binarnym poprzez odwrócenie skierowania pojedynczego łuku dysjunktywnego, generujące uszeregowania dosunięte w lewo [28]; (2) tradycyjnie, najbardziej popularne, oparte na budowie częściowego uszeregowania operacji od początku lub od końca, zwykle z wykorzystaniem schematu GT opisanego dalej, generujace uszeregowania aktywne [224]; (3) oparte na analizie klik w grafie dysjunktywnym, z wykorzystaniem dodatkowych silnych własności eliminacyjnych, z podziałem binarnym poprzez odwrócenie skierowania pojedynczego luku dysjunktywnego, generujace uszeregowania dosunięte w lewo [52]; (4) oparte na nie-binarnym dynamicznym schemacie podziału, korzystającym z rozbicia drogi krytycznej na bloki operacji, wsparte dodatkowymi silnymi własnościami eliminacyjnymi, w tym własnościami eliminacyjnymi drogi krytycznej i bloków zadań, generujące uszeregowania dosunięte w lewo [132]. Precyzyjny opis każdego z tych podejść wymaga wprowadzenia i opisania wielu specyficznych nietrywialnych własności teoretycznych. Ponieważ praktyczna przydatność schematu B&B dla problemu gniazdowego ciągle jeszcze pozostaje kontrowersyjna, zatem tylko odsyłamy zainteresowanego czytelnika do wymienionych pozycji literaturowych.

Jednym z ważniejszych zagadnień rozważanych w kontekście schematu B&B dla problemu gniazdowego oraz oceny jakości metod przybliżonych, są efektywne metody wyznaczania dolnych ograniczeń wartości funkcji celu. Ze względu na złożoność obliczeniową oraz kłopoty algorytmiczne, zdecydowanie najwięcej wyników otrzymano dla problemów z kryterium $C_{\max}$ oraz kryteriami podobnymi ($L_{\max}$, $W_{\max}$, etc.). Metody te różnią się między sobą zastosowaną relaksacją oraz złożonością obliczeniową. I tak, najprostsze z nich, relaksują łuki dysjunktywne nieskierowane (nie należące do reprezentacji częściowej) oraz wyznaczaja długość drogi w grafie zawierającym reprezentację częściową. Kolejna grupa metod wprowadza relaksację przepustowości wszystkich maszyn z wyjatkiem jednej. Otrzymujemy w ten sposób problem szeregowania na jednej maszynie $(1|r_i, q_i, prec, d_{ij}|C_{\max})$, z czasmi gotowości r_i, czasami dostarczania q_i, z relacją częściowego porządku $\mathcal{R}$ nałożoną na czasy oczekiwania pomiędzy operacjami połączonymi relacją poprzedzania. Wartości r_i, q_i, $\mathcal{R}$, d_{ij} wynikają z analizy odpowiednich dróg w grafie. Ponieważ tak otrzymany problem jest NP-trudny, stosowane są dalsze relaksacje, w tym prowadzące do problemu $1|r_i, q_i|C_{\max}$ oraz jego pochodnych omówionych szczegółowo w Rozdz. 9. W praktyce ponieważ problem $1|r_i, q_i|C_{\max}$ ma stosunkowo efektywny algorytm (patrz Rozdz. 9.9), mimo jego oczywistej NP-trudności często jest on także stosowany bezpośrednio jako umiarkowanie kosztowne narzędzie do wyznaczania dolnego ogranicze-

nia lub do konstrukcji bardziej zaawansowanych metod przybliżonych. W literaturze znane są także wysoce specjalizowane algorytmy dedykowane dla problemu $1|r_i, q_i, prec, d_{ij}|C_{\max}$. Krańcowo odmienna techniką jest relaksacja dualna, która omówiono bardziej szczegółowo w Rozdz. 14.2.

14.4 Algorytmy priorytetowe

Istnieją co najmniej trzy ogólne schematy heurystyczne, patrz przegląd w pracy [371], wszystkie bazujące na budowie częściowego uszeregowania operacji, w których można osadzać różne priorytetowe reguły wyboru. Ogólnie, reguły te określają preferecje dla wyboru kolejnej operacja do wykonywaniu, spośród pewnego zbioru operacji oczekujących na wykonywanie (zbiór operacji szeregowalnych, kolejka do stanowiska, itp. [279,388]. Wśród najczęściej wymienianych reguł są: SPT (shortest processing time) operacja z najkrótszym czasem wykonywania, LPT (longest processing time) operacja z najdłuższym czasem wykonywania, MWR (most work remaining) operacja o największej pozostałej pracochłonności liczonej sumą czasów wykonywania operacji technologicznych występujących po danej operacji LWR (least work remaining) operacja o najmniejszej pozostałej pracochłonności, MOR (most operations remaining) zadanie o największej liczbie pozostałych operacji, LOR (least operations remaining) zadanie o najmniejszej liczbie pozostałych operacji, ERT (earliest ready time) operacja najwcześniej dostępna, EDD (earliest due date) operacja najwcześniej kończona, FIFO (first in first out) w kolejności napływu, RANDOM (random) losowo.

Schemat GT

Pierwsza z wymienionych klas algorytmów priorytetowych bazuje na schemacie GT, który generuje drzewo rozwiązań (analogiczne jak w schemacie B&B) w celu rozstrzygania konfliktów wykonywania operacji żądających tej samej maszyny w nierozłącznych przedziałach czasu. Ogólna zasada algorytmu GT opiera się na tworzeniu częściowego uszeregowania operacji, poczynając od uszeregowania pustego, dołączając kolejno operacje z zachowaniem relacji porządku technologicznego. Niech $\mathcal{SH}$ będzie zbiorem operacji *szeregowalnych*, tzn. nieuszeregowanych, mających uszeregowane wszystkie poprzedniki w grafie poprzedzań, $(\mathcal{O},\mathcal{U})$ w pewnym kroku procedury. Oznaczmy przez r_j najwcześniejszy termin rozpoczęcia wykonywania operacji $j \in \mathcal{U}$. Terminy te możemy wyznaczyć jako długości odpowiednich dróg w grafie zawierającym łuki technologiczne $\mathcal{U}$ oraz łuki kolejnościowe wynikające z uszeregowania częściowgo (reprezentacja częsciowa). Dalej, identyfiku-

jemy operację v taką, że $(r_v + p_v) = \min_{j \in \mathcal{SH}}(r_j + p_j)$ oraz maszynę $k = \nu_v$. Wszystkie operacje $j \in \mathcal{SH}$, które spełniają warunek $r_j \leqslant r_v + p_v$, $\nu_j = k$, żądają dostępu do maszyny k w nierozłącznych przedziałach czasu i tworzą zbiór konfliktowy $\mathcal{K}_k$ ma maszynie k. Rozstrzygnięcie konfliku w schemacie GT odbywa się poprzez analizę wszystkich wariantów, co odpowiada wygenerowaniu wszystkich gałęzi drzewa skojarzonych z rozszerzeniem uszeregowania częściowego o operację j, $j \in \mathcal{K}_k$. Zauważmy, że analiza ta nie bierze pod uwagę stanu maszyny k (wolna, zajęta) w chwili podejmowania decyzji o szeregowaniu. Schemat GT umożliwia wygenerowanie wszystkich uszeregowań aktywnych.

Wymieniona klasa algorytmów priorytetowych bazuje na schemacie GT w tym sensie, że konfikt jest rozstrzygany jednoznacznie i autorytatywnie poprzez zastosowanie wybranej reguły priorytetowej. Stąd na każdym etapie w drzewie wariantów metody GT jest wybierany tylko jeden następnik oraz pomijane pozostałe, co przejściu od korzenia do pewnego węzła terminalnego wzdłuż jednej ścieżki. Pewnym rozszerzeniem tak otrzymanego algorytmu jest dopuszczenie do analizy na każdym etapie podejmowania decyzji, nie jednego lecz pewnej niewielkiej liczby wariantów podobnie jak w metodzie FBS. Ze względów implementacyjnych warianty te są często generowane sekwencyjnie, co odpowiada realizacji algorytmu priorytetowego z powrotami.

Uszeregowania częściowe

Druga klasa algorytmów bazuje na budowie częściowego uporządkowania operacji, począwszy od uszeregowania pustego, dodając w każdym kroku jedną szeregowalną operację. Wybór operacji do uszeregowania odbywa się według przyjetej reguły priorytetowej, przy czym musi ona uwzględniać nie tylko pilność operacji, ale także moment dostępności operacji. Do tego celu najbardziej nadają się reguły typu FCFS oraz ERT. Ponieważ podejscie to nie analizuje zbioru konfliktowego, jest szybsze jednak kosztem jakości generowanych rozwiązań. Metoda generuje rozwiązania dosunięte w lewo.

Symulacja zdarzeń

Trzecia klasa jest w swojej istocie przebiegiem symulacji komputerowej z dyskretnym zbiorem zdarzeń oraz określonymi regułami piorytetowymi obsługi kolejek do stanowisk. Reguły te mogą być różne dla różnych stanowisk, co powoduje możliwość wygenerowania dość dużego zbioru rozwiązań, spośród których najlepsze może zostać wybrane. Generowane są rozwiązania

dosunięte w lewo.

Algorytmy priorytetowe są jednym z częściej stosowanych narzędzi do rozwiązywania złożonych zagadnień szeregowania, jakim jest problem gniazdowy. Mimo, że dostarczane rozwiązania są względnie odległe od rozwiązań optymalnych w granicach do 30%, ich implementacja komputerowa jest względnie prosta. Pozwalają też na pewna elastyczność związaną ze zmiana reguł osadzanych w ogólnym schemacie algorytmu.

14.5 Algorytmy aproksymacyjne

Wprawdzie konstrukcja i analiza algorytmów tej klasy są dość złożona, przedstawiony poniżej algorytm [331] jest przykładem oryginalnego algorytmu ulosowionego dostarczającego dobrych intuicji dotyczących budowy innych algorytmów tej klasy.

Dla potrzeb algorytmu wprowadzimy następujące oznaczenia

$$LB_J = \max_{1 \leqslant i \leqslant n} \sum_{j=l_{i-1}+1}^{l_i} p_j \tag{14.11}$$

$$LB_M = \max_{1 \leqslant i \leqslant m} \sum_{j \in \mathcal{O}; \nu_j = i} p_j. \tag{14.12}$$

Wielkości LB_J i LB_M są klasycznymi dolnymi ograniczeniami problemu gniazdowego. Dalej oznaczmy, $p_{\max} = \max_{j \in \mathcal{O}} p_j$ oraz $o_{\max} = \max_{1 \leqslant i \leqslant n} o_i$. Algorytm podstawowy składa się trzech kroków:

1. Wygeneruj rozwiązanie (niedopuszczalne) S spełniające *tylko* wymagania porządku technologicznego, to znaczy $S_{l_i+j} = \sum_{s=1}^{j-1} p_{l_i+s}$. Rozwiązanie to ma wartość $C_{\max} = LB_J$, lecz więcej niż jedno zadanie może zostać przydzielone do maszyny w tym samym momencie czasu.

2. Zaburz terminy rozpoczęcia operacji każdego zadania i o wielkość Δ_i to znaczy $S_{l_i+j} := S_{l_i+j} + \Delta_i$, $j = 1, \ldots, o_i$, gdzie Δ_i jest całkowitą liczbą losową o rozkładzie równomiernym na przedziale $[0, LB_M]$, $i = 1, \ldots, n$.

3. "Rozciągnij" i "spłaszcz" otrzymane uszeregowanie tak, by w każdym momencie czasu na każdej maszynie było wykonywane nie więcej niz jedno zadanie.

Kluczowym elementem algorytmu jest Krok 3. Załóżmy, że dysponujemy rozwiązanie S długości L, w którym co najwyżej c zadań jest szeregowanych

równocześnie na każdej maszynie. Jeśli operacje mają jednostkowy czas wykonywania, rozciągnięcie rozwiązania jest trywialne. Wystarczy w tym celu każdą jednostkę czasu zastąpić c jednostkami czasu, przydzielając c operacji w sposób niekonfliktowy do kolejnych jednostek czasu. Takie postępowanie dostarcza rozwiązanie dopuszczalne o długości nie większej niż cL. Podobny zabieg można wykonać jeśli wykonywanie operacji jest przerywalne. W przypadku nieprzerywalnym, pokazano metodę umożliwiającą rozciągnięcie uszeregowania niedopuszczalnego długości L do rozwiązania dopuszczalnego o wartości $O(cL\log(p_{\max}))$.

Zdefiniujmy wpierw problem pomocniczy posiadający czasy wykonywania operacji zaokrąglone do najbliższej potęgi 2 jako $p'_j = 2^{\lceil \log_2 p_j \rceil}$. Odpowiednio, niech $p'_{\max} = \max_{j\in\mathcal{O}} p'_j$. Wychodząc od S można skonstruować S', które używa zmodyfikowanych czasów p'_j i jest co najwyżej 2 razy dłuższe od S; co więcej optymalne rozwiązanie tego nowego problemu jest nie więcej niż dwukrotnie dłuższe od rozwiązania optymalnego problemu oryginalnego. Przejdźmy zatem do konstrukcji żądanego rozwiązania wychodząc od S'.

Blokiem czasowym nazywamy przedział czasowy taki, że każda operacja zaczynająca się w tym przedziale ma długość nie większą niż długość całego przedziału. Definicja nie wymaga by wyspecyfikowane operacje kończyły się w podanym przedziale. Możemy podzielić S' na $\lceil \frac{L}{p'_{\max}} \rceil$ kolejnych bloków czasowych o wielkości $p'_{\max}$. Celem naszym jest podanie rekurencyjnego algorytmu rozciągającego każdy blok czasowy wielkości p, gdzie p jest potęgą 2, na ciąg *segmentów* o łącznej długości $p\log p$; wszystkie operacje uszeregowane w segmencie o długości T mają długość T i rozpoczynają się od początku tego segmentu.

W celu uszeregowania bloku czasowego B wielkości $p'_{\max}$ konstruujemy wpierw końcowy segment długości $p'_{\max}$, a następnie krótsze segmenty poprzedzające poprzez rekurencyjne wywołania algorytmu. Dla każdej operacji długości $p'_{\max}$, która zaczyna się w B szeregujemy tą operację tak, by zaczynała się na początku końcowego segmentu i następnie usuwamy z B. Po tym kroku, każda pozostałą operacja zaczynajaca się w B ma długość co najwyżej $p'_{\max}/2$, zatem B może zostać podzielony na dwa bloki czasowe B_1 i B_2, każdy o długości nie większej niż $p'_{\max}/2$, w stosunku do których możemy powtórzyć rekurencyjnie nasze postępowanie. Otrzymane rozwiązanie posiada co najwyżej c operacji o jednakowym czasie trwania T w segmencie o wielkości T. Operacje te są następnie szeregowane kolejno, jedna po drugiej, w celu zapewnienia jednostkowej przepustowości maszyny.

Wykonując szereg złożonych analiz amatematycznych dotyczących redukowalności $p_{\max}$ oraz liczby zadań, ostateczny rezultat gwarantuje uzyskanie

z największym prawdopodobieństwem rozwiązania problemu gniazdowego o wartości $C_{\max}$ nie większej niż $O(\frac{\log^2(o_{\max}m)}{\log\log(o_{\max}m)}C^*_{\max})$. Nie wnikając szczegółowo w rzeczywisty cel określenia błędu z dokładnością do rzędu funkcji, wartość *wyłącznie* współczynnika $\frac{\log^2(o_{\max}m)}{\log\log(o_{\max}m)}$ dla wspomnianego przykładu testowego FT10 ($n = 10$, $m = 10$, $o = 100$) wynosi ponad 15. Zakładając regularna strukturę przykładu (tzn. $n = o_{\max}$) minimalna osiągana wartość tego współczynnika jest w przyblizeniu równa 1.9 (dla $m = 3$) oraz jest więkasza od 2 dla $m > 4$. Współczynnik ten wzrasta zdecydowanie wolnej niz n ze wzrostem rozmiaru problemu. Mimo, iż jest to postęp w stosunku do oszacowania n, otrzymane gwarancje wciąż pozostają niedostateczne. Gwarancja błędu o wartości błędu o wartości mniejszej niż 2 (porównaj z błędem eksperymentalnym 1.05 dla TS) oznacza błąd względny rzędu 100% w odniesieniu do rozwiązania optymalnego.

Zaproponowana metoda prowadzi w dalszym ciągu do konstrukcji wielomianowego algorytmu deterministycznego, który wyznacza rozwiązanie o wartości $C_{\max}$ nie większej niż $O(\log^2(o_{\max}m)C^*_{\max})$. Algorytm ten jest "odlosowioną" wersją algorytmu opisanego na wstępie tego rozdziału, posiadającą zmieniony Krok 2. W tym przypaku wrtości Δ_i są wybierane deterministycznie w przedziale $[0, LB_M/\log(o_{max}m)]$, tak by otrzymać uszeregowanie posiadajace nie więcej niż $O(\log(o_{\max}m))$ operacji wykonywanych równocześnie na każdej maszynie w dowolnym momencie czasu. Konstrukcja żądanego uszeregowania jest przeprowadzana przy pomocy pomocniczego zadania PL. Nie wnikając w dalsze matematyczne aspekty tego problemu, zauważmy jedynie, że dla przykładu testowego FT10 wartość współczynnika $\log^2(o_{\max}m)$ wynosi ponad 40.

Rozwiązywanie problemów gniazdowych przez ich aproksymację wydaje się ciągle jeszcze niedostatecznie efektywne z praktycznego punktu widzenia, co przyznają często sami autorzy algorytmów. Podstawowy lecz niegroźny pesymizm jest związany z elementarnym rezultatem dotyczącym *nieaproksymowalności* [385], ważnym już dla problemu przepływowego: jeśli $P \neq NP$ to nie istnieje wielomianowy algorytm ϵ-aproksymacyjny dla problemu gniazdowego z kryterium $C_{\max}$ dla $\epsilon < 1.25$. W praktyce oznacza to niemożność zagwarantowania błędu wzglednego aproksymacji poniżej 25% w odniesieniu do rozwiązania optymalnego, co w porównaniu z eksperymentalnie otrzymanym błędem poniżej 1-5% dla metod LS wydaje się znaczacym mankamentem. Dalej, wszystkie znane dotychczas algorytmy przybliżone gwarantowały błąd ϵ niw większy niż n co jest już skrajnym pesymizmem. (Algorytm wybierający rozwiązanie losowo jest także n-aproksymacyjny). Rezultat dotyczący nieaproksymowalności został otrzymany dla przykładu problemu

gniazdowego o niewielkim rozmiarze, stąd przypuszczenie, iż mogą istnieć algorytmy aproksymacyjne zachowujące się dobrze w sensie asymptotycznym, dla duzych przykładów problemów. Co więcej, w sensie teoretycznym, *każdy* algorytm ϵ-aproksymacyjny dla $\epsilon < n$ należy uznać za krok pozytywny w kierunku konstrukcji efektywnych algorytmów aproksymacyjnych dla rozważanego problemu. Jak do tej pory, otrzymane wyniki są teoretycznie interesujące, choć znane oszacowania błędu podawane z dokładnością do rzędu funkcji, w praktyce są wciąż mało użyteczne.

14.6 Poszukiwania lokalne

Podstawowym problemem w algorytmach poszukiwań lokalnych jest sposób budowy sąsiedztwa, wynikające stąd jeg własności oraz metoda oceny rozwiązań sąsiedztwa. Teoretycznie możliwe jest zaprojektowanie wielu różnych sąsiedztw, w zależności od przyjetego sposobu modelowania rozwiązań (dysjunktywne, kombinatoryczne, listowe), tak też jest w praktyce. W literaturze występuja głównie otoczenia mające swoje źródło w wymienionych podejściach, korzystajace ze szczególnych własności takich jak na przykład te wymienione w Rozdz. 14.2. Niestety wszystkie zgłoszone propozycje muszą mieć na względzie problem dopuszczalności generowanych rozwiązań sąsiednich w otoczeniu, co zwykle zwiększa złożoność obliczeniową odpowiednich procedur przeglądu. (Model listowy z listą zadań nie korzysta z żadnych własności i nie musi badać dopuszczalności rozwiązań.) Spotykane są przy tym cztery odmienne podejścia: (1) dla każdego wygenerowanego rozwiązania badana jest kosztownie jego dopuszczalność; (2) wygenerowanie rozwiązania jest poprzedzone niekosztownym testem dopuszczalności; (3) funkcja oceny rozwiązania została dobrana tak by zwracać nieatrakcyjną wartość dla rozwiązań niedopuszczalnych; (4) generowane są tylko rozwiązania posiadajace teretyczną gwarancję dopuszczalności. Ocena rozwiązań w sąsiedztwie jest wykonywana poprzez: (a) obliczenie explicite wartości funkcji celu; (b) niekosztowne oszacowanie wartości funkcji celu, zwykle wyliczane w czasie $O(1)$.

Najprostszy rodzaj sąsiedztw (N1) bazuje na zmianie skierowania jednego łuku dysjunktywnego leżącego na (dowolnej) ścieżce krytycznej; korzysta z Własności A i B. Bardziej zaawansowane sąsiedztwo (N1.1) odwraca skierowanie tylko tych łuków na ścieżce krytycznej, które łączą operacje przyległe; korzysta z N1 oraz części Własności C. Kolejne sąsiedztwo wyłącza z poprzedniego łuki łączące wewnętrzne operacje bloku; korzysta z N1 oraz dodatkowo Własności C. Następna propozycja (N1.2) wyłącza z poprzed-

niego otoczenia N1.1 rozwiązania odpowiadające warunkom wymienionym w punkcie D. Wszystkie wymienione otoczenia gwarantują, że wygenerowane rozwiązania będą dopuszczalne.

Kolejne otoczenia są budowane przez zmianę skierowania więcej niż jednego łuku dysjunktywnego. Pierwsza propozycja (N2) odpowiada badaniu wszystkich 3! permutacji operacji leżących na ścieżce krytycznej $\underline{s}_i$, i, $\overline{s}_i$ (z definicji operacje te muszą być wykonywane na tej samej maszynie). Niestety potrzebne jest badanie dopuszczalności tak generowanych rozwiązań. Kolejna propozycja (N2.1) wyłącza z rozwiazań trójki dla których i, $\overline{s}_i$ bądź $\underline{s}_i$, i są równocześnie operacjami wewnetrznymi bloku. Następne otoczenie (N3) jest otrzymywane poprzez przesunięcie operacji z bloku bezpośrednio przed lub bezpośrednio za blok. Jeżeli takie działanie powoduje otrzymanie rozwiązania niedopuszczalnego, przesuwanie jest kontynuowane w odpowiednim kierunku do chwili uzyskania rozwiązania dopuszczalnego. Zauważmy, że rozwiązania generowane tymi metodami mają znaczne zmiany skierowania równocześnie wielu łuków dysjunktywnych w modelu dysjunktywnym, oraz są stosunkowo łatwo interpretowane w modelu kombinatorycznym za pomocą techniki INS (wytnij i wklej). Otoczenia N2, N2.1, N3 zmieniają lokalizację pojedynczej operacji.

Kolejne otoczenia powoduja zmiany położenia jednocześnie kilku operacji. Otoczenie (N5) rozważa dalej tylko operacje pochodzące z bloku i odwraca kolejność równocześnie trzech par operacji: (1) i, j, gdzie $j = \overline{s}_i$ (ta oraz następne są wykonywane z wyjątkiem gdy obie należą do bloku wewnętrznego); (2) $\underline{s}_{\underline{t}_j^k}$, $\underline{t}_j^k$, dla pewnego $k \geqslant 1$; (3) $\overline{t}_i$, $\overline{s}_{\overline{t}_i}$. Symbol $\underline{t}_j^k$ jest definiowany rekurencyjnie: $\underline{t}_j^0 = \underline{t}_j$, $\underline{t}_j^{k+1} = \underline{t}_{\underline{t}_j^k}$.

Krańcowymi pomysłami są drastyczne zmiany lokalizacji wielu operacji równocześnie. Niech m^* będzie maszyną przez którą przechodzi droga krytyczna. Sąsiedztwo (N4) otrzymujemy generując wszystkie skierowania łuków dysjunktywnych na m^* prowadzące do rozwiązań dopuszczalnych. Inny pomysł (N6) polega na generowaniu wszystkich dopuszczalnych rozwiązań poprzez zmianę skierowania łuków dysjunktywnych na $m - k$ maszynach, gdzie $k \ll m$. Jest to jedno z większych i bardziej kosztownych obliczeniowo sąsiedztw.

Samodzielna grupę stanowią sąsiedztwa wykorzystujące opisany wcześniej schemat GT, zwykle wbudowany w algorytm oparty na technice FBS.

Algorytmy LS uważane są za jedne z bardziej efektywnych narzędzi do przybliżonego rozwiązywania problemów gniazdowych. Niestety, najprostsze jednoprzebiegowe metody DS są w tym przypadku względnie mało skuteczne. Dobre wyniki otrzymuje się łącząc LS z odpowiednim mechanizmem

rozpraszania poszukiwań oraz wprowadzając techniki wielostartowe.

14.7 Metoda przesuwanego wąskiego gardła

Metoda SB (Shifted Bottleneck) buduje reprezentację częściową rozpoczynając od reprezentacji pustej, wykonując m-etapowy proces. W każdym etapie ustalana jest kolejność wykonywania operacji tylko na jednej wybranej maszynie, nie dokunując żadnych zmian na innych maszynach, zaś skierowania łuków dysjunktywnych na tej maszynie są dostosowywane tak, by reprezentowały otrzymaną kolejność. Maszyna ta jest traktowana tymczasowo, na czas trwania etapu, jako maszyna krytyczna (bottleneck machine), patrz Rozdz. 9.4. W celu szeregowania wybranej maszyny formułowany jest pomocniczy problem jednomaszynowy $1|r_i, q_i, prec, d_{ij}|C_{\max}$ (lub $1|r_i, q_i|C_{\max}$ w wersji wcześniejszej tego algorytmu), który następnie jest rozwiązywany algorytmem dokładnym. Wartości r_i, q_i, $\mathcal{R}$, d_{ij} wynikają z analizy długości odpowiednich dróg w grafie zawierającym częściowe uporządkowanie. W końcu każdego etapu jest wywoływany pomocniczy algorytm popraw lokalnych, ze strategią najlepszej poprawy, wykorzystujacy sąsiedztwo N4, badajace zmiany orientacji wszystkich łuków dysjunktywnych ustalonych wcześniej na innych maszynach. Tak określony ogólny schemat algorytmu nie precezuje kolejności analizowania maszyn, metody rozwiązania pomocniczego problemu jedno-maszynowego, itd.

Podstawowy wariant algorytmu SBI analizuje maszyny zgodnie z malejącymi wartościami funkcji celu rozwiązań odpowiednich problemów jednomaszynowych formułowanych dla problemu wyjściowego. Po uszeregowaniu jednej maszyny iteracyjna procedura popraw z sąsiedztwem $N4$ jest stosowana w trzech cyklach. W każdym cyklu każda uszeregowana wcześniej maszyna jest rozważana ponownie i przeszeregowywana z użyciem N4. W pierwszym cyklu kolejność analizy maszyn jest zgodna z kolejnością początkową. W drugim i trzecim cyklu maszyny są analizowane w kolejności wynikajacej z wartości problemów jednomaszynowych wyznaczonych w cyklu poprzednim. Gdy wszystkie maszyny zostaną już uszeregowane, cykle procedy popraw są stosowane tak długo jak długo występuja poprawy.

Istnieje wiele wariantów algorytmu SB różniących się niewielką ilością szczegółów oraz głównie wartościami parametrów strojących. Najbardziej zaawansowany wariant (SBII) generuje ograniczone drzewo kolejności analizowania maszyn oraz dla każdej takiej kolejności stosuje SBI.

Metody SB są często polecane jako najbardziej efektywne mim znaczących wad. Są złożone implementacyjnie oraz objawiają eksplozje obliczeń

charakterystyczną dla CB przy wzroście rozmiaru problemu.

14.8 Symulowane wyżarzanie

Praktycznie możliwe jest wykorzystanie dowolnego otoczenia wymienionego w Rozdz. 14.6 do generowania rozwiązania zaburzonego w schemacie SA opisanym w Rozdz. 8. Wykonane badania testowe dla różnych wariantów algorytmu SA z otoczeniami N1 oraz N5.1, z kryterium $C_{\max}$ przy różnych metodach strojenia algorytmu SA, wykazały umiarkowany optymizm w ocenach jakość/czas (**??**). Z tego też względu kolejne propozycje zmierzały w kierunku algorytmów hybrydowych, otrzymanych głównie poprzez wbudowanie doadatkowej procedury poszukiwania lokalnego (pełny przegląd subotoczenia) poprzedzającej wybrór rozwiązania zaburzonego. Zastosowanie własności blokowych jest wysoce rekomendowane w tym celu.

Wydaje się, że dla problemu gniazdowego z innymi kryteriami addytywnymi, jak na przykład $\sum C_i$, podejscie SA może być względnie dobrą alternatywą. Argumentów "za" jest kilka. Żadne z zdefiniowanych otoczeń nie jest odpowiednie dla wymienionego przypadku bowem wartość funkcji celu zależy od skierowania *prawie wszystkich* łuków dysjunktywnych w grafie (dokładniej tych, które wpływają na wartości $\mathcal{C}_i$). Stąd zachodzi potrzeba znacznego powiększenia otoczenia, co implikuje duże koszty jego kompletnego przeszukiwania w przypadku użycia metod DS, TS. Losowe próbkowanie rozwiązań wykonywane przez SA zmniejsza znacznie koszt wykonania pojedynczej iteracji.

14.9 Poszukiwanie z zakazami

Jak do tej pory, rozważono kilka wariantów schematu TS z różnymi sąsiedztwami N1, N1.1, N1.2, N2 dla kryterium $C_{\max}$. Tak naprawdę, otrzymywane wyniki zależą nie tylko od struktury sąsiedztw ale także od sposobu wprowadzania zabronień, badania statusu ruchu, mechanizmu rozpraszania poszukiwań. Te z kolei są dość specyficzne dla każdej podanej implementacji. Zaskakująco dobrą skuteczność można otrzymać już przy niewielkich środkach stosowanych w dziedzinie TS. Jednakże dużo lepsze efekty uzyskuje się wprowadzając dodatkowe mechanizmy intensyfikacji i rozpraszania oparte na przykład na technice skoku powrotnego do atrakcyjnych obszaró poszukiwań, akceleracji poszukiwań (dekompozycja i agrregacja obliczeń). Najlepszy znany obecnie algorytm tego typu TSAB [268] pokazuje, że własność łączności (connectivity) nie jest konieczna do osiagania dobrych wy-

ników numerycznych. Algorytm TSAB stosuje otoczenie N1.2 i posiada, jak dotychczas, najkorzystniejszą relację czas obliczeń/dokładność według niezależnych ocen z pracy [371]. TSAB dla wspomnianego uprzednio przykładu FT10 wyznacza rozwiązanie optymalne (mimo braku gwarancji jego optymalności) w czasie 0.5 sec na współczesnych PC. Pozwala on także rozwiązywać przykłady o rozmiarze 2,000 czynności na 50 maszynach w czasie minut z błędem względnym 4-5%. Algorytmy TS są jednymi z najprostszych implementacyjnie i najbardziej skutecznych w chwili obecnej.

14.10 Spełnianie ograniczeń

Istnieje kilka sposobów formułowania problemu gniazdowego w kategoriach CSP. Poniżej przedstawiamy wyniki z pracy [61]. Najbardziej oczywistym i naturalnym jest model, w którym rozwiązanie jest reprezentowane wektorem terminów rozpoczęcia operacji $S = (S_1, \ldots, S_o, S_*)$ będących jednocześnie zmiennymi decyzyjnymi problemu CSP. Składowa S_* reprezentuje wartość kryterium $C_{\max}$. Zmienne te podlegają ograniczeniom zakresu, np. $r_j \leqslant S_j \leqslant d_j$, $j \in \mathcal{O} \cup \{*\}$, ograniczeniom wynikającym z relacji technologicznych wykonwania operacji oraz z jednostkowej przepustowości maszyn. Zakres dopuszczalnej wartości zmiennych zwany *dziedziną zmiennych* jest dynamiczny (tymczasowy) i zawężany systematycznie poprzez dodatkowo wprowadzane warunki ograniczające o różnorodnym charakterze, tzw. *propagacja ograniczeń*. Dziedziną składowej S_j oznaczmy przez S_j, przy czym zbiór S_j nie musi być spójny. Problem, dla którego, przy danych ograniczeniach, nie istnieje rozwiązanie dopuszczalne jest it niezgodny. Badanie niezgodności jest możliwe, poprzez analizę dziedziny, odpowiednich dolnych ograniczeń, specyficznych kryteriów eliminacji, oraz warunków wyrażalnych w postaci "jeśli ... to". Stwierdzenie niezgodności powoduje odrzucenie problemu i powrót (backtrack), tzn. przejście do analizy pewnego problemu o mniej restryktynych lub opełniających ograniczeniachm wygenerowanego wcześniej. Przeciwnie, stwierzenie zgodności zwykle pozwala skonstruować rozwiązanie dopuszczalne częściowe lub komletne rozwiązujące ograniczenia (posługując się odbudowaną heurezą, choć nie zawsze jest to możliwe. Odpowiedź na pytanie czy istnieje rozwiązanie dopuszczalne dla ograniczeń teoretycznie jest problemem silnie NP-trudnym. Problem zgodny podlega w dalszym ciągu podziałowi poprzez wprowadzenie pewnego wybranego ograniczenia, w sposób analogiczny do podziahemacie B&B, powodując zamknięcie cyklu iteracyjnego algorytmu CSP.

Powyższy, dość ogólny schemat postępowania nie precyzuje wielu istot-

nych szczegółów. Czytelnik zauważy, że istnieje wiele podobieństw tej metody do schematu B&B, stąd wszystkie stosowane tam metody konstrukcji dolnych ograniczeń moga być też tutaj zastosowane. Podobnie jak z algorytmami przybliżonymi, które można zastosować do konstruowania rozwiązań dopuszczalnych. Niezależnie od powyższego, dziedzina CSP oferuje pewne specyficzne narzędzia, do opisu problemu, wprowadzania ograniczeń i badania niezgodności.

Jedną z bardziej efektywnych technik wprowadzania i propagacji ograniczeń, jest metoda *delegowania ograniczeń poprzedzania* (precedence constraint posting, PCP) bazująca na dysjunktywnej reprezentacji rozwiązań. PCP może być interpretowana jako *ogólna sieć chwilowych ograniczeń* (TCN) zdefiniowanych na zmiennych ciągłych $S = (S_1, \ldots, S_o, S_*)$ oraz zbiorze *jednoargumentowych* lub *dwuargumentowych* ograniczeń. Składowa S_i może reprezentować *punkt* (moment czasowy S_i równoważny $C_i = S_i + p_i$ lub *przedział* (wykonywanie operacji w przedziale $[S_i, C_i]$). Ograniczenia moga być *ilościowe* lub *metryczne*. Ograniczenie ilościowe jest przedstawiane dysjunkcją $(S_i\alpha_1 S_j) \vee \ldots \vee (S_i\alpha_k S_j)$, zapisywaną krótko $S_i\{\alpha_1, \ldots, \alpha_k\}S_j$, gdzie α_1 jest *elementarnym ograniczeniem ilościowym*. Dopuszcza się trzy typy elementarnych ograniczeń ilościowych:

1. "przedział–przedział"; zawiera 13 podstawowych relacji zachodzących pomiędzy parami przedziałów $[S_i, C_i]$ oraz $[S_j, C_j]$: *przed* ($C_i \leqslant S_j$); *za* ($C_j \leqslant S_i$); *złączony* ($C_i = S_j$); *złączony pośrednio* ($C_i = \Delta_j$); *nakładający się* (*nierozłączny*) ($[S_i, C_i] \cap [S_j, C_j] \neq \emptyset$); *nakładający się w przedziale o długości* $b-a$ ($[S_i, C_i] \cap [S_j, C_j] = [a, b]$); *w czasie trwania* ($S_j < S_i$, $C_i < S_j$); *zawarty w* ($[S_i, C_i] \subseteq [S_j, C_j]$); *zgodne rozpoczęcia* ($S_i = S_j$); *rozpoczyna się przed* ($S_i < S_j$); *zgodne zakończenia* ($C_i = C_j$); *kończy się przed* ($C_i < C_j$); *identyczny* ($[S_i, C_i] = [S_j, C_j]$).

2. "punkt-punkt": relacje są określone za pomocą klasycznych nierówności liczbowych $<, =, >$,

3. "punkt-przedział" lub "przedział-punkt": zawiera 10 relacji zachodzących pomiędzy punktem S_i a przedziałem $[S_j, C_j]$: *przed* ($S_i < S_j$); *w chwili rozpoczęcia* ($S_i = S_j$); *w czasie trwania* ($Sj < S_i < C_j$); *w chwili zakończenia* ($S_i = C_j$); *za* ($S_i > C_j$); oraz ich inwersje.

Inne typy relacji mogą być traktowane jako złożenie podanych.

Ograniczenie metryczne jest reprezentowsne przez rozłącznych przedziałów $\{\mathcal{I}_1, \ldots \mathcal{I}_k\}$, gdzie $\mathcal{I}_s = [a_s, b_s]$. Wymienia się dwa typy ograniczeń metrycznych. Jednoargumentowe, nałożone na zmienną S_j, ograniczają jej dziedzinę, $S_j = \bigcup_{s=1}^{k} \mathcal{I}_s$. Dwuargumentowe, nałożone na parę zmiennych S_i,

S_j, ograniczają ich wzajemną odległość, $S_j - S_i \in \bigcup_{s=1}^{k} \mathcal{I}_s$. Wprowadzając sztuczną operację początkową o zerowym czasie trwania oraz odpowiadający jej początkowy S_0, każde ograniczenie jednoargumentowe może być wyrażone jako ograniczenie dwuargumentowe.

Sieć TCN określa graf skierowany w którym każdy wierzchołek reprezentuje zmienną S_i zaś krawędź reprezentuje ograniczenie pomiędzy zmiennymi S_i oraz S_j. Wektor S jest rozwiązaniem jeśli spełnia wszystkie ograniczenia ilościowe i metryczne. Sieć jest zgodna jeśli istnieje co najmniej jedno rozwiązanie dopuszczalne. Rozwiązanie problemu polega na przeglądzie rozwiązań reprezentowanych siecią TCN dla wszystkich możliwych ograniczeń generowanych. Nie wnikając w szczegóły generowania różnych sieci TCN wystarczy stwierdzić, że istnieje metoda analizy (etykietowania) sieci, która pozwala na systematyczny i wyczerpujący przegląd rozwiązań.

Bazowy PCP model używa *warunków poprzedzania* wynikających z analizy ograniczeń. Najczęściej są one oparte na pojęciu *luzu czasowego* (slack), lub jego uogólnień, dla par operacji wykonywanych na tej samej maszynie, dla których nie ustalono jeszcze kolejności wykonywania. Przykładowo, jeśli luz pary operacji (i, j) definiowany jako $L_{ij} = d_j - r_i - (p_i + p_j)$, ma własność $L_{ij} < 0$, to operacja i nie może zostać uszeregowana przed operacją j. Korzystając z podanej definicji możemy sformułować oczywiste warunki: (a) jeśli $L_{ij} \geqslant 0$ oraz $L_{ji} < 0$ to musi zachodzić $i \rightarrow j$, (b) jeśli $L_{ij} < 0$ oraz $L_{ji} \geqslant 0$ to musi zachodzić $j \rightarrow i$, (c) jeśli $L_{ij} < 0$ oraz $L_{ji} < 0$ to stan poszukiwań jest niezgodny, (d) jeśli $L_{ij} \geqslant 0$ oraz $L_{ji} \geqslant 0$, to nie sposób, na danym etapie, ustalić jednoznacznie kolejności. Propagacja ograniczeń wymaga dokonania wyboru, która spośród par operacji wykonywanych, na tej maszynie, o nieokreślonej jeszcze kolejności wykonywania, powinna podlagać analizie w danym kroku. Zwykle wybiera się parę posiadającą najmniejszą swobodę, to znaczy z minimalną wartością wyrażenia $\omega_{ij} = \min\{L_{ij}, L_{ji}\}$. W przypadkach znacznej rozpiętości liczb L_{ij} można dokonać pewnego zabiegu normalizacji, re-definując wielkość $\omega_{ij} = \min\{L_{ij}, L_{ji}\}/D$, gdzie $D = \min\{L_{ij}, L_{ji}\}/\max\{L_{ij}, L_{ji}\}$.

Można skonstruować kilka algorytmów mających swoje źródło w PCP. Najprostszy (S-PCP), nie realizujacy powrotów, odpowiada pojedynczemu przejściu od korzenia drzewa rozwiązań do pewnego węzła terminalnego wzdłuż jednej wybranej ścieżki. Jest najmniej kosztowny, lecz może nie dostarczyć żadnego rozwiązania w chwili zakończenia działania. Bardziej wyrafionowany (R-PCP) jest minimalnym rozszerzeniem S-PCP gwarantującym uzyskanie rozwiązania dopuszczalnego. Wykrycie przypadku (c) dla pewnej pary (i, j) implikującego niezgodność stanu poszukiwań i powrót powoduje przejście do trybu obsługi wyjatkowej. Decyzja (i, j) powodująca niezgod-

ność jest zawieszana chwilowo, pozwalając na kontynuację poszukiwań. Gdy rozwiązanie dopuszczalne zostanie znalezione, zawieszona decyzja zostaje przywrócona lecz dane problemu podlegają relaksacji: terminy d_i, d_j zostają zwiększone o wartość $\max\{L_{ij}, L_{ji}\}$. Wieloprzebiegowy wariant M-PCP stosuje wielokrotnie S-PCP lub R-PCP dla wartości d_* wybieranych według techniki szukania binarnego na przedziale $[LB, UB]$, gdzie UB i LB są odpowiednio górnym i dolnym ograniczeniem wartości $C_{\max}$ dla problemu gniazdowego. Odpowiednie wartości można znaleźć stosując algorytmy priorytetowe (dla UB) oraz sekwencje zrelaksowanych problemów jednomaszynowych (dla LB). Ponieważ d_* jest ograniczeniem na wartość funkcji celu, zastosowanie jednego przebiegu R-PCP dostarczy rozwiązania spełniającego żądaną własność, lub też nie (jeśli zajdzie potrzeba użycia relaksacji).

Podejście CSP ma pewne zalety i wady. W praktyce, problemy gniazdowe i pokrewne, posiadają wiele dodatkowych uwarunkowań technologicznych, specyficznych dla każdego procesu produkcyjnego. Ograniczenia te zwykle powodują eliminację znacznej liczby rozwiązań, przyśpieszając tym samym proces obliczeniowy. Algorytmy CSP są interesującą propozycją dla tego typu problemów, bowiem umożliwiają stosunkowo szybka eliminację większości (nieperspektywicznych) rozwiązań. Niestety, wad można znaleźć nieco więcej. Większość algorytmów CSP można wyrazić w kategoriach schematu B&B, zaś odpowiednie warianty S-PCP, R-PCP, M-PCP jako realizację techniki poszukiwania strumieniowego (beam search). Stąd dziedziczą one wszystkie negatywne cechy swoich przodków. Jedynie w przypadku formułowania warunków ograniczających niewyrażalnych w kategoriach znanych dolnych ograniczeń problemu gniazdowego, schemat CSP byłby korzystniejszy, a to rzadko ma miejsce, niestety. Wyniki numeryczne dla serii przykładów testowych, w porównaniu do początkowych schematów SB, są umiarkowanie gorsze. Są one tym bardziej nieobiecujące dla najnowszych schematów SB oraz najlepszych technik TS.

14.11 Poszukiwanie ewolucyjne

Podstawowymi trudnościami w podejściu GA do problemu gniazdowego są sposób kodowania cech rozwiązania w chromosomie oraz specyficzne operatory genetyczne. Praktycznie nie spotyka się kodowania binarnego. Najczęściej chromosom ma formę ciągu symboli z pewnego zbioru i odpowiada permutacji lub ciągowi permutacji. Wielkości te reprezentują kolejność realizacji operacji na poszczególnych maszynach bądź relację kolejnościową pomiędzy operacjami. Ponieważ nie wszystkie tak zdefiniowane chro-

mosomy odpowiadają rozwiązaniu dopuszczalnemu problemu gniazdowego, wprowadzany jest *dekoder* przekształcający kod chromosomu w rozwiązanie dopuszczalne. Chromosom, który nie może być zdekodowany jest *nielegalny*. Do generowania legalnych potomków z legalnych rodziców stosowane są specyficzne operatory genetyczne. Ponieważ ogólny schemat algorytmu GA przedstawiony w Rozdz. ?? zachowuje swoją ważność, ograniczmy się do dalej do prezentacji operatorów.

Operatory zaadaptowane

W Rozdz. 12.11 omówiono operatory genetyczne dla rozwiązań reprezentowanych pojedynczą permutacją, bez zależności kolejnościowych pomiędzy zadaniami. Operatory te mogą one być zastosowane, po pewnej modyfikacji, również do omawianego problemu gniazdowego. Niech $\pi \in \Pi$ będzie rozwiazaniem dopuszczalnym omawianego problemu zaś niech $\sigma \in \Pi(\mathcal{O})$ będzie permutacją na zbiorze operacji $\mathcal{O}$. Wówczas dla każdego $\pi \in \Pi$ istnieje co najmniej jedna permutacja $\sigma \in \Pi(\mathcal{O})$ taka, że porządek wykonywania operacji na maszynach jest zgodny w π i σ. Permutacja σ jest porządkiem topologicznym wierzchołków grafu $G(\pi)$, tych zaś może być wiele. Odwrotnie, każdej σ odpowiada dokładnie jedna kolejność $\pi \in \Pi$. Można ją otrzymać dokonując "rozrzucenia" operacji na poszczególne maszyny w kolejności ich występowania w σ. Mimo iż $\pi \in \Pi$ wyznaczone jest jednoznacznie, otrzymane kolejność nie musi być dopuszczalna, bowiem wymagana jest jeszcze acykliczność $G(\pi)$. Stąd stosując operatory z Rozdz. 12.11 należało by się skoncentrować wyłącznie na do-projektowaniu problemowo-zorientowanego dekodera.

Operatory problemowo-zorientowane

Odmiennym podejściem jest wprowadzenie problemowo-zorientowanych operatorów genetycznych [59] uwzględniających pewne cechy charakterstyczne problemu dla zapewnienia legalności i dopuszczalności.

Operator wymiany podsekwencji SEX (subsequence exchange crossover) odnosi się do rozwiązania reprezentowanego ciągiem permutacji, w którym każda permutacja odnosi się do oddzielnej maszyny. *Podsekwencją* nazywamy zbiór zadań (operacji) wykonywanych kolejno po sobie na pewnej maszynie i występującą równocześnie u obu rodziców lecz niekoniecznie w tej samej kolejności. Kreowanie potomka polega na skopiowaniu genów z wybranej podsekwencji jednego rodzica (w oryginalnej kolejności i położeniu) oraz skopiowaniu pozostałych genów od drugiego rodzica. Ponieważ otrzymany

potomek nie musi być legalny, przeprowadza się korekcję jego genomu przy użyciu pewnej procedury, np. GT opisanej dalej.

Operator porządkowy krzyżowania zadań JOX (job-based order crossover) jest pewną modyfikacją operatora SEX. Dopuszcza on wybór podsekwencji zawierających zadania niekoniecznie wykonywane kolejno po sobie. Potomek jest tworzony poprzez skopiowanie od jednego z rodziców wybranych genów na pozycje, na których one wystąpiły u tego rodzica. Wolne pozycje są wypełniane pozostałymi genami pochodzącymi od drugiego rodzica w kolejności ich występowania na chromosomie tego rodzica. Korekcja genomu jest również niezbędna i w tym przypadku.

Operator wymiany częściowego rozwiązania PSEX (partial schedule exchange crossover) odwołuje się do rozwiązania częściowego jako naturalnego blokowego składnika tworzącego rozwiązanie kompletne. Rozwiązanie częściowe jest identyfikowane jako posiadające to same zadanie na początku i na końcu sekwecji częściowej. Rozwiązanie częściowe jest wybierane u jednego z rodziców oraz odpowiednio dobierane u drugiego rodzica. Potomek zawiera skopiowane rozwiązanie częściowe jednego z rodziców oraz pozostały materiał genetyczny od drugiego z rodziców. Geny opuszczone oraz powielone są korygowane w celu otrzymania rozwiazania dopuszczalnego.

Operator wymiany podłańcucha STEX (substring exchange crossover) działa podobnie jak SEX z tym, że odcinek podlegający wymianie jest określany w standardowy dwu-punktowy sposób. Geny powtórzone są zastępowane brakującymi w sposób przypadkowy.

Operatory łączone z heurezami

Dobre wyniki numeryczne otrzymywane dla niektórych heurez zachęcają do tworzenia operatorów genetycznych, których jądrem jest heureza zaś wynikiem potomek otrzymany na bazie rozwiązań-rodziców.

Operator bazujący na schemacie GT GTX (GT-based cossover) jest uproszczoną wersją schematu GT. Wielowariantowy schemat gałęzienia został zastąpiony pojedynczym arbitralnym wyborem z elementami losowości i informacją genetyczną pochodzącą od rodziców. Dokładniej, w procesie rekombinacji wybierany jest, z jednakowym prawdopodobieństwem, jeden z rodziców. Rodzic ten określa priorytet dla wyboru operacji ze zbioru konfliktowego w tym sensie, że operacja z $j \in \mathcal{K}_k$ uszeregowana najwcześniej u wybranego rodzica jest preferowana. Odpowiednikiem mutacji jest przypadkowy wybór operacji ze zbioru $j \in \mathcal{K}_k$.

Mutacje przez lokalne poszukiwania MLS (neighbourd search-based mutation) stanowia pewne odstępstwo od "klasycznego" schematu GA. Opera-

tor mutacji występujący zwykle w tle otrzymuje pierwszoplanowe znaczenie w poszukiwaniach. Zmutowany potomek jest wybierany jako najlepszy na drodze przeglądu zbioru rozwiązań sąsiednich jednego rodzica. Możliwe są różne definicje sąsiedztwa, w tym z elementami losowości. Przykładowo, wybierany jest losowo ciąg genów rodzica, a następnie badane są wszystkie możliwe permutacje wzajemne lokalizacji tych genów.

Hybrydowe algorytmy GA

Szereg badań eksperymentalnych potwierdziło przewagę metod łączących kilka różnych, czasami krańcowo odmiennych podejść, nad metodami "czystymi". Hybrydyzacja GA polega na wbudowaniu dodatkowego algorytmu heurystycznego dla realizacji co najmniej jednej z następujących funkcji: (1) generacja początkowej, dobrze zaadaptowanej populacji, (2) ocena stopnia adaptacji osobnika, (3) wykonanie dodatkowego procesu poszukiwań "wspomagającego" podstawowy algorytm, zwykle w celu poprawy potomka przed zwróceniem go do cyklu obliczeniowego algorytmu GA. W tym ostatnim przypadku najczęściej używane są rożne metody poszukiwań lokalnych. Taki proces modyfikacji potomka zwykle nazywany jest "uczeniem" lub wstępną adaptacją cech do środowiska (wyjaśniana teorią ewolucji Lamarck'a). Wydaje się jednak, że bardziej właściwym uzasadnieniem jest świadoma modyfikacja genomu (metodami inżynierii genetycznej) w celu poprawy stopnia adaptacji osobnika do środowiska.

14.12 Podejście dualne

Model ten można uzyskać przez pewne rozszerzenie podejścia dualnego podanego w Rozdz. 10.5 dla przypadku jedno-maszynowego. W tym celu ograniczenia problemu wyrażone zostaja w formie

$$g_{tk}(S) \leqslant 1,\ k = 1, \ldots, m,\ t = 1, \ldots, T \tag{14.13}$$

$$S_i + p_i \leqslant S_{i+1},\ i = l_j + 1, \ldots, l_j - 1,\ j = 1, \ldots, n, \tag{14.14}$$

$$0 \leqslant S_i \leqslant T \tag{14.15}$$

gdzie

$$g_{tk}(S) = |\{i \in \mathcal{O} :\ \nu_i = k,\ t < S_j \leqslant t - p_j\}|\,. \tag{14.16}$$

zaś T jest horyzontem planowania lub jego górnym oszacowaniem. Warunek (14.13) jest ograniczeniem na skończoną (jednostkową) przepustowość

maszyn w każdym z elementarnych przedziałów czasu $[t-1,t]$. Dalsze postępowanie w modelowaniu zależy od postaci przyjętego kryterium. Opiszemy je oddzielnie dla kryterium $\sum \mathcal{C}_j$ oraz $\mathcal{C}_{\max}$.

Jeśli kryterium jest $\mathcal{C} \stackrel{\text{def}}{=} \sum \mathcal{C}_j$ to funkcja celu zależy od wektora S i ma postać

$$\mathcal{C}(S) = \sum_{j=1}^{n} (S_{l_j} + p_{l_j}) \tag{14.17}$$

Przy konstukcji funkcji Lagrange'a tylko ograniczeniom (14.13) przypiszemy ceny dualne u_{tk}, $t = 1,\ldots,T$, $k = 1,\ldots,m$, natomiast pozostałe ograniczenia uwzględnimy bezpośrednio poprzez zbiór terminów dopuszczalnych $\mathcal{S}$ spełniających ograniczenia (14.14)–(14.15). Niech $u = [u_{tk}]_{T\times m}$. Problem dualny ma postać

$$\max_{u\geqslant 0} W(u), \tag{14.18}$$

gdzie

$$W(u) = \min_{S\in\mathcal{S}} \sum_{j=1}^{n} L_j(S,u) - \sum_{t=1}^{T}\sum_{k=1}^{m} u_{tk} \tag{14.19}$$

oraz

$$L_j(S,u) = \sum_{i=l_{j-1}+1}^{l_j} \sum_{t=S_i+1}^{S_i+p_i} u_{t\nu_i} + S_{l_j} + p_{l_j}. \tag{14.20}$$

Każda funkcja $L_j(S,u)$ zależy tylko od składowych wektora S skojarzonych z zadaniem j, to znaczy od zmiennych S_i, $i = l_{j-1}+1,\ldots,l_j$. Zatem zależność (14.19), po kilku elementarnych przekształceniach, można przedstawić w postaci

$$W(u) = \sum_{j=1}^{n} V_j(u) - \sum_{t=1}^{T}\sum_{k=1}^{m} u_{tk} + \sum_{j=1}^{n} p_{l_j} \tag{14.21}$$

gdzie

$$V_j(u) = \min_{S\in\mathcal{S}} \left[\sum_{i=l_{j-1}+1}^{l_j} (U_{S_i+p_i,\nu_i} - U_{S_i,\nu_i}) + S_{l_j} \right]. \tag{14.22}$$

oraz

$$U_{tk} = \sum_{s=1}^{t} u_{sk}. \tag{14.23}$$

Wielkości U_{tk} mogą być obliczone rekurencyjnie z zależności $U_{tk} = U_{t-1,k} + u_{tk}$, $t = 1,\ldots,T$, $k = 1,\ldots,m$. Wyznaczenie minimum po prawej stronie

wzoru (14.22) wymaga pewnych zabiegów ze względu na zależność składowych S_i, $i = l_{j-1}+1, \ldots, l_j$. Oznaczmy minimalną wartość częściową sumy po prawej stronie (14.22) przez

$$Z_i(t,u) = \min_{S\in\mathcal{S};\ S_i\leqslant t} \sum_{s=l_{j-1}+1}^{i} (U_{S_s+p_s,\nu_s} - U_{S_s,\nu_s}) + X_i \tag{14.24}$$

gdzie

$$X_i = \begin{cases} 0 & \text{dla} \quad i \neq l_j \\ S_{l_j} & \text{dla} \quad i = l_j \end{cases} \tag{14.25}$$

Celem naszym jest wyznaczenie

$$V_j(u) = Z_{l_j}(T - p_{l_j}, u). \tag{14.26}$$

Wielkości $Z_i(t,u)$ można policzyć rekurencyjnie korzystając ze schematu PD

$$Z_i(t,u) = \min\{Z_i(t-1,u), Z_{i-1}(t-p_{i-1},u) + U_{t+p_i,\nu_i} - U_{t,\nu_i} + Y_i\} \tag{14.27}$$

gdzie

$$Y_i = \begin{cases} 0 & \text{dla} \quad i \neq l_j \\ t & \text{dla} \quad i = l_j \end{cases} \tag{14.28}$$

dla $t = r_i, \ldots, T - q_i$, przy czym

$$r_i = \sum_{s=l_{j-1}+1}^{i-1} p_s, \quad q_i = \sum_{s=i}^{l_j} p_s \tag{14.29}$$

oraz $i = l_{j-1}+1, \ldots, l_j$. Dla kompletności zakładamy, że $Z_i(t,u) = \infty$ dla $t < r_i$. Wyznaczenie wartości $W(u)$, dla ustalonego u, wymaga czasu rzędu $O(omT)$.

Jeśli funkcją celu jest $C_{\max}$ to należy wprowadzić fikcyjną operację $*$ o zerowym czasie trwania występującą po wszystkich zadaniach i związanych dodatkowymi ograniczeniami kolejnościowymi postaci

$$S_{l_j} + p_{l_j} \leqslant S_*, \ j = 1, \ldots, n, \tag{14.30}$$

a następnie jako wektor zmiennych przyjąć $S = (S_1, \ldots, S_o, S_*)$ zaś jako funkcję celu

$$C_{\max}(S) = S_*. \tag{14.31}$$

Zbiór $\mathcal{S}$ w tym przypadku jest określony przez warunki (14.14)–(14.15) oraz dodatkowo (14.30). Zmienia się także postać problemu dualnego oraz metoda

jego rozwiązania. Wzór (14.18) pozostaje bez zmian, zaś następne zależności przyjmują postać

$$W(u) = \min_{S\in\mathcal{S}}(\sum_{j=1}^{n} L_j(S,u) + L_*(S,u)) - \sum_{t=1}^{T}\sum_{k=1}^{m} u_{tk} \tag{14.32}$$

gdzie $V_j(u)$ jest określone wzorem (14.22) zaś

$$V_*(u) = \min_{S\in\mathcal{S}} S_*. \tag{14.33}$$

Wyznaczenie $V_j(u)$ $j = 1,\ldots,n$ można zrealizować za pomocą (14.26). Dla dodanej operacji wprowadzamy

$$Z_*(t,u) = \min_{S\in\mathcal{S};\, S_*\leqslant t} t \tag{14.34}$$

oraz odpowiedni wzór rekurencyjny

$$Z_*(t,u) = \min\{Z_*(t-1,u), t + \sum_{j=1}^{n} Z_{l_j}(t-p_{l_j},u)\} \tag{14.35}$$

liczony dla $t = r_*,\ldots,T$, gdzie $r_* = \max_{1\leqslant j\leqslant n}\sum_{s=l_{j-1}+1}^{l_j} p_s$. Zachodzi $W(u) = Z_*(T,u)$. Podobnie jak poprzednio, obliczenie $W(u)$ dla ustalonego u wymaga czasu rzędu $O(omT)$.

Ekstremalizację $\max_{u\geqslant 0} W(u)$ można zrealizować przy pomocy techniki subgradientowej stosowanej i opisanej szczegółowo w Rozdz. ??. Pozostaje nam jeszcze wskazanie metody aktualizacji górnego ograniczenia przy użyciu pewnej heurezy. Najprostszym rozwiązaniem jest zastosowanie jednościeżkowego schematu GT, w którym priorytet operacji jest ustalany w oparciu o rozwiązanie zadania dolnego poziomu.

14.13 Sieci neuronowe

Problemy optymalizacji kombinatorycznej można rozwiązywać sprzętowo, przy pomocy układów dynamicznych mających analogię do sieci neuronowych. Poniżej podamy przykład podejścia umożliwiający rozwiązanie problemu gniazdowego z kryterium minimalizacji sumy terminów zakończenia zadań. Punktem wyjściowym do analizy jest następujący problem z kryterium $\mathcal{C}$

$$\min_{S\in\mathcal{S}}\sum_{j=1}^{n}(S_{l_j} + p_{l_j}) = \min_{S\in\mathcal{S}}\sum_{j=1}^{n} S_{l_j} + \sum_{j=1}^{n} p_{l_j}, \tag{14.36}$$

przy ograniczeniach (**??**)-(**??**). Ponieważ człon drugi w (**??**) jest stały, w dalszym ciągu będziemy rozważać kryterium zawierające pierwszą cześć wyrażenia (**??**). Przyjmujemy równanie pojedynczego neuronu w postaci []

$$C\frac{dU_i}{dt} = -f_i(S) - \frac{U_i}{R}, \tag{14.37}$$

gdzie $S_i = g(U_i)$, $g(x)$ jest funkcją liniową rosnącą, $f_i(S)$ jest wejściem do neuronu i z sieci sprzężenia zwrotnego, C oraz R są pojemnością i opornością węzła wejściowego. Odpowiednio, funkcja energii układu ma postać

$$E = E_f + E_U = \sum_{i=1}^{o} \int_0^{S_i} f_i(s)ds + \sum_{i=1}^{o} \int_0^{S_i} \frac{U_i(s)}{R} ds. \tag{14.38}$$

Celem naszym jest zaprojektowanie układu (dokładniej szczególnych postaci funkcji $f_i(x),\ i \in \mathcal{O}$), który przy danych warunkach początkowych zmierzałby samorzutnie do stanu o minimalnej energii posiadającej relację do funkcji kryterialnej oraz ograniczeń problemu. W pracy [393] zaproponowano E_f w postaci

$$E_f = \sum_{j=1}^{n} S_{l_j} + u \sum_{j=1}^{n} g(-S_{l_{j-1}+1}) + v \sum_{j=1}^{n} \sum_{i=l_{j-1}+1}^{l_j-1} g(S_i + p_i - S_{i+1})$$

$$+w \sum_{i,j\in\mathcal{O};\ i\neq j;\ \nu_i=\nu_j} h(S_i + p_i - S_j)\ h(S_j + p_j - S_i), \tag{14.39}$$

gdzie $u, v, w > 0$ są pewnymi dużymi stałymi, zaś $g(x) = e^{bx} - bx$, $h(x) = e^{bx} - 1$ dla $x \geqslant 0$ (b jest stałą wynikającą z charakterystyki diody) oraz $g(x) = 0 = h(x)$ dla $x < 0$. Łatwo zauważyć, że pierwszy człon w E_f odpowiada za wartość funkcji kryterialnej (**??**), zaś pozostałe człony reprezentują kary za przekroczenie ograniczeń. Kary te osiągają wartość zerową, gdy ograniczenia są spełnione oraz dodatnią, gdy są naruszone. Odpowiednie postacie funkcji $f_i(S)$ można otrzymać licząc pochodne (**??**) względem S_i. Startując od dowolnych warunków początkowych energia układu maleje, bowiem

$$\frac{dE}{dt} = \sum_{i=1}^{o} (f_j(S) + \frac{U_j}{R}) \cdot \frac{dS_j}{dt} = -\sum_{i=1}^{o} C\frac{dU_j}{dt} \cdot \frac{dS_j}{dt}$$

$$= -\sum_{i=1}^{o} C\ [g^{-1}(S_j)]' \cdot (\frac{dS_j}{dt})^2 \leqslant 0. \tag{14.40}$$

Ostatnia nierówność zachodzi, jeśli tylko $g(x)$ jest monotoniczną funkcją rosnącą, co ma miejsce w omawianym podejściu.

Jak to tej pory nie potwierdzono konkurencyjności tego podejścia względem innych znanych. Niedostatkami metody są problemy skalowania oraz przedwczesna zbieżność do lokalnego optimum pociągająca potrzebę wielokrotnego uruchomienia przebiegu z różnych warunków początkowych.

14.14 Uwagi

Podobnie jak w poprzednich zagadnieniach , utworzona została biblioteka testów numerycznych zawierająca szczególnie trudne przykłady o rozmiarze od 100 do 2,000 operacji (100 zadań, 20 maszyn) [358], dostępna także w Internecie [31,275,355]. Szczególna trudność problemu gniazdowego oraz jego znaczenie praktyczne powodują, że wszystkie znane i nowo powstające algorytmy na świecie są testowane w pierwszej kolejności na tych przykładach. Śledząc rozwój teorii oraz historię powstających algorytmów można stwierdzić, że obecnie potrafimy rozwiązywać z dużą dokładnością przykłady praktyczne problemu gniazdowego z kryterium $C_{\max}$, lub ogólnie $f_{\max}$, o rozmiarach rzędu 2,000 operacji, w czasie minut na PC, stosując najkorzystniejszą aktualnie technikę TS. Fakt ten stanowi o wystarczajaco dobrej praktycznej przydatności odpowiednich algorytmów szeregowania. Uwzględniając moc obliczeniową współczesnych PC oraz możliwości prowadzenia równoległych poszukiwań, daje to realną szansę rozwiązywania przykładów o rozmiarze 5,000–10,000 operacji. Biorąc pod uwagę dane statystyczne podane na wstępie, każdy procent poprawy jakości rozwiązania może dostarczyć znaczących korzyści.

Sukcesy osiągane dla podstawowego problemu gniazdowego skłaniają wielu badaczy do formułowania i analizowania bardziej wyrafinowanych problemów, na przykład uwzględniających czasy transportu, przezbrojenia, terminy gotowości, wielowariantową obsługę, itp., w celu lepszego przybliżenia złożonych systemów wytwarzania. Odpowiednie algorytmy dla tych zagadnień projektowane są zwykle przez analogię, wykorzystując doświadczenia uzyskane dla klasycznych problemów prezentowanych w monografii.

Odmiennie, algorytmy dla problemów gniazdowych z kryterium addytywnym, mimo iż były formułowane w literaturze, ciągle znajdują się jeszcze w fazie rozwoju, głównie ze względu na brak teoretycznych własności szczególnych, wystarczająco korzystnych dla projektowanych algorytmów. Niezaleznie od powyższego, istniej kilka podejść (techniki priorytetowe, metoda dualna, sieci neuronowe) które mogą być rutynowo stosowane do problemów tej klasy.

14.15 Warsztat

1. Wyznacz funkcje $f_j(x)$ dla problemu (**??**)–(**??**).
2. Zasymuluj układ sprzętowy dla przykładu problemu z ???
3. Zaprojektuj sprzętowy układ dynamiczny do rozwiązywania problemu gniazdowego z kryterium C_{max}
4. Zaprojektuj sprzętowy układ dynamiczny do rozwiązywania problemu gniazdowego z kryterium średniego czasu przepływu przy danych niezerowych terminach gotowości zadań.

Bibliografia

[1] Aarts E.H.L., van Laarhoven P.J.M., Simulated annealing: a pedestrian review of the theory and some applications. In: Pattern Recognition and Applications, Eds. Deviijver P.A. and Kittler J., Springer, 1987, Berlin.

[2] Aarts E.H.I., Van Laarhoven P.J.M., Lenstra J.K., Ulder N.I.J., A computational study of local search algorithms for job shop scheduling, *ORSA Journal on Computing*, 1994, 6, 118-125.

[3] Aarts E.H.I., Lenstra J.K., Local search in Combinatorial Optimization, John Wiley and Sons Ltd, Chichester, England, 1997.

[4] Achuthan N.R., Grabowski J., Sidney J.B., Optimal flow-shop scheduling with earliness and tardiness penalties, *OPSEARCH*, 1981, 18, 117–138.

[5] Adams, J., Balas E., Zawack D., The Shifting Bottleneck Procedure for Job Shop Scheduling, *Management Science*, 1988, 34, 391–401.

[6] Adamopolous G.I., Pappis C.P, Scheduling jobs with different, job-dependent earliness and tardiness penalties using SLK methods, *European Journal of Operational Research*, 1996, 88, 336–344.

[7] Adenso-Diaz B.A., Restricted neighborhood in the tabu search for the flow shop problem, *European Journal of Operational Research*, 1992,62, 27–37

[8] Aderohunmu R., Mobolurin A., Bryson N., Joint Vendor Buyer Policy in Jit Manufacturing, *Journal of Operational Research Society*, 1995, 46, 375–385.

[9] Adleman L.M., Molecular Computation of Solutions to Combinatorial Problems, *SCIENCE*, 1994, 266, 1021–1024

[10] Agrawal A., Harhalakis G., Minis I., Nagi R., Just-in-Time Production of Large Assemblies, *IIE Transactions*, 1996, 28, 653–667.

[11] Aho A.V., Hopcroft J.E., Ullman J.D., The design and analysis of computer algorithms, Addison-Wesley Publishing Company, 1974.

[12] Al-Turki U.M., Mittenthal J., Raghavachari M., A dominant subset of V-shaped sequences for a class of single-machine sequencing problems, *European Journal of Operational Research*, 1996, 88, 345–347.

[13] Amin S., Using Adaptive Temperature Control for Solving Optimization Problems, *Baltzer Journals*, 1996, (in print)

[14] Ansari A., Heckle J., JIT purchasing: Impact of freight and inventory costs, *Journal of Purchasing Matl. Management*, 1987, 23, 24–28.

[15] Applegate D., Cook W., A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing*, 1991, 3, 149–156.

[16] Arthnari T.S., Mukhopadhyay A.A., A note on a paper by W. Szwarc, *Naval Research Logistics Quartely*, 1971, 18, 135–138

[17] Ashayeri J., Gelders L.F., Warehouse design optimization, *European Journal of Operational Research*, 1985, 21, 285–294.

[18] Ashour S., A branch and bound algorithm for flow shop scheduling problem, *AIIE Transactions* 2, 1970, 172–176.

[19] Askin R.G., Mitwasi M.G., Goldberg J.B., Determining the Number of Kanbans in Multiitem Just-in-Time Systems, *IIE Transactions*, 1993, 25, 89–98.

[20] Bagchi U., Sullivan R., Chang Y., Minimizing Mean Squared Deviation of Completion Times About a Common Due Date, *Management Science*, 1987, 33, 894–906.

[21] Baker K.R., Introduction to Sequencing and Scheduling, Wiley, New York, 1974.

[22] Baker K.R., A dynamic priority rule for scheduling against due dates, TIMES/ORSA Conference, Houston, 1981.

[23] Baker K.R., Schrage L., Finding an Optimal Sequence by Dynamic Programming: an Extension to Precedence-Related Tasks, *Operations Research*, 1978, 26, 111–120.

[24] Baker K.R., Martin J.B., An experimental comparison of solution algorithms for the single-machine tardiness problem, *Naval Research Logistics Quartely*, 1974, 21, 187–199.

[25] Baker K.R., Su Z.S., Sequencing with due dates and early start times to minimize tardiness, *Naval Research Logistics Quartely*, 1974, 21, 171–176.

[26] Baker K.R., Scudder G.D., Sequencing with earliness and tardiness penalties: A review, *Operations Research*, 1990,30, 22–36.

[27] Baker K.R., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints, Report 8028, Erasmus University of Rotterdam, The Netherlands.

[28] Balas E., Machine sequencing via disjunctive graphs. An implicite enumeration algorithm. *Operations Research*, 1969, 17, 941-957.

[29] Balas E., Vazacopoulous A., Guided Local Search with Shifting Bottleneck for Job Shop Scheduling, Management Science Technical Report MSRR-609(R), Carnegie Mellon University, Pittsburgh, PA 15213, 1995.

[30] Battiti R., Tecchiolli G., The reactive tabu search, *ORSA Journal on Computing*, 1994, 6, 126–140.

[31] Beasley J.E., OR Library: Distributing Test Problems by Electronic Mail. *Journal of Operational Research Society*, 1990, 41, 1069–1072

[32] Belov I.S., Stolin Y.N., An algorithm in a single path operations scheduling problem, *Mathematical Economics and Functional Analysis (Russian)*, 1974, Nauka, Moscow, 248–257.

[33] Birewar D.B., Grossmann I.E., Incorporating scheduling in the optimal design of multiproduct batch plants, *Computers and Chemical Engineering*, 1989, 13, 141–161.

[34] Bitran G.R., Chang L., A mathematical programming approach to a deterministic Kanban system. *Management Science*, 1987, 33, 427–441.

[35] Błażewicz J., Złożoność obliczeniowa problemów kombinatorycznych, Warszawa, WNT, 1988.

[36] Błażewicz J., Domschke W., Pesh E., The job shop scheduling problem: Conventional and new solution techniques, *European Journal of Operational Research*, 1996, 93, 1-33.

[37] Błażewicz J., Ecker K., Schmidt G., Węglarz J., Scheduling in Computer and Manufacturing Systems, Springer-Verlag, 1993.

[38] Błażewicz J., Ecker K.H., Schmidt G., Węglarz J., Scheduling computer and manufacturing processes, Springer Verlag, Berlin, New York, 1996.

[39] Brah S.A, Scheduling in a flow shop with multiple processors, Ph.D. Dissertation, University of Houston, TX, 1988.

[40] Bratley P., Florian M., Robillard P., On sequencing with earliest starts and due dates with application to computing bounds for (n/m/G/F) problem, *Naval Research Logistics Quartely*, 1973, 20, 57–67.

[41] Brown A.P.G., Łomnicki Z.S., Some Application of the Branch and Bound Algorithm for the Machine Scheduling Problem, *Operations Research Quarterly*, 1966, 17, 17–189.

[42] Brucker P., Jurish B., Sieviers B., A Fast Branch&Bound Algorithm for the Job-Shop Problem, *Discrete Applied Mathematics*, 1994, 49, 107–127.

[43] Brucker P., Scheduling Algorithms, Springer-Verlag Berlin-Heidelberg, 1995.

[44] Brucker P., Neyer J., Internet: http://www.mathematik.uni-osnabrueck.de/ research/OR/software.html.

[45] Buffa E.S., Taubert W.L., Production-Inventory Systems: Planning and Control. Richard D. Irwin, Homewood, 1972.

[46] Buzacott J.A., Automatic transfer lines with buffer stocks, *International Journal of Production Research*, 1967, 5, 183–200.

[47] Cai X., V-shape property for job sequences that minimize the expected completion time variance, *European Journal of Operational Research*, 1996, 91, 118–123.

[48] Campbell H.G., Dudek R.A., Smith M.L., A heuristic algorithm for the n job m machine sequencing problem. *Management Science*, 1970, 16, B630–637.

[49] Carlier J., Scheduling jobs with release dates and tails on identical machines to minimize makespan, *European Journal of Operational Research*, 1987, 29, 298–306.

[50] Carlier J., Pinson E., Jackson's pseudo-preemptive schedule for the $Pm|r_i, q_i|C_{max}$ scheduling problem, Technical Report, Universite de Technologie de Compiegne, HEUDIASYC, 1996.

[51] Carlier J., The one-machine sequencing problem,*European Journal of Operational Research*, 1982, 11, 42-47.

[52] Carlier J., Pinson E., An Algorithm for Solving the Job-Shop problem, *Management Science*, 1989, 35, 164–176.

[53] Chapman S.N., Just-in-time supplier inventory and empirical implemmentation model, *International Journal of Production Research*, 1989, 27, 1993–2007.

[54] Chaudhury A., Whinston A.B., Towards an adaptive Kanban system, *International Journal of Production Research*, 1990, 28, 437–458.

[55] Chen H.G., Operator Scheduling Approaches in Group Technology Cells – Information Request Analysis, *IEEE Transactions on Systems, Man, and Cybernetics*, 1995, 25, 438–452.

[56] Chen B., Analysis of Classes of Heuristics for Scheduling a Two-Stage Flow Shop with Parallel Machines at One Stage, *Journal of Operational Research Society*, 1995, 46, 234–244.

[57] Chen C.L., Vempati V.S., Aljaber N., An application of genetic algorithm for flowshop problems, *European Journal of Operational Research*, 1995, 80, 389–396.

[58] Cheng T., Optimal Common Due Date With Limited Completion Time Deviation, *Computer and Operations Research*, 1988, 15, 91–96.

[59] Cheng R., Ge M., Tsujimura Y., A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. *Computers Industrial Engineering*36, 1999, 343-364.

[60] Cheng T.C.E. , Gupta M.C., Survey of scheduling research involving due date detemination decisions, *European Journal of Operational Research*, 1989, 38, 156–166

[61] Cheng C.C., Smith S.F., Applying Constraint Satisfaction Techniques to Job Shop Scheduling, Technical Report CMU-RI-TR-95-03, Carnegie Mellon University, Pittsburgh, 1995.

[62] Cheng L.P. , Ding F.Y., Modifying Mixed-Model Assembly-Line Sequencing Methods to Consider Weighted Variations for Just-in-Time Production Systems, *IIE Transactions*, 1996, 28, 919–927.

[63] Chlebus E., Techniki komputerowe CAx w inżynierii produkcji, WNT, Warszawa, 2000.

[64] Choi J.W., Investment in the Reduction of Uncertainties in Just-in-Time Purchasing Systems, *Naval Research Logistics*, 1994, 41, 257–272.

[65] Chretienne P., Coffman Jr. E.G., Lenstra J.K. and Liu Z., Scheduling theory and its applications. Wiley, Chichester 1995.

[66] Co H.C. , Jacobson S.H., The Kanban Assignment Problem in Serial Just-in-Time Production Systems, *IIE Transactions*, 1994, 26, 76–85.

[67] Congram R.K., Potts C.N., van de Velde S.L., An Iterative Dynasearch Algorithm for the single-machine Total Weighted Tardiness Scheduling Problems, Technical Report, 1998

[68] Conwey R.W., Maxwell W.L., Miller L.W., Theory of Scheduling, Addison-Wesley, 1967.

[69] Cormen T.H., Leiserson C.E., Rivest R.L., Wprowadzenie do algorytmów, WNT 1997.

[70] Corne D., Dorigo M., Gover F. (eds), New ideas in optimization, McGraw-Hill, London, 1999.

[71] Crawford K.M., Blackstone J.H.Jr, Cox J.F., A study of JIT implementation and operating problems, *International Journal of Production Research*, 1988, 26, 1561–1568.

[72] Dannenbring D.G., An evaluation of flow-shop sequencing heuristics. *Management Science*, 1977, 23, 1174–1182.

[73] Das H., Cummings P.T., Le Van M.D., Scheduling of serial multiproduct batch processes via simulated annealing, *Computers and Chemical Engineering*, 1990, 14, 1351–1362.

[74] Davidor Y., A Naturally Occuring Nice & Species Phenomenon: The model and First Results, *Proc. of 4'th Int. Conf. on Genetic Alg.*, 1991, Morgan Kaufmann Publishers, 257–263.

[75] Deal D.E., Yang T., Hallquist S., Job scheduling in petrochemical production: two-stage processing with finite intermediate storage, *Computers and Chemical Engineering*, 1994, 18, 33–344.

[76] Del'Amico M., Trubian M., Applying Tabu Search to the Job-Shop Scheduling Problem, *Annals of Operations Research*, 1993, 41, 213–252.

[77] Deleersnyder J.L., Hodgson T.J., Muller H.M., O'Grady P.J., Kanban controlled pull systems: An analytic approach, *Management Science*, 1989, 35, 1079–1091.

[78] Della Croce F., Narayan V., Tadei R., The two-machine total completion time flow shop problem, *European Journal of Operational Research*, 1996, 90, 227–237.

[79] Diaconis P., Group Representations in Probability and Statistics. *Lecture Notes - Monograph Series* 11, Institute of Mathematical Statistics, Harvard University, 1988.

[80] Diaz B.A., Restricted neighborhood in the tabu search for the flowshop problem, *European Journal of Operational Research*, 1992, 62, 27–37.

[81] Ding F.Y., Cheng L.P., A Simple Sequencing Algorithm for Mixed-Model Assembly Lines in Just-in-Time Production Systems, *Operations Research Letters*, 1993, 13, 27–36.

[82] Ding F.Y., Kittichartphayak D., Heuristics for scheduling flexible flow lines, *Computers Industrial Engineering*, 1994, 26, 27–34.

[83] Dorigo M., Maniezzo V., Colorni A., Ant System: Optimization by a Colony of Cooperating Agents, *IEEE Transactions on Systems, Man, and Cybernetics. Part B: Cybernetics*, 1996, 26, 29–41.

[84] Drexl A, Kimms A, Beyond Maufacturing Resource Planning (MRP II). Advanced Models and Methods for Production Planning, Springer-Verlag, Berlin-Heideberg, 1998.

[85] Dutta S.K., Cunningham A.A., Sequencing two-machine flow-shops with finite intermediate storage, *Management Science*, 1975, 21, 989–996.

[86] Dauzere-Peres J., Lasserre B., Modified shifting bottleneck procedure for job shop scheduling, *International Journal of Production Research*, 1993, 31, 923–932.

[87] Eck B., Pinedo M., Good Solution to Job Scheduling Problems Via Tabu Search. Presented at Joint ORSA/TIMS Meeting, Vancouver 1989.

[88] Edosomwan J.A., Marsh C., Streamlining the material flow process for just-ini-time production, *Industrial Engineering*, 1989, 21, 46–50.

[89] El-Rayal T.E., Hollier R.H., A review of plant design techniques, *International Journal of Production Research*, 1970, 8, 263–279.

[90] Ernst R., Pyke D.F., Optimal Base Stock Policies and Truck Capacity in a 2-Echelon System, *Naval Research Logistics*, 1993, 40, 879–903.

[91] Eshelman L.J., Schaffer J.D., Preventing Premature Convergence in Genetic Algorithms by Preventing Incest, *Proc. of 4'th Int. Conf. of Genetic Alg.*, 1991, Morgan Kaufmann Publishers, 115–122.

[92] Fial T., Közelitö algorithmus a haróm gép problémára, *Alkalmazott Matematikai Logopok*, 1977, 3, 389–398.

[93] Fisher M.L., A dual algorithm for one-machine scheduling problem, *Mathematical Programming*, 1976, 11, 229–251.

[94] Fisher H., Thompson G.L., Probabilistic learning combinations of local job shop scheduling rules. in: Industrial Scheduling, Muth J.F. and Thompson G.L., eds., Prentice Hall, Englewood Cliffs, 1963, 225-251.

[95] Flapper S.D.P.,Miltenburg J., Wijngaard J., Embedding JIT into MRP. *International Journal of Production Research*, 1991, 29, 329–341.

[96] Foster G., Horngren C.T., Costs accounting and cost management in a JIT environment, *J. Cost. Mgmt.*, 1988, 4–14

[97] French S., Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop. Horwood, Chichester, UK, 1982.

[98] Funk J.L., A comparison of inventory cost reduction strategies in a JIT manufacturing system. *International Journal of Production Research*, 1989, 27, 1065–1080.

[99] Garey M.R., Johnson D.S., Sethi R., The Complexity of Flow Shop and Job Shop Schedules, *Mathematics of Operations Research*, 1976, 1, 117–129.

[100] Garey M.R., Johnson D.S., Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., New York, 1979.

[101] Garey M.R., Tarjan R.E., Wilfong G.T., One-Processor Scheduling with Symmetric Earliness and Tardiness Penalties, *Mathematics of Operational Research*, 1988, 13, 330–348.

[102] Garfinkel R.S., Nemhauser G.L., Programowanie całkowitoliczbowe, PWN, Warszawa, 1978.

[103] Gelders L.F., Sambandam N., Four simple heuristics for a flow-shop, *International Journal of Production Research*, 1978, 16, 221–231.

[104] Geofrion A. M., Marsten R. E., Integer programming algorithms: a Framwork and state-of-the art survey, Management Sci., vol. 9, 1972, 132-153.

[105] Georges-Schleuter R., Comparison of Local Mating Strategies in Massively Parallel Genetic Algorithms, PPSN, North-Holland, 1992, 2, 553–562.

[106] Gilbert J.P., The state of JIT implementation in the U.S.A., *International Journal of Production Research*, 1990, 28, 1099–1109.

[107] Gilmore P.C., Gomory R.E., Sequencing a one state variable machine: a Solvable Case of the Traveling Salesman Problem, *Operations Research*, 1964, 12, 655–679.

[108] Glauber R.J., Time-dependent statistics of the Ishing model, *Journal of Mathematical Phisics*, 1963, 4, 294–299.

[109] Globerson S., Millen R., Determining learning curves in group technology settings, *International Journal of Production Research*, 1989, 27, 1653–1664.

[110] Glover F., Laguna M., Tabu Search, Kluwer Academic Publishers, Massachusetts USA, 1997.

[111] Glover F., Taillard E. de Werra D., A Users Guide to Tabu Search, *Annals of Operations Research*, 1993, 41, 3–28.

[112] Goldberg D.E., Algorytmy genetyczne i ich zastosowanie, WNT 1995.

[113] Goldberg A.V., Tarjan R.E., A new approach to the maximum flow problem, *Journal of ACM*, 1988, 35, 921–940.

[114] Golden B., Assad A., Perspectives of the vehicle routing: Exciting new developments. *Operations Research*, 1986, 34, 803–810.

[115] Golhar D.Y., Stamm C.L., The just-in-time philosophy: A literature review. *International Journal of Production Research*, 1991, 29, 657–676.

[116] Georges-Schleuter M., Comparison of Local Matting Strategies in Massively Parallel Genetic Algorithms, PPSN, 1992, 2, North-Holland, 553–562.

[117] Gonzales T., Ibarra O.H., Sahni S., Bounds for LPT schedules on uniform processors, *SIAM Journal on Computing*, 1977, 6, 155–166.

[118] Gonzales T., Sahni S., Flowshop and Jobshop Schedules: Complexity and Approximation, *Operations Research*, 1978, 26, 36–52.

[119] Goyal S.K., Gunasekaran A., Multistage production-inventory systems. *European Journal of Operational Research*, 1990, 46, 1–20.

[120] Grabowski J., Skubalska E., Smutnicki C., On flow shop scheduling with release and due dates to minimize maximum lateness. *Journal of Operational Research Society*, 1983, 34, 615–620.

[121] Grabowski J., Smutnicki C., Problem kolejnościowy taśmowy. Własności, algorytmy i zastosowania. Część I. Podstawowe własoności. *Przegląd Statystyczny*, 1983, XXX, z.3/4, 275–288.

[122] Grabowski J., Smutnicki C., Problem kolejnościowy taśmowy. Własności, algorytmy i zastosowania. Część II. Dolne ograniczenia. *Przegląd Statystyczny*, 1984, XXXI, z.1/2, 31–45.

[123] Grabowski J., Smutnicki C., Minimalizacja maksymalnego kosztu w zagadnieniu kolejnościowym taśmowym. Część I. Podstawowe własności i algorytmy. *Archiwum Automatyki i Telemechaniki*, 1984, XXIX, z.1–2, 36–55.

[124] Grabowski J., Smutnicki C., Minimalizacja maksymalnego kosztu w zagadnieniu kolejnościowym taśmowym. Część II. Dolne ograniczenia, wyniki obliczeń i zastosowania. *Archiwum Automatyki i Telemechaniki*, 1984, XXIX, z. 1–2, 57–74.

[125] Grabowski J., Nowicki E., Smutnicki C., Zdrzałka S., Biblioteka procedur optymalizacji dla problemów szeregowania. Część I. Klasyfikacja, własności i algorytmy, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1984, 74, 77–85.

[126] Grabowski J., Nowicki E., Smutnicki C., Zdrzałka S., Biblioteka procedur optymalizacji dla problemów szeregowania. Część II. Struktura

biblioteki. Techniki obliczeniowe. Efektywność, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1984, 74.

[127] Grabowski J., Nowicki E., Smutnicki C., Nowe metody obliczania dolnego ograniczenia dla problemu kolejnościowego gniazdowego, *Zeszyty Naukowe AGH, Ser. Automatyka*, 1985, 39, 171–175.

[128] Grabowski J., Smutnicki C., Algorymy przybliżone dla problemu kolejnościowego taśmowego, *Zeszyty Naukowe AGH, Ser. Automatyka*, 1985, 39, 177–184.

[129] Grabowski J., E. Nowicki, Zdrzalka S., A Block Approach for Single Machine Scheduling with Release Dates and Due Dates. *European Journal of Operational Research*, 1986, 26, 278–285.

[130] Grabowski J., Nowicki E., Smutnicki C., Podejście blokowe do problemu kolejnościowego gniazdowego, *Zeszyty Naukowe Politechniki Śląskiej, Ser. Automatyka*, 1986, 84, 73–80.

[131] Grabowski J., Smutnicki C., Problemy szeregowania z minimaksową funkcją kary. *Archiwum Automatyki i Telemechaniki*, 1986, XXXI, z.1–2, 21–37.

[132] Grabowski J., Nowicki E., Smutnicki C., Algorytm blokowy szeregowania operacji w systemie gniazdowym, *Przegląd Statystyczny*, 1988, 35, z. 1/2, 67–80.

[133] Grabowski J., Nowicki E., Smutnicki C., Minimalizacja maksymalnego kosztu w gniazdowych zagadnieniach kolejnościowych, *Archiwum Automatyki i Telemechaniki*, 1988, 33, z.3, 389–402.

[134] Grabowski J., Smutnicki C., A Block Approach for Flow-Shop Scheduling to Minimize Maximum Cost, *Proc. of XIII Conference Cybernetics and Systems '96*, 1996, 826–831.

[135] Grabowski J, Pempera J., New block properties for the flow shop problem with application in TS, *Journal of Operational Research Society*, 2001, 52, 210–220.

[136] Grabowski J, Pempera J., Smutnicki C., Scheduling in production of concrete wares, *Proc. of Symposium of Opeations Research SOR'96, Braunschewig, Germany*, (in print).

[137] Graells M., Espuña A., Puigjaner L., Sequencing intermediate products: A practical solution for multipurpose production scheduling, *Computers and Chemical Engineering*, 1996, 20, 1137–1142.

[138] Graham R.L., Bounds on multiprocessing timing anomalies, *SIAM Journal on Applied Mathematics*, 1969, 17, 263–269.

[139] Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics*, 1979, 5, 287–326.

[140] Grau R., Espuña A., Puigjaner L., Completion times in multipurpose batch plants with set-up, transfer and clean-up times, *Computers and Chemical Engineering*, 1996, 20, 1143–1148.

[141] Gravel M., Price W.L., Using Kanban in a job-shop environment, *International Journal of Production Research*, 1988, 26, 1105–1118.

[142] Groeflin H., Luss H., Rosenwein M.B., Wahls E.T., Final assembly sequencing for just-in-time manufacturing, *International Journal of Production Research*, 1989, 27, 199–213.

[143] Grout J.R. , Christy D.P., An Inventory Model of Incentives for on-Time Delivery in Just-in-Time Purchasing Contracts, *Naval Research Logistics*, 1993, 40, 863–877.

[144] Gunaskekaran A., Goyal S.K., Martikainen T. , Yli-Olli P., Modeling and analysis of Just-In-Time manufacturing systems, *International Journal of Production Economics*, 1993, 32, 23–37.

[145] Gunasekaran A., Goyal S.K., Martikainen T., Yliolli P., Equipment Selection-Problems in Just-in-Time Manufacturing Systems, *Journal of Operational Research Society*, 1993, 44, 345–353.

[146] Gupta J.N.D., A functional heuristic algorithm for the flow-shop scheduling problem. *Operations Research Quarterly*, 1971, 22, 39–47.

[147] Gupta Y.P., A feasibility study of JIT-purchasing systems implementation in a manufacturing facility, *International Journal on Operational Production Management*, 1987, 10, 31–41.

[148] Gupta J.N.D., Comparative Evaluation of Heuristic Algorithms for the Single Machine Scheduling Problem with Two Operations per Job and Time-Lags, *Journal of Global Optimization*, 1996, 9, 239–250.

[149] Gupta J.N.D., Flow shop scheduling via sorting analogy; multisorting algorithms, Proc. ORSA/TIMS Joint Meeting, San Juan, Puerto Rico, Oct 1974, 16–18.

[150] Gupta Y.P., Bagchi P., Inbound freight consolidation under just-in-time procurements: Application of clearing models, *J.Bus.Logist.*, 1987, 8, 74–94.

[151] Gupta Y.P., Gupta M.C., A system dynamics model for a muti-stage multi-line dual-card JIT-kanban system, *International Journal of Production Research*, 1989, 27, 309–352.

[152] Gupta S.K., Kyparisis J., Single machine scheduling research, *OMEGA International Journal of Management Science*15, 1987, 207–227.

[153] Hahm J., Yano C.A., The Economic Lot and Delivery Scheduling Problem - The Common Cycle Case, *IIE Transactions*, 1995, 27, 113–125.

[154] Hahn C.K., Pinto P.A., Bragg D.J., Just-in-time production and purchasing, *Journal of Purchasing Matl. Management*, 1983, 19, 2–10.

[155] Hall R.W., Zero inventories. Down–Jones–Irwin, Homewood, 1983.

[156] Hall R.W., Attaining Manufacturing Excellence. Down–Jones–Irwin, Homewood, 1987.

[157] Hall N.G., Kubiak W., Sethi S.P., Earliness–Tardiness Scheduling Problems, II: Deviation of Completion Times About a Restrictive Common Due Date, *Operations Research*, 1991, 39, 847–856.

[158] Hall N.G., Posner M.E., Earliness-Tardiness Scheduling Problems, I: Weighted Deviation of Completion Times About a Common Due Date, *Operations Research*, 1991, 39, 836–846.

[159] Hall L.A., Shmoys D.B., Jackson's rule for single-machine scheduling: Making a good heuristic better, Technical Report, 1990.

[160] Hall G.N., Sriskandarajah C., A survey of machine scheduling problems with blocking and no-wait in process, *Operations Research*44, 1996, 510-524.

[161] Hay E.J., The Just-In-Time Breakthrough. Wiley, New York, 1988.

[162] Herring B., Simulation helps tire manufacturer change from push to pull system in controling material flow, *Industrial Engineering*, 1989, 21, 38–40.

[163] Herrmann J.W., Lee C.Y., On scheduling to minimize earliness-tardiness and batch delivery costs with a common due date, *European Journal of Operational Research*, 1993, 70, 272–288.

[164] Ho J.C., Chang Y.L., A new heuristic for the n-job, m-machine flowshop problem. *European Journal of Operational Research*, 1990, 52, 194–206.

[165] Ho J.C., Flowshop sequencing with mean flowtime objective, *European Journal of Operational Research*, 1995, 81, 571–578.

[166] Hodgson T.J., Wang D., Optimal hybrid push/pull control strategies for a parallel multistage system. Part I. *International Journal of Production Research*, 1991, 29, 1279–1287.

[167] Hoogeveen H., Hurkens C., Lenstra J.K., Vandervelde A., Lower bounds for the multiprocessor flow shop, Presented at 2nd Workshop on Models and Algorithms for Planning and Scheduling, Wernigerode, May 1995.

[168] Hoogeveen J.A., Kawaguchi T., Minimizing total completion time in a two-machine flowshop: analysis of special cases, *Working paper*, Eindhoven University of Technology, 1995.

[169] Horn W.A., *Single-machine job sequencing with treelike precedence ordering and linear delay penalties*, SIAM Journal on Applied Mathematics, vol. 23, 1972, 189-202.

[170] Hu T. C., *Parallel sequencing and assembly line problems*, Opns. Res., vol. 9, 1961, 841-848.

[171] Huang P.V., Rees L.P., Taylor III B.W., A simulation analysis of the Japaneese Just-in-time technique (with Kanbans) for a multi-line, multi-stage production system. *Decision Science*, 1983, 14, 326–344.

[172] Hubscher R., Glover F., Applying Tabu Search with Influential Diversification to Multiprocessor Scheduling. Technical Report, University of Colorado at Boulder, 1992.

[173] Hundal T.S., Rajgopal J., An extension of Palmers's heuristic for the flow-shop scheduling problem. *International Journal of Production Economics*, 1988, 26, 1119–1124.

[174] Hunsucker J.L., Shah J.R., Comparative performance analysis of priority rules in a constrained flow shop with multiple processors environment. *European Journal of Operational Research*, 1994, 72, 102–114.

[175] Hurink J, Internet: ftp://ftp.mathematik. Uni-Osnabrueck.DE/pub osm/preprints/software/openshop.

[176] Ignall E., Schrage L., Application of the branch and bound technique to some flow shop scheduling problem, *Operations Research*, 1956, 13, 400-402.

[177] Ignall E., Silver E.A., The output of a two stage system with unreliable machines and limited storage, *Industrial Engineering*, 1987, 19, 50–55.

[178] Im J.H., Lee S.M., Implementation of just-in-time systems in US manufacturing firms, *International Journal on Operational Production Management*, 1989, 9, 5–14.

[179] Inman R.R., Bulfin R.L., Sequencing JIT mixed-model assembly lines, *Management Science*, 1991, 37, 901–904.

[180] Ishibuchi H., Misaki S., Tanaka H., Modified simulated annealing algorithms for the flowshop sequencing problems, *European Journal of Operational Research*, 1995, 81, 388–398.

[181] Jackson J.R., Scheduling a production line to minimize maximum tardiness, Research Report, Management Science Research Project, University of California, Los Angeles, 1955.

[182] Jackson J.R., An extension of Johnson's results on job lot scheduling, *Naval Research Logistics Quartely*, 1956, 3, 201–203.

[183] Janiak A., Wybrane problemy i algorytmy szeregowania zadań i rozdziału zasobów, Akademicka Oficyna Wydawnicza, PLJ, Warszawa, 1999.

[184] Józefowska J., Węglarz J., *On a methodology for discrete-continuous scheduling*, European Journal of Operational Research, 1998, 107, 338-353.

[185] Johnson S.M., *Optimal two- and three-stage production schedules with setup times included*, Naval Res. Logist. Quart., 1954, 1, 61-68.

[186] Jung J.H., Lee H.K., Yang D.R., Lee I.B., Completion times and optimal scheduling for serial multi-product processes with transfer and set-up times in zero-wait policy, *Computers and Chemical Engineering*, 1994, 18, 537–544.

[187] Kachur R.G., Electronic firm combines plant move with switch to JIT manufacturing, *Industrial Engineering*, 1989, 21, 44–48.

[188] Kacprzyk J., Zbiory rozmyte w analizie systemowej, PWN 1986,

[189] Kanet J., Minimizing the Average Deviation of Job Completion Times About a Common Due Date, *Naval Research Logistics Quartely*, 1981, 28, 643–651.

[190] Karmarkar U.S., Lot sizes, lead times and in-process inventories, *Management Science*, 1987, 33, 409–418.

[191] Karmarkar U.S., Kekre S., Freeman S., Lotsizing and lead time performance in a manufacturing cell. *Interfaces*, 1985, 15, 1–9.

[192] Karp R.M., *Reducibility among combinatorial problems*, in: Miller, R.E. and Thatcher J.W. (red.) "Complexity of computer computations", New York, Plenum Press, 1972, 85-104.

[193] Kim T.M., Just-in-time manufacturing system: A periodic pull system, *International Journal of Production Research*, 1985, 23, 553–562.

[194] Kim Y.D., Lim H.G., Park M.W., Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process, *European Journal of Operational Research*, 1996, 91, 124–143.

[195] Kimura O., Terada H., Design and analysis of pull system: A method of multi-stage production control, *International Journal of Production Research*, 1981, 19, 241–253.

[196] King J.R., Spachis A.S., Heuristics for Flowshop Scheduling, *International Journal of Production Research*, 1980, 19, 345–357.

[197] Kirkavak N., Dincer C., Performance Evaluation Models for Single-Item Periodic Pull Production Systems, *Journal of Operational Research Society*, 1996, 47, 239–250.

[198] Kirkpatrick S, Gelatt C.D., Vecchi M.P., Optimisation by simulated annealing, *Science*, 1983, 220, 671–680.

[199] Kise H., Ibaraki T., Mine H., Performance analysis of six approximation algorithms for the one-machine maximum lateness scheduling problem with ready times, *Journal of the Operations Research Society of Japan*, 1979, 22, 205–244.

[200] Korbut A.A., Finkelsztejn J.J., Programowanie dyskretne, PWN, Warszawa, 1974.

[201] Koulamas C., A new constructive heuristic for the flow shop scheduling problem, Technical Report, 1996.

[202] Koulamas C., Single-machine scheduling with time windows and earliness and tardiness penalties, *European Journal of Operational Research*, 1996, 91, 190–202.

[203] Krajewski L.J., King B.E., Ritzman L.P., Wong D.S., Kanban, MRP and shaping the production environment, *Working Paper Series 83–19*, College of Administrative Science, Ohio State University, 1983.

[204] Krone M.J., Steiglitz K., Heuristic-programming solution of a flowshop scheduling problem, *Operations Research*, 1974, 22, 629–638.

[205] Ku H.M., Karimi I., Completion time algorithm for serial multiproduct batch processes with shared storage, *Computers and Chemical Engineering*, 1990, 14, 49–69.

[206] Kubiak W., A pseudo-polynomial algorithm for a two-machine no-wait job-shop scheduling problem, *European Journal of Operational Research*, 1989, 43, 267–270.

[207] Kubiak W., Completion time variance minimization on a single machine is difficult, *Operations Research Letters*, 1993, 14, 49–59.

[208] Kubiak W., Sethi S., A note on "Level schedules for mixed-model assembly lines in just-in-time production systems", *Management Science*, 1991, 37, 121–122.

[209] Kubiak W., Minimizing variation of production rates in just-in-time systems: A survey, *European Journal of Operational Research*, 1993, 259–271.

[210] Kunde M., Steppat H., First fit decreasing scheduling on uniform multiprocessors, *Discrete Applied Mathematics*, 1985, 10, 165–177.

[211] Kuriyan K., Reklaitis G.V., Scheduling network flowshops so as to minimize makespan, *Computers and Chemical Engineering*, 1989, 13, 187–200.

[212] Kusiak A., Aplication of operational research models and techniques in flexible manufacturing systems, *European Journal of Operational Research*, 1986, 24, 336–345.

[213] Kusiak A., Heragu S., The facility layout problem, *European Journal of Operational Research*, 1987, 29, 229–251.

[214] Van Laarhoven P.J.M., Aarts E.H.L., Lenstra J.K., Job Shop Scheduling by Simulated Annealing. *Operations Research*, 1992, 40, 113–125.

[215] Labetoulle J., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Preemptive scheduling of uniform machines subject to release dates, *Progress in Combinatorial Optimization*, 1994, Academic Press, Florida, 245–261.

[216] Lageweg B.J., Lenstra J.K., Rinnooy Kan A.H.G., Job-Shop Scheduling by Implicit Enumeration, *Management Science*, 1977, 24, 441–450.

[217] Larsson T., Patriksson M., Strömberg A.B.: Conditional subgradient optimization – Theory and application, *European Journal of Operational Research*88, 1996, 382-403.

[218] Lai T.C., A Note on Heuristics of Flow-Shop Scheduling, *Technical Report*, National Taiwan University, 1995.

[219] Lakshiminarayan I., Lakshanan R., Papineau R., Rochete R., Optimal single-machine scheduling with earliness and tardiness penalties, *Operations Research*, 1978, 26, 1079–1082.

[220] Larson N., Kusiak A., Work-in-Process Space Allocation – A Model and an Industrial Application, *IIE Transactions*, 1995, 27, 497–506.

[221] Lawler E.L., *On scheduling problems with deferral costs*, Management Sci., vol. 11, 1964, 280-288.

[222] Lawler E.L., Optimal sequencing of a single machine subject to precedence constraints, *Management Science*, 1973, 19, 544–546.

[223] Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Recent Developments in Deterministic Sequencing and Scheduling: A Survey. [In:] M.A.H.

Dempster, Lenstra J.K. , Rinnooy Kan A.H.G. (Eds.) *Deterministic and Stochastic Scheduling*, Reichel, Dordrecht, The Netherlands, 1982.

[224] Lenstra J.K., Sequencing by Enumeration Methods, Mathematical Centre Tract 69, Mathematisch Centrum, Amsterdam, 1977.

[225] Lenstra J.K., Rinnooy Kan A.H.G., and Brucker P., Complexity of machine scheduling problems, *Operations Research*23, 1975, 475-482.

[226] Lawrence S., Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement). Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, 1984.

[227] Lee H.K., Jung J.H., Lee I.B., An evolutionary approach to optimal synthesis of multiproduct batch plant, *Computers and Chemical Engineering*, 1996, 20, 1149–1157.

[228] Lee I., Shaw M.J., A neural-net approach to real-time flow shop sequencing, *Computers Industrial Engineering*, 2000, 38, 125–147.

[229] Lee L.C., Parametric appraisal of the JIT systemm, *International Journal of Production Research*, 1987, 25, 1415–1429.

[230] Levulis R., Group Technology 1978: The State-of-the-Art. K.W. Tunnel Company, Chicago, 1978.

[231] Li A., Co H.C., A dynamic programming model for the kanban assignment problem in a multi-period production system. *International Journal of Production Research*, 1991, 29, 1–16.

[232] Lipski W., Kombinatoryka dla programistów, WNT, Warszawa, 1982.

[233] Lourenço H.R.: Sevast'yanov's algorithm for the flow-shop scheduling problem, *European Journal of Operational Research*91, 1996, 176–189.

[234] McCormick S.T., Pinedo M.L., Shenker S., Wolf B., Sequencing in an Assembly Line with Blocking to Minimize Cycle Time, *Operations Research*, 1989, 37, 925–935.

[235] MacMahon G.B., Florian M., On scheduling with ready times and due dates to minimize maximum lateness, *Operations Research*, 1975, 23, 475–482.

[236] Mattfeld D.C., Kopfler H., Bierwirth C., Control of Parallel Population Dynamics: Social-Like Behavior of GA-Individuals Solves Large-Scale Job Shop Problems, Technical Report, Department of Economics, University of Bremen, 1995.

[237] Matsuo H.C., Suh C.J., Sullivan R.S., A Controlled Search Simulated Annealing Method for the General Jobshop Scheduling Problem. Working Paper 03-04-88, Department of Management, The University of Texas at Austin, 1988.

[238] McMahon G., Florian M., On scheduling with ready times land due dates to minimize maximum lateness, *Operations Research*, 23, 1975, 475-482.

[239] Miltenburg J., Level schedules for mixed-model assembly lines in just-in-time production systems, *Management Science*, 1989, 35, 192–207.

[240] Miltenburg J., Wijngaard J., Designing and phasing in just-in-time production systems, *International Journal of Production Research*, 1991, 29, 115–131.

[241] Miltenburg J., On the Equivalence of Jit and MRP as Technologies for Reducing Wastes in Manufacturing, *Naval Research Logistics*, 1993, 40, 905–924.

[242] Miltenburg J., Steiner G., Yeomans S., A dynamic programming algorithm for scheduling mixed-model just-in-time production sytems, *Mathematical and Computer Modelling*, 1990, 13, 57-66.

[243] Miyazaki S., Nishiyama N., Hashimoto F., An adjacent pairwise approach to the mean flow-time scheduling problem, *Journal of the Operations Research Society of Japan*, 1978, 21, 287–299.

[244] Miyazaki S., Nishiyama N., Analysis for minimizing weighted mean flow-time in flow-shop scheduling, *Journal of the Operations Research Society of Japan*, 1980, 23, 118–132.

[245] Miyazaki Y., Ohta H., Nashiyama N., The optimal operating of kanban to minimize the total operation cost, *International Journal of Production Research*, 1990, 26, 1605–1611.

[246] Modi A.K., Karimi I.A., Design of multiproduct batch processes with finite intermediate storage, *Computers and Chemical Engineering*, 1989, 13, 127–139.

[247] Monden Y., How Toyota shortened supply lot production time, waiting time and conveyance time, *Industrial Engineering*, 1981, 22, 22–30.

[248] Monden Y., Toyota Production System. Institute of Industrial Enginners, Atlanta, 1983.

[249] Mosheiov G., V-Shaped Policies for Scheduling Deteriorating Jobs, *Operations Research*, 1991, 979–991.

[250] Mühlenbein H., Gorges-Schleuter I., Die Evolutionsstrategie: Prinzip für parallele Algorithmen, GMD Annual Report, 1988.

[251] Musier R.F.H., Evans L.B., An approximate method for the production scheduling of industrial batch processes with parallel units, *Computers and Chemical Engineering*, 1989, 13, 229–238.

[252] Nawaz M., Enscore Jr. E.E., Ham I., A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, *OMEGA International Journal of Management Science*11, 1983, 91-95.

[253] Nissanci I., Nicoll A.D., Project planning network is integrated plan for implementing just-in-time, *Industrial Engineering*, 1987, 19, 50–55.

[254] Nori V.S., Sarker B.R., Cyclic Scheduling for a Multiproduct, Single-Facility Production System Operating Under a Just-in-Time Delivery Policy, *Journal of Operational Research Society*, 1996, 47, 930–935.

[255] Nowicki E., Smutnicki C., On lower bounds on the minimum maximum lateness on one machine subject to release dates, *OPSEARCH*, 1987, 24, 106–110

[256] Nowicki E., Smutnicki C., *Resource-constrained project scheduling. Basic properties*. Lecture Notes in Economics and Mathematical Systems 351, Springer-Verlag, 1988, 229-235.

[257] Nowicki E., Smutnicki C., *Modelowanie ograniczeń buforowania w systemie przepływowym*, Elektrotechnika, 1997, tom 1, z. 1, 330-336.

[258] Nowicki E., Smutnicki C., Zdrzałka S., *Algorytmy aproksymacyjne w wybranych zagadnieniach kolejnościowych przy kryterium minimalizacji sumy kar*, Archiwum Automatyki i Telemechaniki, 1989, 34, 289-299.

[259] Nowicki E., Zastosowanie techniki tabu search do harmonogramowania elastycznych gniazd produkcyjnych, Zesz. Nauk. Pol. Śl., Ser. Automatyka, 1996, 118, 153-154.

[260] Nowicki E., The permutation flow shop with buffers. A tabu search approach, European Journal of Operational Research, 1999, 116, 205-219.

[261] Nowicki E., Metoda tabu w problemach szeregowania zadań produkcyjnych, Prace naukowe Instytutu Cybernetyki Technicznej Pol. Wrocł., Seria Monografie, 1999.

[262] Nowicki E., Smutnicki C., Worst-case analysis of an approximation algorithm for flow-shop scheduling, *Operations Research Letters*8, 1989, 171-177.

[263] Nowicki E., Smutnicki C., Worst-case analysis of Dannenbring's algorithm for flow-shop scheduling, *Operations Research Letters*10, 1991, 473-480.

[264] Nowicki E., Smutnicki C., New results in the worst-case analysis for flow-shop scheduling, *Discrete Applied Mathematics*46, 1993, 21-41.

[265] Nowicki E., Smutnicki C., A note on worst-case analysis of approximation algorithms for a scheduling problem, *European Journal of Operational Research*74, 1994, 128-134.

[266] Nowicki E., Smutnicki C., An approximation algorithm for a single-machine scheduling problem with release times and delivery times, *Discrete Applied Mathematics*48, 1994, 69-79.

[267] Nowicki E., Smutnicki C., The flow shop with parallel machines. A tabu search approach, *European Journal of Operational Research*106, 1998, 226-253).

[268] Nowicki E., Smutnicki C., A fast taboo search algorithm for the job shop problem, *Management Science*42, 1996,797-813.

[269] Nowicki E., Smutnicki C., A fast taboo search algorithm for the permutation flow-shop problem, *European Journal of Operational Research*91, 1996, 160-175.

[270] Nowicki E., Zdrzałka S., A note on minimizing maximum lateness in a one-machine sequencing problem with release dates, *European Journal of Operational Research*23, 1986, 266-267.

[271] Occena L.G., Yokota T., Modelling of an automated guided vehicle system (AGVS) in a just-in-time (JIT) enivironment. *International Journal of Production Research*, 1991, 29, 495–511.

[272] Ogbu F.A., Smith D.K., The application of the simulated annealing algorithm to the solution of the $n/m/C_{\max}$ flowshop problem. *Computer and Operations Research*, 1990, 17, 243–253.

[273] O'Neal C.R., The buyer–seller linkage in a just-in-time environment, *Journal of Purchasing Matl. Management*, 1987, 23, 8–13.

[274] Ohly K., Sterowanie produkcją na żądanie, Praca magisterska, PWr, 1998.

[275] OR library. *Internet: http://mscmga.ms.ic.ac.uk/info.html*. jobshop1.txt, jobshop2.txt.

[276] Osman I.H., Potts C.N., Simulated Annealing for Permutation Flow Shop Scheduling, *OMEGA International Journal of Management Science*, 1989, 17, 551–557.

[277] Page E.S., An Approach to Scheduling Jobs on Machines, *Journal of Royal Statistics Society*, 1961, B23, 484–492.

[278] Palmer D.S., Sequencing Jobs Through a Multi-Stage Process in the Minimum Total Time - A Quick Method of Obtaining Near Optimum. *Operations Research Quarterly*, 1965, 16, 101–107.

[279] Panwalker S.S., Iskander W., A survey of scheduling rules, *Operations Research*, 1977, 25, 45–61.

[280] Panwalker S., Smith M., Seidmann A., Common Due Date Assignment to Minimize Total Penalty for the One Machine Scheduling Problemm, *Operations Research*, 1982, 30, 391–399.

[281] Papadimitriou C.H., Kenellakis P.C., Flowshop Scheduling with Limited Temporary Storage, *Journal of Association for Computing Machinery*, 1980, 27, 533–549.

[282] Parnaby J., A systems approach to the implementation of JIT methodologies in Lucas industries, *International Journal of Production Research*, 1988, 26, 483–492.

[283] Pekny J.F., Miller D.L., Exact solution of the no-wait flowshop scheduling problem with a comparison to heuristic methods, *Computers and Chemical Engineering*, 1991, 15, 741–748.

[284] Perkins J.R., Kumar P.R., Optimal-Control of Pull Manufacturing Systems, *IEEE Transactions on Automatic Control*, 1995, 40, 2040–2051.

[285] Pesch E., Learning in automated manufacturing. A local search approach, Physica, Heidelberg 1994.

[286] Philipoom P.R., Rees L.P., Taylor B.W., Huang P.Y., An investigation of the factors influencing the number of Kanbans required in the implementation of the JIT technique with Kanbans, *International Journal of Production Research*, 1987, 25, 457–472.

[287] Philipoom P.R., Rees L.P., Taylor B.W., Huang P.Y., A mathematical programming approach for detrmining workcentre lotsizes in just-in-time system with signal Kanbans, *International Journal of Production Research*, 1990, 28, 1–15.

[288] Pinedo M.L., Scheduling: Theory, Algorithms, and Systems, Prentice Hall, Englewood Cliffs, 1995.

[289] Porteus E.L., Investing in reduced set-ups in the EOQ model, *Management Science*, 1985, 31, 998–1010.

[290] Porteus E.L., Optimal lotsizing, process quality improvement and set-up cost reduction, *Operations Research*, 1986, 34, 137–144.

[291] Potts C.N., Analysis of a heuristic for one-machine sequencing with release dates and delivery times, *Operations Research*, 1980, 28, 1436-1441.

[292] Potts C.N., Analysis of linear programming heuristic for scheduling unrelated parallel machines, *Discrete Applied Mathematics*, 1985, 10, 155–164.

[293] Potts C.N., Shmoys D.B., Williamson D.P., Permutation vs non-permutation shop schedules, *Operations Research Letters*10, 1991, 281–284.

[294] Przysada J., Szecówka P., Systemy wspomagające zarządzanie projektami, Raport ICT PWr PRE 36/94 1994.

[295] Rajagopalan D., Karimi I.A., Completion times in serial mixed-storage multiproduct processes with transfer and set-up times, *Computers and Chemical Engineering*, 1989, 13, 175–186.

[296] Rajendran C., Chaudhuri D., An efficient heuristic approach to the scheduling of jobs in a flow-shop, *European Journal of Operational Research*, 1991, 61, 318–325.

[297] Rajendran C., Chaudhuri D., A flowshop scheduling algorithm to minimze total flowtime, *Journal of the Operations Research Society of Japan*, 1991, 34, 28–46.

[298] Rajendran C., Heuristic algorithm for scheduling in a flowshop to minimize total flowtime, *International Journal of Production Economics*, 1993, 29, 65–73.

[299] Rajendran C., A heuristic for scheduling in flowshop and flowline-based manufacturing cell with multiple-criteria, *International Journal of Production Research*, 1994, 32, 2541–2558.

[300] Rajendran C., Heuristics for scheduling in flowshop with multiple objectives, *European Journal of Operational Research*, 1995, 82, 540–555.

[301] Ramsay M.L., Brown S., Tabibzadeh K., Push, pull and squeeze shop floor control with simulation, *Industrial Engineering*, 1990, 22, 39–45.

[302] Rand G.K., MRP, JIT and OPT, [In:] Hendry L.G. and Eglese R.W. (Eds.), Operational Research Society 1990. Birmingham, 103–136.

[303] Rees L.P., Huang P.Y., Taylor II B.W., A comparative analysis of an MRP lot-for-lot system and a Kanban system for a multi-stage operation, *International Journal of Production Research*, 1989, 27, 1427–1443

[304] Reeves C., Heuristics for scheduling a single machine subject to unequal job release times, *European Journal of Operational Research*, 1995, 80, 397–403.

[305] Reeves C.R., A genetic algorithm for flowshop sequencing, *Computer and Operations Research*, 1995, 22, 5–13.

[306] Reeves C., Yamada T., Genetic Algorithms, Path Relinking and the Flowshop Sequencing Problem, *Evolutionary Computation*, 1995, 80, 397–403.

[307] Richmond L.E., Blackstone J.H.Jr, Just-in-time in the plastics processing industry, *International Journal of Production Research*, 1988, 26, 27–34.

[308] Rinnooy Kan A.H.G., Machine Scheduling Problems: Classification, Complexity and Computations, Nijhoff, The Hague, 1976.

[309] Röck H., Schmidt G., Machine Aggregation Heuristics in Shop Scheduling. *Methods of Operations Research*, 1982, 45, 303–314.

[310] Rodrigues M.T.M., Gimeno L., Passos C.A.S., Campos T., Reactive scheduling approach for multipurpose chemical batch plants, *Computers and Chemical Engineering*, 1996, 20, 1215–1220.

[311] Ronen D., Perspectives on practical aspects of truck routing and scheduling, *European Journal of Operational Research*, 1988, 35, 137–145.

[312] Santos D.L., Hunsucker J.L., Deal D.E., Global lower bounds for flow shops with multiple processors, *European Journal of Operational Research*, 1995, 80, 112–120.

[313] Sarker B.R., Harris R.D., The effect of imbalance in a just-in-time production system: A simulation study, *International Journal of Production Research*, 1988, 21, 1–18.

[314] Sarker B.R., Fitzsimmons J.A., The performance of push and pull systems: A simulation and comparative study, *International Journal of Production Research*, 1989, 27, 1715–1732.

[315] Sassani F., A simulation study on performance improvement of group technology cells, *International Journal of Production Research*, 1990, 28, 293–300.

[316] Sawik T., Optymalizacja dyskretna w elastycznych systemach produkcyjnych, Warszawa, WNT, 1992.

[317] Sawik T., Planowanie i sterowanie produkcji w elastycznych systemach montażowych, Warszawa, WNT, 1996.

[318] Sawik T., Multilevel scheduling of multistage production with limited in process inventory, *Journal of Operational Research Society*, 1987, 38, 651–664.

[319] Sawik T., Hierarchical scheduling flow shop with parallel machines and finite in process buffers, *Archiwum Automatyki i Telemechaniki*, 1988, 33, 403–411.

[320] Sawik T., A scheduling algorithm for flexible flow lines with limited intermediated buffers. *Applied Stochastic Models and Data Analysis*, 1993, 9, 127–138.

[321] Sawik T., Scheduling flexible flow lines with no in-process buffers. *International Journal of Production Research*, 1995, 33, 1359–1370.

[322] Scanzon L., JIT Theory Goes On Line At Sundstrand Data Control, *Industrial Engineering*, 1989, 34, 31–34.

[323] Schonenberger R.G., Some observations on the advantages and implementation issues of just-in-time prodURction systems, *Journal of Opertions Management*, 1982, 3, 1–11.

[324] Shonenberger R.J., Japaneese Manufacturing Techniques: Nine Lessons in Simplicity. The Free Press, New York, 1982.

[325] Schonberger R.J., Gilbert J.P., Just-in-time purchasing: A chalenge for U.S. industry. *Cali. Mgmt. Rev.* , 1983, XXVI, 54–68.

[326] Schrage L., Obtaining optimal solution to resource constrained network scheduling problem, unpublished manuscript, 1971.

[327] Schrage L., Baker K.R., Dynamic Programming Solution of Sequencing Problems with Precedence Constraints, *Operations Research*, 1978, 26, 444–449.

[328] Schrorer B.J., Microcomputer analyzes 2-card Kanban system just-in-time small batch production, *Industrial Engineering*, 1984, 16, 54–65.

[329] Sevast'yanov S.V., On an asymptotic approach to some problems in scheduling theory, *Abstracts of papers at 3rd All-Union Conference of Problems of Theoretical Cybernetics (Russian)*, Novosibirsk, 1974, 67–69.

[330] Shaukat A. B., Hunsucker J.L., Branch and bound algorithm for the flow shop with multiple processors. *European Journal of Operational Research*, 1991, 51, 88–99.

[331] Shmoys D.B., Stein C., Wein J., Improved Approximation Algorithms for Shop Scheduling Problems, *Proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms*, 1991.

[332] Sidney J.B., Optimal single-machine scheduling with earliness and tardiness penalties, *Operations Research*, 1977, 25, 62–69.

[333] Silver E.A., Peterson R., Decision Systems for Inventory Management and Production Planning. Wiley, New York, 1985.

[334] Simers D., Priest J., Gary J., Just-in-time techniques in process manufacturing reduced lead time, cost; raise productivity, quality. *Industrial Engineering*, 1989, 21, 19–23.

[335] Simulation, http://www.idsia.ch

[336] Singh N., Shek K.H., Meloche D., The developement of a Kanban system: A case study. *International Journal on Operational Production Management*, 1990, 10, 28–36.

[337] Sipper D., Shapira R., JIT vs. WIP-a trade-off analysis, *International Journal of Production Research*, 1989, 27, 903–914.

[338] Słowiński R., Hapke M., Scheduling under fazziness, *Studies in Fuzziness and Soft Computing*, Springer, vol. 37, 2000.

[339] Słowiński R., Węglarz J. (Eds.), *Advances in project scheduling*, Elsevier, Amsterdam, 1989.

[340] Smith W.E., Various optimizers for single-state production, *Naval Research Logistics Quartely*3, 1956, 56-66.

[341] Smutnicki C., O pewnej klasie algorytmó aproksymacyjnych dla problemów szeregowania, *Zeszyty Naukowe PŚl.*, Seria: Automatyka, 1990, 100, 311–321.

[342] Smutnicki C., Some results of the worst-case analysis for flow shop scheduling, *European Journal of Operational Research*, 1998, 109, 66–87.

[343] Special issue on decision support system for RCPS, *European Journal of Operational Research*, 1994, 79.

[344] Smutnicki C., Optimization and control in just-in time manufacturing systems, Prace naukowe Instytutu Cybernetyki Technicznej Pol. Wrocł., Seria Monografie, 1997.

[345] Spachis A.S., King J.R., Job-Shop Scheduling Heuristics with Local Neighborhood Search. *International Journal of Production Research*, 1979, 17, 507–526.

[346] Spence A.M., Porteus E.L., Set-up reduction and increased effective capacity, *Management Science*, 1987, 33, 1291–1301.

[347] Sriskandarajah C., Sethi S.P., Scheduling algorithms for flexible flowshops: Worst and average case performance. *European Journal of Operational Research*, 1989, 43, 43–60.

[348] Stopler S., Bierwirth C., The application of parallel genetic algorithm to the $n/m/P/C_{\max}$ flow shop problem, Working paper, University of Bremen, 1992.

[349] Storer R.H., David Wu S., Vaccari R., New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling, *Management Science*, 1992, 10, 1495–1509.

[350] Sugimori Y., Kasunoki F.C., Uchikawa S., Toyota production system and Kanban system, materialization of just-in-time and respect-for-human system. *International Journal of Production Research*, 1977, 15, 553–564.

[351] Sysło M., Deo M., Kowalik J.S., Algorytmy optymalizacji dyskretnej z programami w języku Pascal, Warszawa, 1993.

[352] Szwarc W., Solution of the Akers-Friedman problem, Opns. Res., vol. 8, 1960, 782-788.

[353] Szwarc W., Optimal elimination method in the $m \times n$ problem, *Operations Research*, 1973, 20, 1250–1259.

[354] Tadeusiewicz R., Sieci neuronowe, Akademicka Oficyna Wydawnicza, Warszawa, 1993.

[355] Taillard E., home page http://www.eivd.ch/ina/collaborateurs/etd/default.htm

[356] Taillard E., Parallel Taboo Search Technique for the Jobshop Scheduling Problem. Working Paper ORWP 89/11 (revised version October 1992), Departement de Mathematiques, Ecole Polytechnique Federale De Lausanne, Lausanne, Switzerland, 1989.

[357] Taillard E., Some efficient heuristic methods for flow shop sequencing, *European Journal of Operational Research*47, 1990, 65-74.

[358] Taillard E., Benchmarks for basic scheduling problems, *European Journal of Operational Research*64, 1993, 278-285.

[359] Taheri J., Northern telecom tackles succesfull implementation of cellular manufacturing, *Industrial Engineering*, 1990, 22, 38–43.

[360] Tanaev V.S., Gordon V.S. and Shafransky Y.M., Scheduling Theory. Single Stage Systems, Kluwer Academic Publishers, Dordrecht, 1994.

[361] Tanaev V.S., Sotskov Y.N. and Strusevich Y.A., Scheduling Theory. Multi Stage Systems, Kluwer Academic Publishers, Dordrecht, 1994.

[362] Tandon M., Cummings P.T. , LeVan M.D., Flowshop sequencing with non-permutation schedules, *Computers and Chemical Engineering*, 1991, 15, 601–607.

[363] The dutch national logistics site http://www1.tip.nl/t470430/

[364] Toczyłowski E., Modelling FMS operational scheduling problems by m-processors, *Elektrotechnika*, 1995, 14, 429–436.

[365] Turner S., Booth D., Comparison of heuristics for flow shop sequencing. *OMEGA International Journal of Management Science*, 1987, 15, 75–78.

[366] Weng X., Ventura J.A., Scheduling about a given due date to minimize mean squared deviation of completion times, *European Journal of Operational Research*, 1996, 88, 328–335.

[367] Vaessens R.J.M., Aarts E.H.L., Lenstra J.K., Job Shop Scheduling by Local Search, *Memorandum COSOR 94-05*, Eidhoven University of Technology, 1994.

[368] Vaessens R.J.M., Generalised Job Shop Scheduling: Complexity and Local Search, Ph.D.Thesis, Ponsen & Looijen BV, Wageningen, The Netherlands, 1995.

[369] Vandervelde A., Minimizing the makespan in a muliprocesor flowshop, Master's thesis, Eindhoven University of Technology, 1994.

[370] Vaessens R.J.M., OR library, Internet: http://mscmga.ms.ic.ac.uk.-info.html. Files flowshop1.txt, flowshop2.txt.

[371] Vaessens R.J.M., Aarts E.H.I., Lenstra J.K., Job shop scheduling by local search, INFORMS Journal on Computing, 1996, 8, 302-317.

[372] Wala K., Chmiel W., *Local search and genetic heuristic for the permutational optimization problem*, Proc. of 7th International Symposium on System-Modeling-Control, 1993, Zakopane, 252-257.

[373] Wala K., Chmiel W., Investigation of cross-over genetic operators for permutation optimization problems, *Elektrotechnika*, 1995, 14, 451–458.

[374] Wala K., *Algorytm ewolucyjny harmonogramowania procesów wytwarzania w systemach komputerowo zintegrowanych*, Zesz. Nauk. Pol. Śl., Ser. Automatyka, 1998, 125, 249-259.

[375] Walukiewicz S., Programowanie dyskretne, PWN 1986.

[376] Wellsons M.C., Reklaitis G.V., Optimal schedule generation for a single-product production line-I. Problem formulation., *Computers and Chemical Engineering*, 1989, 13, 201–212.

[377] Wellsons M.C., Reklaitis G.V., Optimal schedule generation for a single-product production line-II. Identification of dominant unique path sequences, *Computers and Chemical Engineering*, 1989, 13, 213–227.

[378] Wemmerlöw U., Hyer N.L., Procudures for the part family/machine group identification problem in cellular manufacture, *Journal of Opertions Management*, 1986, 6, 125–147.

[379] Wemmerlöw U., Hyer N.L., Research issues in cellular manufacturing *International Journal of Production Research*, 1987, 25, 413–431.

[380] Werner F., On the Heuristic Solution of the Permutation Flow Shop Problem, *Computers and Operations Research*, 1993, 20, 707–722.

[381] Werner F., Winkler A., Insertion Techniques for the Heuristic Solution of the Job Shop Problem. Report, Technische Universität "Otto von Guericke", Magdeburg, 1992.

[382] Węglarz J., *Project scheduling with continuously-divisible double constrained resource*, Management Science, 1981, 27, 1040-1053.

[383] Widmer M., Hertz A., A new heuristic method for the flow shop sequencing problem, *European Journal of Operational Research*, 1989, 41, 186–193.

[384] Wittrock R.J., An adaptable scheduling algorithm for flexible flow lines. *Operations Research*, 1988, 36, 445–453.

[385] Williamson D.P., Hal L.A., Hoogeven J.A., Hurkens C.A.J., Lenstra J.K., Sevast'janov S.V., Shmoys D.B., Short Shop Schedules, *Operations Research*45, 1997, 288–294.

[386] Woodruff D.L., Zemel E., Hashing vectors for tabu search, *Annals of Operations Research*, 1993, 41, 123–137.

[387] Woodruff D.L., Proposals for Chunking and Tabu Search, Technical Raport, Graduate School of Management, Davis, 1994.

[388] Wróblewski K.J., Krawczyński R., Kosieradzka A., Kasprzyk S., Reguły priorytetu w sterowaniu przepływem produkcji, WNT 1984.

[389] Yeomans J.S., A Simple Sequencing Algorithm for Mixed-Model Assembly Lines in Just-in-Time Production Systems – Comment, *Operations Research Letters*, 1995, 16, 299–301.

[390] Yih Y., Trace-diven knowledge acquisition (TDKA) for rule based real time scheduling system, *Journal of Intelligent Manufacturing*, 1991, 217–230.

[391] Zangwill W.I., From EOQ towards ZI. *Management Science*, 1987, 33, 1209–1223.

[392] Zdrzałka S., Grabowski J., An algorithm for single machine sequencing with release dates to minimize maximum cost, *Discrete Applied Mathematics*, 1989, 23, 73–89.

[393] Zhou D., Cherkassky V., Baldwin T.R., Olson D.E., A neural network approach to job-shop scheduling, *IEEE Transactions on Neural Networks* 2, 1991, 175–179.