Wydział Elektroniki
Katedra Automatyki, Mechatroniki i Systemów Sterowania
Politechnika Wrocławska

Seria: PRE nr W04/2021/P-023

Algorytmy optymalizacyjne dla problemu szeregowania zadań ze sprzężeniami czasowymi

(rozprawa doktorska)

Radosław Idzikowski

PROMOTOR:
Prof. dr hab. Wojciech Bożejko

Słowa kluczowe:
–optymalizacja
–szeregowanie zadań
–sprzężenia czasowe
–algorytmy

WROCŁAW, maj 2021

Spis treści

Spis treści **3**

Lista symboli **7**

Lista skrótów **11**

Streszczenie **13**

Abstract **14**

1 Wprowadzenie **15**
1.1 Wstęp 15
1.2 Tezy pracy 16
1.3 Zakres pracy 16

I Klasyfikacja problemów i metod szeregowania zadań **19**

2 Problemy szeregowania zadań **21**
2.1 Rodzaje problemów szeregowania zadań 22
2.2 Problemy przepływowe 24
2.3 Dodatkowe ograniczenia problemów 28
2.4 Funkcje celu 35
2.5 Wnioski i uwagi 37

3 Algorytmy optymalizacyjne **39**
3.1 Metody dokładne 39
3.2 Metody przybliżone 43
3.2.1 Algorytmy konstrukcyjne 43
3.2.2 Algorytmy przeszukiwania lokalnego 44
3.2.3 Algorytmy Populacyjne 48

3.3 Wnioski i uwagi 52

II Własności i algorytmy dla problemów przepływowych 53

4 Problemy ze sprzężeniami czasowymi 55
4.1 Permutacyjny problem przepływowy 56
4.1.1 Definicja problemu 56
4.1.2 Własności problemu 59
4.1.3 Algorytmy optymalizacyjne 63
4.1.4 Eksperymenty obliczeniowe 67
4.1.5 Wnioski i uwagi 68
4.2 Niepermutacyjny problem przepływowy 69
4.2.1 Definicja problemu 69
4.2.2 Własności problemu 73
4.2.3 Algorytmy optymalizacyjne 80
4.2.4 Eksperymenty obliczeniowe 83
4.2.5 Akceleracja funkcji celu 85
4.2.6 Wnioski i uwagi 92

5 Problemy z przezbrojeniami 95
5.1 Problem przepływowy bez przestojów 96
5.1.1 Definicja problemu 96
5.1.2 Własności 100
5.1.3 Algorytmy optymalizacyjne 103
5.1.4 Eksperymenty obliczeniowe 105
5.1.5 Wnioski i uwagi 106
5.2 Szeregowania rozłącznych przezbrojeń 108
5.2.1 Definicja problemu 108
5.2.2 Własności problemu 112
5.2.3 Algorytmy optymalizacyjne 122
5.2.4 Eksperymenty obliczeniowe 128
5.2.5 Wnioski i uwagi 133

III Zastosowania praktyczne 135

6 Aplikacje 137
6.1 Produkcja urządzeń AGD 137
6.1.1 Definicja problemu 138
6.1.2 Wnioski i uwagi 138

6.2 Transport spedycyjny 140
6.2.1 Definicja problemu 141
6.2.2 Algorytmy optymalizacyjne 144
6.2.3 Aplikacja użytkownika 147
6.2.4 Eksperymenty obliczeniowe 147
6.2.5 Wnioski i uwagi 150
6.3 Obiór od stałych dostawców 150
6.3.1 Definicja problemu 150
6.3.2 Aplikacja użytkownika 152
6.3.3 Wnioski i uwagi 153

7 Wnioski końcowe **155**

Bibliografia **159**

Spis tablic **173**

Spis rysunków **175**

Spis algorytmów **177**

Indeks **179**

Lista symboli

A	—	macierz najdłuższych ścieżek w grafie
$a_{u,v}$	—	długość ścieżki między wierzchołkami u i v
`A`	—	ruch typu zamień sąsiedni
B	—	blok operacji
B_a	—	blok na maszynie a
`B`	—	ruch typu zamień blokowy
b_i	—	operacja i-ta w bloku
$\boldsymbol{C}$	—	macierz momentów zakończenia wykonywania operacji
C_i	—	moment zakończenia zadania i
C_i^a	—	moment zakończenia operacji i-tej na maszynie a
${}^{\sigma}\boldsymbol{C}$	—	macierz momentów zakończenia wykonywania przezbrojeń
${}^{\sigma}C_i^a$	—	moment zakończenia przezbrojenia po operacji i-tej na maszynie a
$C_{\max}$	—	długość uszeregowania, czas zakończenia wszystkich zadań
$\mathcal{C}_n$	—	n-ta liczba Catalana
`C`	—	kadencja
d_i	—	żądany czas zakończenia i-tego zadania
$\hat{d}_a$	—	maksymalny czas przestoju na maszynie a
$\hat{D}_i^a$	—	waga łuku powrotnego z wierzchołka $(a, i+1)$ do (a, i) dla PFSSP-TC oraz FSSP-TC
$\mathcal{D}(\sigma_k, i)$	—	liczba przezbrojeń wykonanych na maszynie a w σ_k
E_1	—	zbiór krawędzi ograniczeń maszynowych w grafie $\mathcal{G}(\pi)$
E_2	—	zbiór krawędzi ograniczeń technologicznych w grafie $\mathcal{G}(\pi)$
E_3	—	zbiór krawędzi powrotnych w grafie $\mathcal{G}(\pi)$
E_4	—	zbiór krawędzi powrotnych pionowych w grafie $\mathcal{G}(\pi)$
$\mathcal{G}(\pi)$	—	graf dla kolejności π
H	—	trójwymiarowa tablica przechodnich domknięć ścieżek
$h_{u,w,v}$	—	tymczasowa długość najkrótszej ścieżki przechodzącej od wierzchołka u do wierzchołka v, który przechodzi przez wierzchołek w

$\mathtt{I}$	—	ruch typu wstaw
$\mathcal{J}$	—	zbiór zadań
$\mathtt{K}$	—	dozwolona liczba iteracji bez poprawy
L	—	ścieżka kratowa
L_l	—	punkt l-ty na ścieżce kratowej L
$\mathrm{len}(p)$	—	długość ścieżki p
$\mathrm{len}^a(p)$	—	długość ścieżki p^a na maszynie a
λ_i	—	długość i-tego zadania/ładunku
Λ	—	maksymalna długość zadania/ładunku dla jednego pojazdu/maszyny
$\mathcal{M}$	—	zbiór maszyn
m	—	liczba maszyn
$\mathtt{M}$	—	ruch typu zamień maksymalny
n	—	liczba zadań
N	—	zbiór zadań nieuszeregowanych
$\mathcal{N}(\pi)$	—	sąsiedztwo rozwiązania π
ν	—	węzeł
ν_{L}	—	poziom węzła
ν_{N}	—	zbiór zadań nieuszeregowanych na danym poziomie
ν_π	—	rozwiązanie częściowe w węźle
$O(f(n))$	—	asymptotyczne górne oszacowanie funkcji $f(n)$
$\mathcal{O}$	—	zbiór wszystkich operacji
$\mathcal{O}_i$	—	zbiór operacji zadania i-tego
o	—	całkowita liczba operacji
o_i	—	liczba operacji w zadaniu i
Ω	—	maksymalna waga/masa dla jednego pojazdu/maszyny
ω_i	—	waga/masa i-tego zadania/ładunku
p	—	ścieżka
p^c	—	ścieżka krytyczna
p_i^a	—	czas wykonywania i-tej operacji na maszynie a
$\mathcal{P}_k$	—	k-ta populacja
$\mathcal{P}_0$	—	populacja początkowa
π	—	rozwiązanie, kolejność
$\boldsymbol{\pi}$	—	rozwiązanie, krotka kolejności na m maszynach
$\pi(i)$	—	funkcja zwracająca indeks operacji z zadania i-tego w kolejności π
π^{-1}	—	funkcja odwrotna do π
π'	—	rozwiązanie sąsiednie do π
π^*	—	najlepsze znalezione rozwiązanie
π^{opt}	—	rozwiązanie optymalne

π^{ref}	—	rozwiązanie referencyjne
$\pi_a(i)$	—	funkcja zwracająca indeks operacji z zadania i-tego w kolejności π_a na maszynie a
π_a^{-1}	—	funkcja odwrotna do π_a
Π	—	zbiór wszystkich możliwych rozwiązań, kolejności π
r_i	—	czas dostępności i-tego zadania
$\hat{r}_a$	—	minimalny czas przestoju na maszynie a
Q	—	kolejka priorytetowa
q_i	—	czas dostarczenia i-tego zadania
$s_{i,j}^a$	—	czas wykonywania przezbrojenia miedzy operacjami i oraz j na maszynie a
$\hat{s}_{i,j}^a$	—	koszt wykonywania przezbrojenia miedzy operacjami i oraz j na maszynie a
$\boldsymbol{S}$	—	macierz momentów rozpoczęcia wykonywania operacji
$\mathcal{S}$	—	zbiór przezbrojeń
$\hat{\mathcal{S}}$	—	macierz kosztów przezbrojeń
$\mathtt{S}$	—	ruch typu zamień
$\mathtt{S}_p$	—	przyśpieszenie
S_i	—	moment rozpoczęcia wykonywania zadania i
S_i^a	—	moment rozpoczęcia wykonywania operacji i-tej na maszynie a
${}^{\sigma}\boldsymbol{S}$	—	macierz momentów rozpoczęcia wykonywania przezbrojeń
${}^{\sigma}S_i^a$	—	moment rozpoczęcia wykonywania przezbrojenia po operacji i-tej na maszynie a
$\hat{S}_i^a$	—	waga łuku powrotnego z wierzchołka $(a, i+1)$ do (a, i) dla NPFSSP-S
σ	—	kolejność wykonania przezbrojeń
σ_k	—	częściowa kolejność wykonania przezbrojeń składająca się k pierwszych elementów σ
Σ	—	zbiór wszystkich możliwych kolejności σ
Σ_{feas}	—	zbiór wszystkich dopuszczalnych kolejności σ
$t_i^{a,b}$	—	czas transportu między operacjami z maszyny a na b dla zadania i-tego
$t_{i,j}^a$	—	przezbrojenie po zadaniu i, a przed zadaniem j, na maszynie a
$\mathcal{T}$	—	zbiór wszystkich możliwych kolejności τ
$\mathtt{T}_o$	—	temperatura początkowa
T_i	—	spóźnienie i-tego zadania
τ	—	kolejność wykonania przezbrojeń
τ_i^a	—	przezbrojenie wykonywane po zadaniu i-tym na maszynie a

$\Theta_a(\nu_\pi)$	—	funkcja zwracająca ostatnie uszeregowanie zadanie na maszynie a w częściowym rozwiązaniu π
$\mathcal{U}$	—	podproblem
w_i	—	kara za spóźnienie i-tego zadania
x	—	zmienna decyzyjna w MILP
`W`	—	szerokość sąsiedztwa, liczba przeglądanych elementów

Lista skrótów

ABC	—	Sztuczna Kolonia Pszczół, *Artificial Bee Colony*
B&B	—	Metoda Podziału i Ograniczeń, *Branch and Bound method*
BF	—	metoda siłowa, *Brute Force*
BS	—	Przeszukiwanie Snopowe, *Beam Search*
CVRP	—	problem marszrutyzacji pojazdów z ograniczeniem pojemności pojazdów, *Capacitated Vehicle Routing Problem*
COVRP	—	otwarty problem marszrutyzacji pojazdów z ograniczeniem pojemności pojazdów, *Open Capacitated Vehicle Routing Problem*
DEAP	—	Rozproszone algorytmy ewolucyjne w jęzuku Ptyhon *Distributed Evolutionary Algorithms in Python*
EA	—	algorytm Ewolucyjny, *Evolutionary Algorithms*
FFSSP	—	elastyczny problem przepływowy szeregowania zadań, *Flexible Flow Shop Scheduling Problem*
FJSSP	—	elastyczny problem gniazdowy szeregowania zadań, *Flexible Job Shop Scheduling Problem*
FSSP	—	problem przepływowy szeregowania zadań, *Flow Shop Scheduling problem*
FSSP-TC	—	problem szeregowania zadań ze sprzężeniami czasowymi, *Flow Shop Scheduling Problem with Time Coupling*
GA	—	Algorytm Genetyczny, *Genetic Algorithm*
GSSP	—	problem ogólny szeregowania zadań, *General Shop Scheduling Problem*
GTR	—	reprezentacja wielkiej trasy, *Giant Tour Representation*
GUI	—	graficzny interfejs użytkownika, *Graphical User Interface*
HFSSP	—	hybrydowy problem przepływowy szeregowania zadań, *Hybrid Flow Shop Scheduling Problem*
IA	—	Algorytm Wyspowy, *Island Algorithm*
JSSP	—	problem gniazdowy szeregowania zadań, *Job Shop Scheduling Problem*

LB	—	dolne oszacowanie, *Lower Bound*
LP	—	Programowanie Liniowe, *Linear Programing*
LS	—	Przeszukiwanie Lokalne, *Local Search*
MA	—	Algorytm Memetyczny, *Memetic Algorithm*
MILP	—	Programowanie Liniowe Całkowitoliczbowe, *Mixed Integar Linear Programing*
NN	—	Najbliższy Sąsiad, *Nearest Neighbor*
NPFSSP-S	—	permutacyjny problem *no idle Permutation Flow Shop Scheduling Problem with Setups*
OSSP	—	otwarty problem szeregowania zadań, *Open Shop Scheduling Problem*
OVRP	—	otwarty problem marszrutyzacji pojazdów, *Open Vehicle Routing Problem*
PA	—	Algorytm Populacyjny, *Population-based Algorithm*
PD	—	Programowanie Dynamiczne, *Dynamic Programing*
PFSSP	—	permutacyjny problem przepływowy szeregowania zadań, *Permutation Flow Shop Scheduling Problem*
PFSSP-TC	—	permutacyjny problem szeregowania zadań ze sprzężeniami czasowymi, *Permutation Flow Shop Scheduling Problem with Time Coupling*
PMSP	—	problem szeregowania zadań z równoległymi maszynami, *Parallel Machines Scheduling Problem*
QS	—	Sortowanie Szybkie, *Quick Sort*
SA	—	Symulowane Wyżarzanie, *Simulated annealing*
SI	—	inteligencja roju, *Swarm Intelligence*
SMSP	—	jednomaszynowy problem szeregowania zadań, *Single Machine Scheduling Problem*
SOA	—	Algorytm Rojowy, *Swarm-based Optimization Algorithm*
TS	—	Przeszukiwanie z Zabronieniami, *Tabu Search*
TSC	—	Przeszukiwanie z Zabronieniami ze złożonymi ruchami, *Tabu Search witch Complex moves*
TSP	—	problem komiwojażera , *Travelling Salesman Problem*
UB	—	górne oszacowanie, *Upper Bound*
VRP	—	problem marszrutyzacji pojazdów, *Vehicle Routing Problem*
VRP-TW	—	problem marszrutyzacji pojazdów z oknami czasowymi, *Vehicle Routing Problem with Time Windows*

Streszczenie

Rozprawa dotyczy problemów szeregowania zadań ze sprzężeniami czasowymi. Pod pojęciem sprzężeń czasowych rozumiane są dodatkowe relacje między operacjami. W szczególności dotyczy to operacji przyporządkowanych do różnych zadań, ale wykonywanych na tej samej maszynie – np. ograniczony czas przestoju maszyny pomiędzy kolejno wykonywanymi operacjami. W pracy na przykładzie problemu przepływowego szeregowania zadań zaproponowano uogólniony model uwzględniania sprężeń czasowych. Za pomocą tego modelu można przedstawić ograniczenia znane już w literaturze, między innymi takie jak ograniczenie „bez przestojów" lub „z ograniczonym przestojem", a także nowe z „minimalnym" oraz „maksymalnym" przestojem maszyny. Dla różnych wariantów problemów przepływowych, w tym z przezbrojeniami, zaproponowano modele matematyczne oraz grafowe. Następnie wykazano szereg własności, w tym eliminacyjnych, dla omawianych problemów, oraz zaproponowano różne algorytmy optymalizacyjne wykorzystujące udowodnione własności.

Praca została podzielona na trzy części. Część pierwsza zawiera wprowadzenie do teorii szeregowania zadań wraz z obszernym przeglądem literatury. Część druga jest poświęconą problemom przepływowym szeregowania zadań ze sprzężeniami czasowymi. Dla omawianych problemów zaproponowano szereg dedykowanych algorytmów poczynając od metod liczenia funkcji celu, poprzez algorytmy wykrywające bloki, a kończąc na algorytmach optymalizacyjnych dla rozpatrywanych problemów. Omówiono i zaimplementowano algorytmy klasyczne, oraz uwzględniające najnowsze trendy w problematyce szeregowania zadań i innych problemach optymalizacji dyskretnej. W rozprawie wykorzystano następujące algorytmy oraz metody: Metoda Podziału i Ograniczeń, Programowanie Dynamiczne, Całkowitoliczbowe Programowanie Liniowe, Przeszukiwanie Snopowe, Przeszukiwanie z Zabronieniami, Sztuczna Kolonia Pszczół, Algorytm Genetyczny oraz algorytmy dedykowane, w tym konstrukcyjne. Poza analizą teoretyczną przeprowadzono również badania eksperymentalne dla badanych problemów z wykorzystaniem wymienionych powyżej algorytmów. Ostatnia część dysertacji została poświęcona praktycznym zastosowaniom zaproponowanych metod szeregowania zadań. Przedstawiono praktyczny problem szeregowania przezbrojeń na przykładzie produkcji urządzeń AGD. Następnie skupiono się na innych problemach optymalizacji dyskretnej, w tym na problemie marszrutyzacji pojazdów. Uzyskane rezultaty teoretyczne zostały wykorzystane w konstrukcji dedykowanych metod i aplikacji wspomagających zarządzanie flotą pojazdów w przedsiębiorstwach operujących w różnych branżach.

Abstract

In this thesis the problem of task scheduling with time couplings is considered. Time couplings are understood as additional relations between operations. In particular this includes operations from different jobs, but processed on the same machine – e.g. limited allowed idle time between subsequent operation processed on the machine. In the thesis, a generalized model for time couplings was proposed with the flowshop scheduling problem used as an example. This model can be used to portray both problem constraints known form the literature, including the „no idle" and „limited idle" constraints, as well as new „minimal" and „maximal" idle constraints. Mathematical and graph models were proposed for different variants of flowshop scheduling problems, including variants with setup times. Furthermore, a number of properties, including elimination properties, for the aforementioned problems were formulated. Several different optimization algorithms employing the proven problem properties were proposed.

The thesis was divided into three main parts. The first part contains an introduction to theory of task scheduling, including an extensive overview of the literature. The second part is dedicated to the topic of flowshop scheduling problems with time couplings. For those problems, a number of new algorithms were proposed. This includes methods for calculating the goal function, algorithms for detection of blocks of jobs as well as full solving methods for the aforementioned problems. The algorithms discussed and implemented in this thesis include both classic ones as well as one encompassing the newest trends in task scheduling and similar areas of discrete optimization. The following algorithms and methods were used throughout this thesis: Branch and Bound, Dynamic Programming, Mixed-Integer Linear Programming, Beam Search, Tabu Search, Artificial Bee Colony, Genetic Algorithms as well as dedicated algorithms, including constructive ones. Aside from theoretical analysis, experimental research on the aforementioned problems and algorithms were conducted. The third and final part of the thesis is dedicated to the practical applications of the proposed solving methods for the task scheduling problems. A practical problem of scheduling setups for the process of production of home appliances was described. Furthermore, applications for other discrete optimization problems were considered, including a Vehicle Routing Problem example. Formulated theoretical results of the thesis were employed for construction of dedicated solving methods and computer applications in order to aid the process of managing vehicle fleets in companies from various branches of the industry.

Rozdział 1

Wprowadzenie

Celem niniejszego rozdziału jest przedstawienie kierunku badań rozwijanego w rozprawie doktorskiej. Rozdział zawiera także tezy oraz szczegółowy zakres niniejszej rozprawy.

1.1 Wstęp

W procesie automatycznego zarządzania liniami produkcyjnymi czy procesami budowlanymi próbuje się dostosować proces do modeli znanych z literatury, często zaliczanych do klasy problemów silnie NP-trudnych. Z braku algorytmów dokładnych o wielomianowej złożoności obliczeniowej stosuje się algorytmy przybliżone inspirowane różnymi procesami zachodzącymi między innymi w naturze. W celu poprawy efektywności stosowanych metod rozważa się wprowadzanie nowych technik do istniejących już algorytmów lub tworzenie procedur równoległych wykorzystując środowiska wielowątkowe:

- wielordzeniowe procesory,
- karty graficzne,
- koprocesory,
- klastry obliczeniowe.

W wielu wypadkach okazuję się, że opisane w literaturze modele nie są wystarczające. Model zawsze jest przybliżeniem rzeczywistego procesu. Zastosowanie zbyt prostego modelu będzie skutkowało znalezieniem rozwiązania obarczonego dodatkowymi kosztami, których w modelu nie uwzględniono. Dlatego niniejsza rozprawa skupi się na zaproponowaniu nowych modeli, bliższych praktycznym zastosowaniom oraz przedstawieniu nowych dedykowanych algorytmów oraz sposobu dostosowania już istniejących metod.

1.2 Tezy pracy

Zasadniczym celem pracy jest wykazanie, że wykorzystując specyficzne własności problemów oraz metody projektowania algorytmów, możliwe jest:

- zaproponowanie nowych, bardziej ogólnych względem klasycznych podejść, modeli problemów szeregowania zadań z uwzględnieniem dodatkowych ograniczeń dotyczących zależności pomiędzy operacjami, tzw. sprzężeń czasowych,
- zmniejszenie rozmiaru przestrzeni rozwiązań problemów szeregowania zadań ze sprzężeniami czasowymi poprzez zastosowanie tzw. własności eliminacyjnych,
- zaprojektowanie dedykowanych algorytmów dokładnych i przybliżonych dla rozważanych klas problemów,
- przyśpieszenie obliczeń dzięki wykorzystaniu środowisk równoległych.

Powyżej sformułowane tezy zostaną w pracy udowodnione w następującym procesie badawczym:

- przedstawione zostaną modele matematyczne oraz grafowe znane z literatury oraz rozszerzone o rozłączne przezbrojenia lub sprzężenia czasowe.
- udowodnione zostaną własności eliminacyjne dla zaproponowanych modeli,
- opracowane zostaną efektywne, dedykowane algorytmy dokładne, przybliżone oraz równoległe,
- przeprowadzone zostaną eksperymenty obliczeniowe z wykorzystaniem danych testowych z literatury oraz nowych, wygenerowanych losowo.

Prowadzone badania mają charakter interdyscyplinarny obejmując dyscypliny: (1) automatyka, elektronika i elektrotechnika, (2) informatyka techniczna i telekomunikacja, (3) inżynieria lądowa i transport. Charakter badań wynika ze specyfiki dyscyplin i obejmuje: modelowanie problemów przemysłowych i transportowych, projektowanie algorytmów, obliczenia równoległe, metody sztucznej inteligencji, teorie złożoności obliczeniowej, teorię szeregowania zadań oraz teorię harmonogramowania procesów.

1.3 Zakres pracy

Praca została podzielona na kilka głównych części: Wstęp, następnie Klasyfikacja problemów i metod szeregowania zadań (Część I.), Własności i al-

gorytmy dla problemów przepływowych (Część II.), Zastosowanie metod w praktyce (Część III.) oraz Wnioski końcowe.

Część pierwsza zawiera wprowadzenie do teorii szeregowania zadań wraz z przeglądem literatury. Część została podzielona na dwa rozdziały. Rozdział 2. został poświęcony problemom szeregowania zadań, w szczególności problemom przepływowym. Przedstawiono modele matematyczne i grafowe oraz pokazano różne dodatkowe ograniczenia problemów, skupiając się na problemach ze sprzężeniami czasowymi, gdzie pojawiają się zależności między operacjami. W Rozdziale 3. skupiono się na algorytmach optymalizacyjnych stosowanych dla problemów szeregowania zadań. Przedstawiono schematy algorytmów dokładnych oraz przybliżonych najczęściej spotykanych w literaturze. Algorytmy przybliżone dodatkowo podzielono na trzy grupy: (1) konstrukcyjne, (2) lokalnego poszukiwania oraz (3) populacyjne.

W Części drugiej sformułowano nowe definicje problemów przepływowych szeregowania zadań ze sprzężeniami czasowymi oraz sformułowano dla nich szereg własności. Następnie zostały pokazane dostosowane lub utworzone nowe wersje algorytmów optymalizacyjnych. Dla każdego zaproponowanego problemu przeprowadzono badania eksperymentalne w celu potwierdzenia sformułowanych własności. Ta cześć również została podzielona na dwa główne rozdziały.

Rozdział 4. poświęcono problemom ze sprzężeniami czasowymi. W pierwszej kolejności sformułowano model matematyczny oraz grafowy dla permutacyjnego problemu przepływowego ze sprzężeniami czasowymi. Zidentyfikowano kilka własności, w tym efektywną metodę liczenia wartości funkcji celu. Dla problemu zaproponowano kilka algorytmów:(1) Metodę Podziału i Ograniczeń, (2) przeszukiwanie snopowe, (3) Przeszukiwanie z Zabronieniami oraz (3) Sztuczną Kolonię Pszczół. Następnie sformułowano model matematyczny oraz grafowy dla niepermutacyjnej wersji problemu. Zidentyfikowano nowe własności problemu, w szczególności własności eliminacyjne oparte na tzw. blokach. Zaproponowane zostały dwie metody rozwiązania: (1) Metoda Podziału i Ograniczeń oraz (2) Przeszukiwanie z Zabronieniami uwzględniające własności blokowe. Pokazano również równoległy sposób liczenia wartości funkcji celu za pomocą odpowiednio zaadaptowanego algorytmu Floyda-Warshalla.

Następny Rozdział 5. poświęcono problemom z przezbrojeniami. Najpierw sformułowano model matematyczny oraz grafowy dla permutacyjnego problemu przepływowego z ciągłą pracą maszyn oraz przezbrojeniami zależnymi od kolejności wykonywania zadań. Zidentyfikowano kilka własności, w tym metodę liczenia wartości funkcji celu oraz zdefiniowano dedykowane sąsiedztwo. W celu weryfikacji wyników przeprowadzono badania eks-

perymentalne z wykorzystaniem różnych algorytmów: (1) Przeszukiwania z Zabronieniami oraz (2) Algorytmu Genetycznego. Druga część rozdziału została poświęcona problemowi szeregowania rozłącznych przezbrojeń na przykładzie problemu przepływowego. Sformułowano model matematyczny oraz grafowy. Następie obliczono rozmiar przestrzeni rozwiązań dla dwóch maszyn oraz wykazano jego powiązanie z liczbami Catalana. Następnie zidentyfikowano własności eliminacyjne oparte na blokach. Zaprezentowano wyniki potwierdzone badaniami eksperymentalnymi z użyciem różnych metod: (1) Całkowitoliczbowego Programowania Liniowego, (2) dedykowanego algorytmu zachłannego dla dwóch i więcej maszyn oraz (3) Przeszukiwanie z Zabronieniami.

W Części III wyróżniono Rozdział 6. poświęcony trzem przykładom zastosowania opracowanych modeli oraz algorytmów optymalizacyjnych dla rzeczywistych problemów. Pierwszym opisanym przypadkiem jest rzeczywisty problem szeregowania zadań oraz rozłącznych przezbrojeń na przykładzie firmy produkującej urządzania gospodarstwa domowego. Następnie rozważono dwa przypadki problemów marszrutyzacji pojazdów, które zostały transformowane do problemów szeregowania zadań z przezbrojeniami. W pierwszej kolejności został rozpatrzony proces wyznaczania tras samochodów ciężarowych dla firmy logistycznej zajmującej się spedycją towarów na terenie Stanów Zjednoczonych Ameryki. Ostatnim rozważanym przypadkiem jest proces zarządzania flotą cystern w firmie zajmującej się odbiorem towarów o stałej strukturze odbiorców.

Rozprawę kończy podsumowanie oraz spis literatury liczący 146 pozycji. We wnioskach końcowych zawarto przegląd uzyskanych wyników oraz perspektywy dalszych badań nad rozszerzeniem sprzężeń czasowych i rozłącznych przezbrojeń na inne klasy problemów szeregowania zadań.

Część I

Klasyfikacja problemów i metod szeregowania zadań

Rozdział 2

Problemy szeregowania zadań

W tej części pracy skupiono się na przedstawieniu elementów teorii szeregowania zadań. Pokazano podstawowe pojęcia, niezbędne ograniczenia oraz modele grafowe, w tym matematyczne dla klasycznych problemów szeregowania zadań.

Elementarnymi pojęciami są: *zadanie*, *operacja* i *maszyna*. Poprzez zadanie rozumiemy proces lub zbiór procesów zwanych operacjami, które muszą być wykonane przez maszynę. Sposób działania maszyn lub zależności pomiędzy operacjami mogą być opisane poprzez dodatkowe ograniczenia, które zostaną szerzej opisane w Podrozdziale 2.3. W Podrozdziale 2.1 opisano różne sposoby przechodzenia zadań przez system. Ponadto w Podrozdziale 2.2 poświęcono skupiono się na problemach przepływowych. Na koniec opisano różne sposoby oceniania danego rozwiązania w zależności od kryterium optymalizacyjnego.

Niezbędnym elementem do rozwiązania wybranego problemu szeregowania zadań jest zbudowanie dla niego odpowiedniego modelu matematycznego. Przez $\mathcal{J} = \{1, 2, \ldots, n\}$ oznacymy zbiór n zadań oraz przez $\mathcal{M} = \{1, 2, \ldots, m\}$ zbiór m maszyn. Każde zadanie $i \in \mathcal{J}$ składa się z o_i operacji ze zbioru $\mathcal{O}_i = \{l_i+1,\ l_i+2,\ \ldots,\ l_i+o_i\}$. Przez $\mathcal{O}$ oznaczmy zbiór wszystkich operacji, w którego skład wchodzą wszystkie operacje z poszczególnych zadań:

$$\mathcal{O} = \bigcup_{i \in \mathcal{J}} \mathcal{O}_i. \tag{2.1}$$

Przez $|O|$ oznaczmy liczbę wszystkich operacji. Jeśli w ramach jednego zadania mamy więcej niż jedną operację ($o_i > 1$), w takim wypadku operacje muszą być wykonywane według ustalonej kolejności zwanej porządkiem

technologicznym, tzn. operacja $a+1$ w zadaniu i musi być wykonywana po zakończeniu operacji a z tego samego zadania. Czas trwania operacji i-tej ma maszynie a oznaczmy przez p_i^a. W literaturze często możemy spotkać się z oznaczeniem z dwoma oddzielonymi przecinkiem indeksami (maszynowym i zadaniowym) w indeksie dolnym. Przyjęta w niniejszej dysertacji notacja ma na celu poprawienie czytelności w dowodach matematycznych w późniejszych rozdziałach pracy. Symbole i i j będą używane w odniesieniu do zadań. Podobnie, a i b będą używane w odniesieniu do maszyn oraz k i l do przezbrojeń.

Rozwiązaniem dowolnego problemu szeregowania zadań jest harmonogram $(\boldsymbol{S}, \boldsymbol{C})$, który składa się z dwuwymiarowego wektora momentów rozpoczęcia wykonywania wszystkich operacji $\boldsymbol{S}$ oraz dwuwymiarowego wektora momentów zakończenia wykonywania wszystkich operacji $\boldsymbol{C}$. Dla danego harmonogramu $(\boldsymbol{S}, \boldsymbol{C})$ można wyznaczyć jednoznaczną kolejność wykonywania zadań π.

Teoria szeregowania zadań jest dynamicznie rozwijająca się dziedziną, dlatego w celu jej uporządkowania w roku 1979 Graham i inni [54] zaproponowali reprezentację trójpolową:

$$\alpha|\beta|\gamma, \tag{2.2}$$

gdzie przyjmuje się jako α – typ problemu (Podrozdział 2.1), β – dodatkowe niestandardowe ograniczenia problemu (Podrozdział 2.3), γ – postać funkcji celu (Podrozdział 2.4). Przykłady użycia notacji: (1) $1||\sum w_i T_i$, (2) $F||C_{\max}$ lub (3) $J3|ST_{sd}|STS$. Notacja jest na bieżąco rozszerzana poprzez różne artykuły przeglądowe. W pracy [27] autor dokonał przeglądu różnych prac dotyczących szeregowania zadań, z kolei w [3] skupiono się na problemach z przezbrojeniami oraz w innych pracach dotyczących konkretnych problemów jednomaszynowych [75], przepływowych [43], niepermutacyjnych przepływowych [109], hybrydowych przepływowych [112] czy gniazdowych [120].

2.1 Rodzaje problemów szeregowania zadań

Smutnicki [121] zaproponował podział pola α dotyczącego opisu problemu na ciąg trzech symboli $\alpha_3\alpha_2\alpha_1$. Przez symbol α_1 oznaczamy liczbę maszyn w opisywanym systemie. Wartość może zostać pominięta, wtedy przyjmujemy dowolną liczbę maszyn m. Dla problemów jednomaszynowych (*single machine scheduling problem*, SMSP) [55] przyjmujemy $\alpha_1 = 1$ oraz pozostałe symbole pozostawiamy puste, ponieważ nie ma dodatkowych relacji

technologicznych jeśli liczba operacji w każdym zdaniu wynosi $o_i = 1$ oraz suma wszystkich operacji jest równa liczbie zadań $o = n$.

Symbol α_2 oznacza typ problemu opisywanego systemu oraz determinuje sposób przechodzenia zadań przez niego zadań. W literaturze najczęściej rozpatruje się:

- FP – permutacyjny problem przepływowy szeregowania zadań (*Permutation Flow Shop Scheduling Problem*, PFSSP) [99], w którym liczba operacji w każdym zadaniu jest równa liczbie maszyn, każda operacja z pojedynczego zadania musi być wykonana na dedykowanej maszynie według ustalonej kolejności zadań dla wszystkich maszyn,
- F – niepermutacyjny problem przepływowy szeregowania zadań, często nazywany w skrócie problemem przepływowym (*Flow Shop Scheduling Problem*, FSSP) [58], w którym założenie są zbliżone do FSSP, jednak dopuszcza się wykonywania zadań w różnych kolejnościach na każdej maszynie,
- J – problem gniazdowy szeregowania zadań (*Job Shop Scheduling Problem*, JSSP) [86], w odróżnieniu od FSSP i PFSSP zadania mogą mieć różną liczbę operacji oraz inną kolejność technologiczną,
- FF – elastyczny problem przepływowy szeregowania zadań (*Flexible Flow Shop Scheduling Problem*, FFSSP) [82], FSSP w którym część operacji może być wykonywana na różnych maszynach,
- FJ – elastyczny problem gniazdowy szeregowania zadań (*Flexible Job Shop Scheduling Problem*, FJSSP) [103], JSSP w którym część operacji może być wykonywana na różnych maszynach,
- HF – hybrydowy problem przepływowy szeregowania zadań (*Hybird Flow Shop Scheduling Problem*, HFSSP) [112], w przeciwieństwie do FSSP zadanie musi przejść przez wszystkie gniazda produkcyjne, które składają się z maszyn do wykonywania zadań,
- I – problem szeregowania zadań z równoległymi maszynami (*Parallel Machine Scheduling Problem*, PMSP), podobnie jak w problemie SMSP każde zadanie ma tylko jedną operację, ale w tym przypadku można wykonać ją na jednej z maszyn równoległych,
- G – problem ogólny szeregowania zadań (*General Shop Scheduling Problem*, GSSP) [136], zależności technologiczne między operacjami są opisane grafem,
- O – problem otwarty szeregowania zadań (*Open Shop Scheduling Problem*, OSSP) [119], operacje są wykonywanie na dedykowanych maszynach, ale mogą być wykonane w dowolnej kolejności.

Symbol α_3 definiuje typ maszyn–sposób wykonywania operacji na maszynach. W przypadku jeśli każda operacją ma dedykowaną (przypisaną)

maszynę na której może zostać wykonana, to symbol pozostaje pusty. Jeśli operacja może być wykonana na więcej niż jednej maszynie to możemy wyróżnić następujące typy:

P – jednakowe maszyny równoległe (*identical parallel machines*),
Q – jednorodne maszyny równoległe (*uniform parallel machines*),
R – niejednorodne maszyny równoległe (*unrelated parallel machines*).

Powyżej przedstawiono najczęściej spotykane rodzaje problemów szeregowania zadań w literaturze. W dalszej części pracy skupiono się głównie na badaniu oraz formułowaniu własności dla różnych wariantów problemów przepływowych, dlatego w Podrozdziale 2.2 zostanie przedstawiony szczegółowy model matematyczny oraz grafowy dla FSSP i PFSSP.

2.2 Problemy przepływowe

W klasycznych problemach przepływowych permutacyjnych i niepermutacyjnych liczba operacji w każdym zadaniu jest równa liczbie maszyn w systemie ($o_i = m$). Zatem liczba wszystkich operacji wynosi $o = nm$. Każde zadanie i musi zostać wykonane na maszynach w naturalnej kolejności: $1 \to 2 \to \cdots \to m$, zwanej marszrutą technologiczną. Operacje nie mogą być przerywane. W danym czasie może być wykonywana tylko jedna operacja na każdej maszynie.

Harmonogram $(\boldsymbol{S}, \boldsymbol{C})$ reprezentujący czas pracy maszyn ma następującą postać: momenty rozpoczęcia wykonywania zadań są opisane przez $\boldsymbol{S} = [S_i^a]^{m \times n}$, gdzie moment rozpoczęcia wykonywania S_i^a dotyczy operacji i-tej na maszynie a, oraz momenty zakończenia wykonywania zadań są opisane przez $\boldsymbol{C} = [C_i^a]^{m \times n}$, gdzie C_i^a to moment zakończenia wykonywania operacji i-tej na maszynie a.

Permutacyjny problem przepływowy

Dla PFSSP rozwiązanie jest reprezentowane jednoznacznie przez taką samą kolejność zadań na wszystkich maszynach $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, która jest permutacją zbioru zadań $\mathcal{J}$. Niech

$$\pi \in \Pi, \tag{2.3}$$

gdzie Π to zbiór wszystkich możliwych permutacji zbioru $\mathcal{J}$. Dla danej permutacji możemy utworzyć dopuszczalny harmonogram (S, C), który musi

spełniać następujące ograniczenia:

$$\forall a \in \mathcal{M} \;\; \forall i \in \mathcal{J} \quad C^a_{\pi(i)} = S^a_{\pi(i)} + p^a_{\pi(i)}, \tag{2.4}$$

$$\forall a \in \mathcal{M} \;\; \forall i \in \mathcal{J} \setminus \{1\} \quad S^a_{\pi(i)} \geqslant C^a_{\pi(i-1)}, \tag{2.5}$$

$$\forall a \in \mathcal{M} \setminus \{1\} \;\; \forall i \in \mathcal{J} \quad S^a_{\pi(i)} \geqslant C^{a-1}_{\pi(i)}. \tag{2.6}$$

Nie tracąc na ogólności można założyć, że pierwsza operacja na maszynie pierwszej rozpoczyna się w momencie czasowym $S^1_1 = 0$. Ograniczenie (2.4) zapewnia ciągłość wykonywania operacji. Ograniczenie (2.5) wymusza rozpoczęcie wykonywania kolejnej operacji na danej maszynie dopiero po zakończeniu wykonywania poprzedniej operacji. Podobnie ograniczenie (2.6) zapewnia, że operacja w ramach tego samego zadania rozpocznie się wykonywać dopiero po zakończeniu wykonywania operacji na poprzedniej maszynie (poprzednika technologicznego). Harmonogram nazwiemy dosuniętym w lewo (*left-shifted*), jeśli żadne operacje nie mogą rozpocząć wykonywać się wcześniej bez naruszenia ograniczeń lub zmiany kolejności przetwarzania.

Jedną z klasycznych technik szeregowania zadań jest budowa modelu grafowego dla danego rozwiązania π [95]. Należy utworzyć graf $G(\pi) = (V,\ E_1 \cup E_2)$, gdzie:

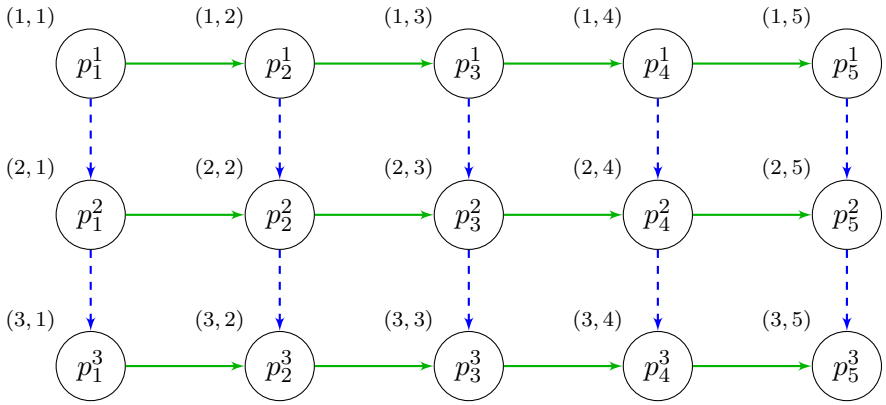
$$V = \bigcup_{\substack{a \in \mathcal{M} \\ i \in \mathcal{J}}} \Big\{ (a, i) \Big\}. \tag{2.7}$$

Przez V został oznaczony zbiór wierzchołków reprezentujących operacje (dla uproszczenia wizualizacji ustawione w formie siatki). Operacja z zadania i-tego na maszynie a jest reprezentowana przez obciążony wierzchołek w i-tej kolumnie oraz a-tym wierszu oznaczony parą (a, i) z wagą p^a_i.

$$E_1 = \bigcup_{\substack{a \in \mathcal{M} \\ i \in \mathcal{J} \setminus \{n\}}} \Big\{ \Big((a, i),\ (a, i+1) \Big) \Big\}, \tag{2.8}$$

$$E_2 = \bigcup_{\substack{a \in \mathcal{M} \setminus \{m\} \\ i \in \mathcal{J}}} \Big\{ \Big((a, i),\ (a+1, i) \Big) \Big\}. \tag{2.9}$$

Łuki grafu $\mathcal{G}(\pi)$ reprezentują ograniczenia problemu: (1) poziome E_1 – ograniczenia maszynowe oraz (2) pionowe E_2 – ograniczenia technologiczne. Długość najdłużej ścieżki dochodzącej do danego wierzchołka (a, i) jest równa odpowiedniemu czasowi zakończenia wykonywania C^a_i. Strukturę grafu $G(\pi)$ dla problemu o rozmiarze 5×3 oraz kolejności $\pi = (1, 2, 3, 4, 5)$ przedstawiono na Rysunku 2.1. Dla ułatwienia czytelności łuki E_1 i E_2 oznaczono odpowiednio zieloną linią ciągła i niebieską przerywaną.



Rysunek 2.1: Graf $\mathcal{G}(\pi)$ dla problemu PFSSP dla permutacji π

Tablica 2.1: Instancja problemu PFSSP o rozmiarze $n = 5$ na $m = 3$

i	p_i^1	p_i^2	p_i^3
1	1	2	1
2	3	3	3
3	3	2	2
4	2	1	6
5	1	4	1

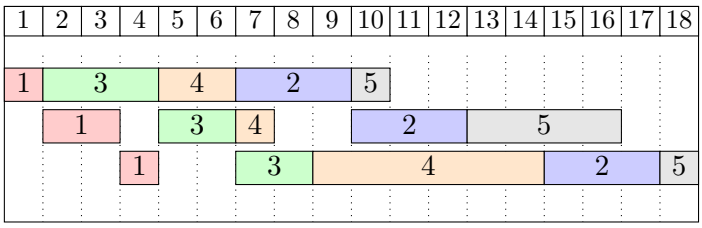
Przykład 2.1 Rozważmy instancję problemu PFSSP o rozmiarze $n = 5$ oraz $m = 3$ z Tabeli 2.1. Na Rysunku 2.2 przedstawiono schemat Gantta z dosuniętym w lewo harmonogramem dla $\pi = (1, 3, 4, 2, 5)$. □

Niepermutacyjny problem przepływowy

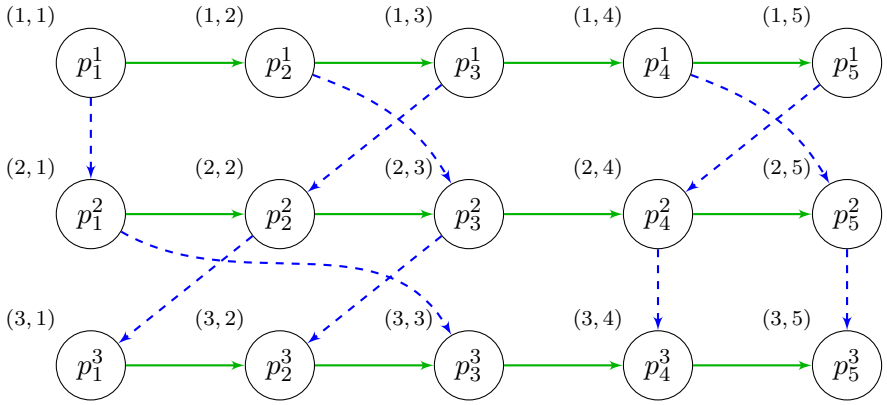
W problemie FSSP z harmonogramem $(\boldsymbol{S}, \boldsymbol{C})$ mamy powiązaną sekwencje permutacji wykonywania zadań dla każdej maszyny: $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_m)$, gdzie $\pi_a = (\pi_a(1), \pi_a(2), \ldots, \pi_a(n))$. Niech $\pi_a(j)$ oznacza operację z zadania j-tego w kolejności na maszynie a. Dopuszczalny harmonogram musi spełniać te same ograniczenia co dla PFSSP: o ciągłości operacji (2.4), maszynowe (2.5) oraz nowe ograniczenie technologiczne:

$$\forall a \in \mathcal{M} \setminus \{1\} \quad \forall i \in \mathcal{J} \quad S^a_{\pi(i)} \geqslant C^{a-1}_{\pi^{-1}_{a-1}(\pi_a(i))}, \tag{2.10}$$

gdzie funkcja odwrotna $\pi_a^{-1}(i)$ zwraca pozycje operacji z zadania i w π_a dla maszyny a, tj.: $\pi_a^{-1}(i) = j \iff \pi_a(j) = i$.



Rysunek 2.2: Harmonogram dla instancji PFSSP z Przykładu 2.1

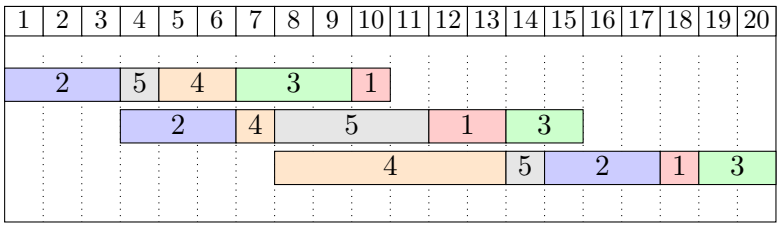


Rysunek 2.3: Przykładowy graf $\mathcal{G}(\boldsymbol{\pi})$ dla problemu FSSP

Podobnie jak dla FSSP możemy zbudować graf rozwiązania $\mathcal{G}(\boldsymbol{\pi})$, na tej samej zasadzie utworzymy wierzchołki V (2.7) oraz łuki poziome E_1 (2.8). Dla zbioru łuków pionowych musimy uwzględnić różne kolejności wykonywania zadań na maszynach:

$$E_2 = \bigcup_{\substack{a \in \mathcal{M} \setminus \{1\} \\ i \in \mathcal{J}}} \left\{ \Big((a-1, \pi_{a-1}^{-1}(\pi_a(i))),\ (a, i) \Big) \right\}. \tag{2.11}$$

Przykład 2.2 Rozważmy instancję FSSP dla $n = 5$ i $m = 3$ z Tabeli 2.1. Na Rysunku 2.4 przedstawiono schemat Gantta z dosuniętym w lewo harmonogramem. Na Rysunku 2.3 pokazano przykładową strukturę grafu $\mathcal{G}(\boldsymbol{\pi})$ dla $\boldsymbol{\pi} = ((2, 5, 4, 3, 1),\ (2, 4, 5, 1, 3),\ (4, 5, 2, 1, 3))$.



Rysunek 2.4: Harmonogram dla instancji FSSP z przykładu 2.2

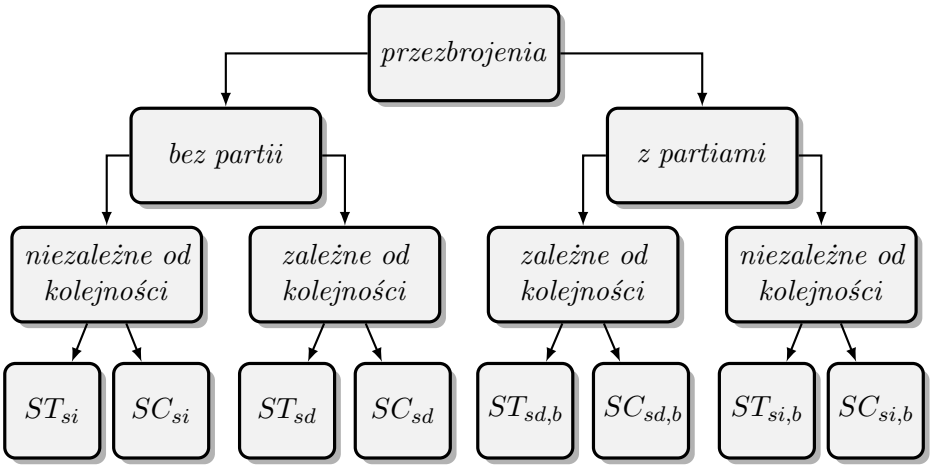
2.3 Dodatkowe ograniczenia problemów

Pole β określa dodatkowe parametry oraz ograniczenia opisywanego problemu. Poniżej skupiono się na przedstawieniu ograniczeń zależnych między operacjami, nazywanymi dalej `sprzężeniami czasowymi` (*time couplings*), co wynika z podjętej tematyki niniejszej pracy.

Przezbrojenia

Najczęściej badanym dodatkowym ograniczaniem jest wprowadzenie przezbrojeń (*setups*) dla różnych wariantów klasycznych problemów szeregowano zadań, jak PFSSP, FSSP, JSSP, OSSP, PMPS oraz innych. Ze względu na rzeczywistą potrzebę przygotowania, czyszczenia czy zmiany konfiguracji maszyny między operacjami, przezbrojenia stały się powszechnie stosowną koncepcją w teorii szeregowania zadań. Już na początku lat 60 ubiegłego wieku rozważano przezbrojenia dla problemów produkcyjnych, w pracy [64] przedstawiono algorytm dokładny dla dwu-maszynowego PFSSP. W przeglądzie [3] podkreślono znaczenie przezbrojeń w procesie modelowania problemów szeregowania zadań. Autorzy zwracają uwagę, że koszt przezbrojenia nie jest równy jego czasowi, chociaż przyjęło się stosować takie przybliżenie. Powstały różne definicje czasów przezbrojeń, jednak najczęściej spotykaną klasyfikacją obejmującą zakres wszystkich problemów szeregowania zadań, takich jak GSSP [106], JSSP [120] czy innych pokrewnych [146], jest podział na:

- ST_{si} – czas przezbrojenia zależny od następnej i poprzedniej operacji (*sequence independent setup time*),
- SC_{si} – koszt przezbrojenia zależny od następnej poprzedniej operacji (*sequence independent setup cost*),
- ST_{sd} – czas przezbrojenia zależny tylko od następnej operacji (*sequence dependent setup time*),
- SC_{sd} – koszt przezbrojenia zależny tylko od następnej operacji (*sequence dependent setup cost*),



Rysunek 2.5: Klasyfikacja przezbrojeń

- $ST_{si,b}$ – czas przezbrojenia zależny od następnej i poprzedniej operacji w partiach zadań (*sequence-independent batch setup time*),
- $SC_{si,b}$ – koszt przezbrojenia zależny od następnej i poprzedniej operacji w partiach zadań (*sequence-independent batch setup cost*),
- $ST_{sd,b}$ – czas przezbrojenia zależny tylko od następnej operacji dla partii zadań (*sequence-dependent batch setup time*),
- $SC_{sd,b}$ – koszt przezbrojenia zależny tylko od następnej operacji dla partii zadań (*sequence-dependent batch setup cost*).

Czas przezbrojenia może być stały, zależny tylko od następnej operacji na tej maszynie lub zależny zarówno od następnej jak i poprzedniej operacji. Często rozróżnia się również rodziny lub partie zadań (*batch*). Zadania w ramach partii lub rodziny są tego samego typu. W takim przypadku możemy rozróżnić dwa rodzaje przezbrojeń: (1) „małe" lub nawet zerowe (między zadaniami jednego typu), (2) „duże" (między zadaniami różnego typu). Można spotkać się z podejściem ignorowania "małych" przezbrojeń z powodu ich mniejszego wpływu na rozwiązanie.

Przyjmijmy, że koszt przezbrojeń jest równy czasowi trwania przezbrojenia. Przez $s^a_{i,j}$ oznaczymy czas trwania przezbrojenia na maszynie a po operacji z i-tego zadania oraz przed operacja z j-tego zadana na tej samej maszynie (w przypadku przezbrojeń zależnych od kolejności zadań). Dla przezbrojeń niezależnych od kolejności zadań wszystkie wartości dla przezbrojeń po operacji na konkretnej maszynie wynosi tyle samo (niezależnie od kolejnego zadania):

$$\forall a \in \mathcal{M} \quad \forall i,j,k \in \mathcal{J} \quad s^a_{i,j} = s^a_{i,k}. \tag{2.12}$$

i	$s^1_{i,1}$	$s^1_{i,2}$	$s^1_{i,3}$	$s^1_{i,4}$	$s^1_{i,5}$	$s^2_{i,1}$	$s^2_{i,2}$	$s^2_{i,3}$	$s^2_{i,4}$	$s^2_{i,5}$	$s^3_{i,1}$	$s^3_{i,2}$	$s^3_{i,3}$	$s^3_{i,4}$	$s^3_{i,5}$
1	0	2	**2**	1	1	0	2	**2**	1	1	0	2	**2**	1	2
2	3	0	1	2	**2**	3	0	2	1	**2**	3	0	1	2	**1**
3	3	2	0	**1**	1	3	2	0	**3**	1	3	2	0	**2**	1
4	2	**1**	1	0	3	2	**1**	1	0	1	2	**1**	1	0	1
5	1	4	3	1	0	1	4	1	1	0	3	4	1	2	0

Tablica 2.2: Macierze czasów przezbrojeń między operacjami

Podstawowy harmonogram $(\boldsymbol{S}, \boldsymbol{C})$ reprezentujący czas pracy należy uzupełnić o: (1) momenty rozpoczęcia wykonywania przezbrojeń opisanych przez macierz ${}^{\sigma}\boldsymbol{S} = [{}^{\sigma}S^a_i]^{m\times n-1}$ oraz (2) momenty zakończenia wykonywania przezbrojeń opisanych przez macierz ${}^{\sigma}\boldsymbol{C} = [{}^{\sigma}C^a_i]^{m\times n-1}$, gdzie moment ${}^{\sigma}S^a_i$ oraz ${}^{\sigma}C^a_i$ dotyczą przezbrojenia na maszynie a po operacji i-tej w kolejności. Dopuszczalny harmonogram musi spełniać ograniczenie o ciągłości zadań (2.4) oraz ograniczenie technologiczne (2.6), jak dla podstawowego PFSSP, oraz ograniczenia wynikające z wprowadzenia przezbrojeń:

$$\forall a \in \mathcal{M} \quad \forall i \in \mathcal{J} \setminus \{1\} \quad {}^{\sigma}C^a_{\pi(i)} = {}^{\sigma}S^a_{\pi(i)} + s^a_{\pi(i),\pi(i+1)}, \tag{2.13}$$

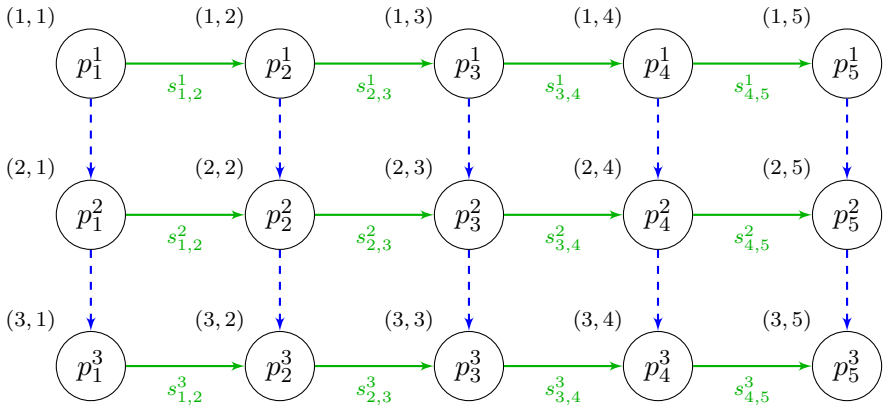
$$\forall a \in \mathcal{M} \quad \forall i \in \mathcal{J} \setminus \{1\} \quad {}^{\sigma}S^a_{\pi(i)} \geqslant C^a_{\pi(i)}, \tag{2.14}$$

$$\forall a \in \mathcal{M} \quad \forall i \in \mathcal{J} \setminus \{1\} \quad S^a_{\pi(i+1)} \geqslant {}^{\sigma}C^a_{\pi(i)}, \tag{2.15}$$

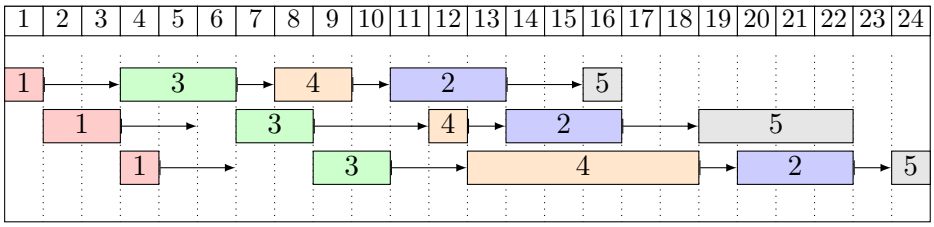
Ograniczenie (2.13) zapewnia ciągłość wykonywania przezbrojenia. Ograniczenie (2.14) wymusza rozpoczęcie wykonywania przezbrojenia na danej maszynie dopiero po zakończeniu wykonywania poprzedzającej operacji oraz analogicznie (2.15) wymusza rozpoczęcie wykonywania kolejnej operacji dopiero po zakończeniu wykonywania przezbrojenia. Jeśli nie mamy dodatkowych ograniczeń problemu to równania (2.5), (2.14) oraz (2.15) możemy zredukować do jednego:

$$\forall a \in \mathcal{M} \quad \forall i \in \mathcal{J} \setminus \{1\} \quad S^a_{\pi(i))} \geqslant C^a_{\pi(i-1)} + s^a_{\pi(i),\pi(i+1)}. \tag{2.16}$$

Przykład 2.3 Rozważmy instancję PFSSP z przezbrojeniami dla $n = 5$ oraz $m = 3$, dla czasów wykonywania z Tabeli 2.1. Następnie należy rozszerzyć instancję o macierze przezbrojeń dla każdej maszyny, które przedstawiono w Tabli 2.2. Na modelu grafowym trzeba uwzględnić wagi na łukach poziomych (maszynowych) E_2 w zależności od operacji po oraz przed którą występuje w kolejności wykonywania na konkretnej maszynie. Strukturę



Rysunek 2.6: Graf $\mathcal{G}(\pi)$ dla problemu PFSSP z przezbrojeniami typu ST_{sd}



Rysunek 2.7: Harmonogram dla PFSSP z przezbrojeniami z przykładu 2.3

grafu $\mathcal{G}(\pi)$ rozszerzonego o przezbrojenia pokazano na Rysunku 2.6. Na rysunku 2.7 przedstawiono schemat Gantta z dosuniętym w lewo harmonogramem dla PFSSP z przezbrojeniami dla $\pi = (1, 3, 4, 2, 5)$. Potrzebne czasy przezbrojeń zaznaczano pogrubioną czcionką w Tabeli 2.2. □

W literaturze można spotkać się z przezbrojeniami rozłącznymi, znanymi częściej jako przezbrojenia z wykorzystaniem jednego serwera (*setups and single server*). Z powodu ograniczonych zasobów w danym czasie może być wykonywane przezbrojenie tylko na jednej maszynie.

Autorzy zazwyczaj skupiają się na omówieniu szczególnych przypadków dla FSSP. W pracy [25] autorzy skupili się na pokazaniu algorytmów dla przypadków z identycznymi lub jednostkowymi czasami wykonywania operacji oraz przezbrojeń. W [83] zaproponowano kilka własności dla problemu dwumaszynowego (identyczne czasy wykonywania operacji, krótkie operacje itp.). Poza FSSP [115, 128] można również spotkać w literaturze inne problemy z rozłącznymi przezbrojeniami, np.: (1) szeregowanie zadań na równoległych maszynach [137], (2) szeregowanie procesów o ograniczonych

zasobach [98] lub (3) dobór wielości partii [133]. W dalszej części pracy rozłączne przezbrojenia zostaną poddane szczegółowej analizie. Następnie zostanie zaproponowanych kilka nowych własności.

Ciągła praca maszyn

Innym rozpatrywanym bardzo często w literaturze ograniczaniem jest ciągła praca maszyn `bez przestojów` (*no idle*) [101, 145]. Problem ma silne podłoże praktyczne, ponieważ w wielu branżach przestój maszyny generuje koszty. Interesującym przykładem jest branża budowlana, gdzie w systemie potokowym odpowiednikiem maszyn są fronty robót (wykopy, fundamentowanie, stawienie ścian, itp.) a odpowiednikiem zadań obiekty budowlane. Każda operacja w ramach jednego zadania (budowy) musi przejść przez kolejne maszyny (fronty budowlane/ekipy). Bardzo często minimalizuje się przerwy między operacjami na tej samej maszynie [10], gdzie szczególnym przypadkiem jest brak przerw – ciągła praca maszyn. W celu zapewnia pracy bez przestojów oprócz standardowych ograniczeń dla PFSSP (2.4)-(2.6) należy uwzględnić dodatkowo równanie:

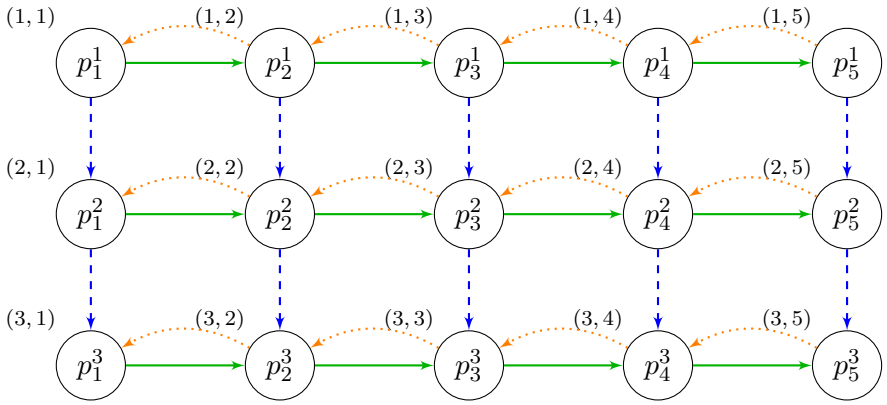
$$\forall a \in \mathcal{M} \quad \forall i \in \mathcal{J} \setminus \{1\} \quad S^a_{\pi(i)} = C^a_{\pi(i-1)}, \tag{2.17}$$

Przykładowy harmonogram dosunięty w lewo dla PFFSP z ciągła pracą maszyn o rozmiarze 5×3 dla permutacji $\pi = (1, 2, 3, 4, 5)$ przedstawiono za pomocą schematu Gantta na Rysunku 2.9. Model grafowy rozszerzony o łuki „powrotne" (*return arcs*, zaznaczone czerwoną oraz wykropkowaną linią) pokazano na Rysunku 2.8. Łuki „powrotne" mają zagwarantować ciągłość wykonywania pracy na maszynach bez przestojów. Takie łuki możemy opisać zbiorem:

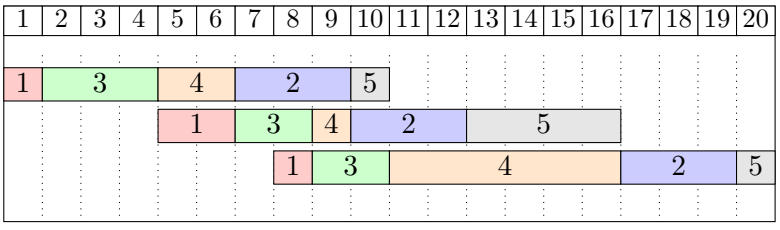
$$E_3 = \bigcup_{\substack{a \in \mathcal{M} \\ i \in \mathcal{J} \setminus \{1\}}} \Big\{ \big((a, i),\ (a, i-1) \big) \Big\}. \tag{2.18}$$

Ciągłość wykonywania zadań

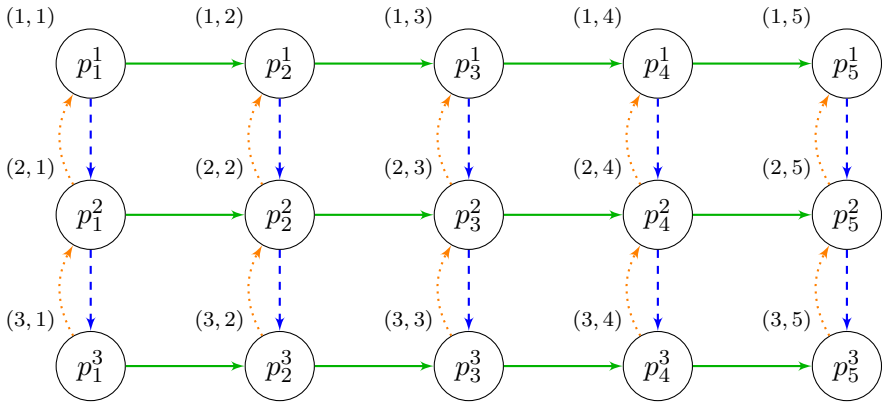
W procesach hutniczych możemy się spotkać z inną sytuacją. Kiedy należy zachować ciągłość wykonywania operacji między operacjami w ramach zadań. Analogicznie do ograniczenia *bez przestojów*. Przypadek kiedy następna operacja wewnątrz zadania musi być wykonania niezwłocznie po poprzedniej w literaturze nazywane jest ograniczeniem `bez czekania` (*no*



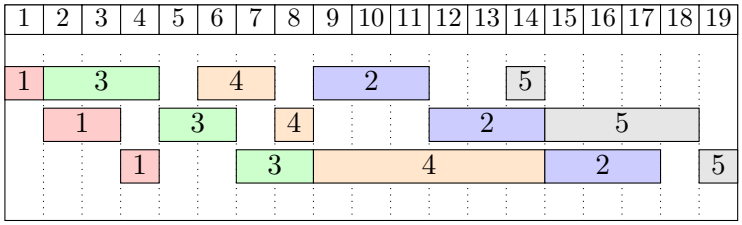
Rysunek 2.8: Graf $\mathcal{G}(\pi)$ dla problemu PFSSP bez przestojów



Rysunek 2.9: Harmonogram dla instancji PFSSP bez przestojów



Rysunek 2.10: graf $\mathcal{G}(\pi)$ dla problemu PFSSP bez czekania



Rysunek 2.11: Harmonogram dla instancji PFSSP bez czekania

wait) [38, 141]. Analogicznie jak dla poprzedniego ograniczenia musimy dla podstawowego PFSSP wprowadzić dodatkowe równianie:

$$\forall a \in \mathcal{M} \setminus \{1\} \quad \forall i \in \mathcal{J} \quad S^a_{\pi(i)} = C^{a-1}_{\pi(i)}, \tag{2.19}$$

Przykładowy harmonogram dosunięty w lewo dla PFFSP z ciągłością wykonywania zadań o rozmiarze 5×3 dla permutacji $\pi = (1, 2, 3, 4, 5)$ oraz instancji z Tabeli 2.1 przedstawiono przy użyciu schematu Gantta na Rysunku 2.11. Rozszerzony model grafowy o łuki „powrotne pionowe" (*vertical return arc*, zaznaczone czerwoną oraz wykropkowaną linią) pokazano na Rysunku 2.10. Łuki „powrotne pionowe" mają zagwarantować ciągłość wykonywania operacji w ramach każdego zadania bez czekania. Wspomniane łuki możemy opisać zbiorem:

$$E_4 = \bigcup_{\substack{a \in \mathcal{M} \setminus \{1\} \\ i \in \mathcal{J}}} \Big\{ \big((a, i),\ (a-1, i) \big) \Big\}. \tag{2.20}$$

Inne ograniczenia

Jeśli w rozpatrywanym wariancie problemu nie ma dodatkowych ograniczeń technologicznych, wtedy pole β pozostawiamy puste. W literaturze możemy się spotkać również z między innymi takimi parametrami jak:

- r_i – czas dostępności (*ready time*) [84], każde zadanie musi być najpierw dostarczone lub przygotowane zanim będzie mogło być przetworzone w systemie $r_i \leqslant C_i^1$,
- q_i – czas dostarczenia (*quit time*) [102], każde zadanie potrzebuje dodatkowego czas na opuszczenie systemu po zakończeniu wykonywania na ostatniej maszynie, niezależnie od innych zadań,
- d_i – żądany czas zakończenia wykonywania operacji (*due date*) [111], każde zadanie musi się zakończyć przed swoim żądanym czasem zakończenia wykonywania: $C_i^m \leqslant d_i$, w przeciwnym wypadku zostania naliczona proporcjonalna kara ze współczynnikiem w_i,
- $t_i^{a,b}$ – czas transportu (*transportation time*) [123], pomiędzy operacjami w ramach zadania jest wymagany transport przy użyciu dodatkowego sprzętu, np.: wózka AGV (*Automated Guided Vehicle*).
- `pmtn` – przerywać (*preemptive*) [61], dopuszcza się możliwość przerywania operacji oraz wznowienia operacji od momentu przerwania,
- `prec` – istnieje narzucony częściowy porządek technologiczny wykonywania zadań,
- `no store` – bez magazynowania [117], brak możliwości składowania zasobów na stanowisku między wykonywaniem operacji.

W rozdziale przedstawiono najczęściej spotykane dodatkowe parametry lub ograniczenia dla problemów szeregowania zadań. W Części II dysertacji będą proponowane nowe parametry oraz nowe ograniczenia dla problemów przepływowych ze sprzężeniami czasowymi.

2.4 Funkcje celu

Dla problemów szeregowania zadań ostatnie pole γ w notacji Grahama definiuje postać funkcji celu, zwanej również kryterium optymalizacyjnym. W praktyce spotykamy się z dwoma klasami funkcji celu, maksymalizacyjnym

$$f_{\max} = \max_{i \in \mathcal{J}} f_i(C_i^m) \tag{2.21}$$

oraz sumacyjnym (addytywnym)

$$\sum f_i = \sum_{i \in \mathcal{J}} f_i(C_i^m), \tag{2.22}$$

gdzie f_i jest funkcją związaną z każdym zadaniem i oraz zależną od czasu zakończenia wykonywania tego zadania.

W literaturze najczęściej spotykanym kryterium optymalizacyjnym jest kryterium minimalizacji terminu zakończenia wykonywania wszystkich zadań $C_{\max}$ – czyli długość uszeregowania (*makespan*). W praktyce termin ten określa moment najpóźniej zakończonego zadania:

$$C_{\max} = \max_{i \in \mathcal{J}} C_i. \tag{2.23}$$

gdzie $C_i = C_i^m$ jest momentem opuszczenia systemu przez i-te zadanie, czyli momentem zakończenia wykonywania zadania i na ostatniej m-tej maszynie. Dla problemu PFSSP – $FP||C_{\max}$ po uwzględnieniu zarówno ograniczeń maszynowych i technologicznych, do wyznaczenia momentów zakończenia wykonywania operacji C_i^a możemy użyć wzoru rekurencyjnego:

$$C^a_{\pi(i)} = \begin{cases} p^a_{\pi(i)} & \text{dla } \ a = 1 \wedge i = 1, \\ C^a_{\pi(i)} + p^a_{\pi(i)} & \text{dla } \ a = 1 \wedge i > 1, \\ C^{a-1}_{\pi(i)} + p^a_{\pi(i)} & \text{dla } \ a > 1 \wedge i = 1, \\ \max\{C^{a-1}_{\pi(i)}, C^a_{\pi(i-1)}\} + p^a_{\pi(i)} & \text{w przeciwnym wypadku.} \end{cases} \tag{2.24}$$

Moment zakończenia wykonywania ostatniej operacji z ostatniego zadania będzie największy, więc będzie równy długości uszeregowania:

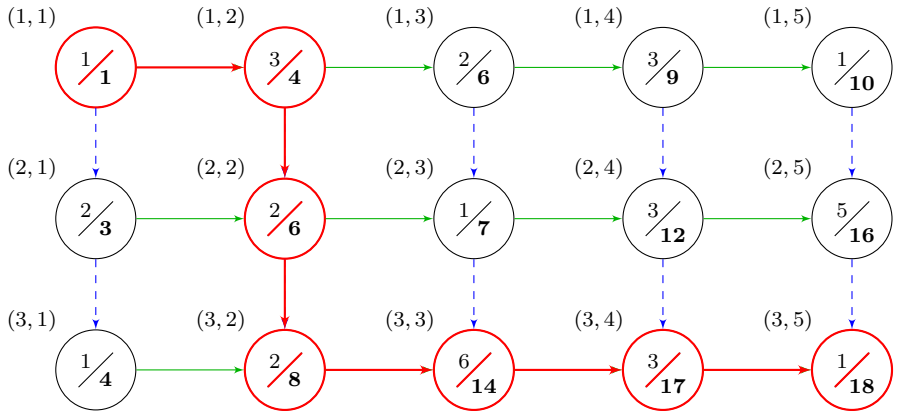
$$C_{\max} = C_n^m. \tag{2.25}$$

Złożoność obliczeniowa metody opisanej wzorem (2.24) wynosi $O(mn)$. Wyznaczone momenty zakończenia wykonywania operacji odpowiadają długościom ścieżek w grafie $\mathcal{G}(\pi)$: długość najdłuższej ścieżki dochodzącej do wierzchołka (a, i) wynosi C_i^a. Długość uszeregowania $C_{\max}$ jest równoważna z najdłuższą ścieżką w całym grafie $\mathcal{G}(\pi)$, którą będziemy nazywać ścieżką krytyczną. Na Rysunku 2.12 przedstawiono graf $\mathcal{G}(\pi)$ uzupełniony dodatkowo o długości ścieżek o wartości C_i^a w danym wierzchołku, oznaczone pogrubioną czcionką. Czerwony kolorem zaznaczono ścieżkę krytyczną. Analizując wzory (2.25) i (2.24) można stwierdzić, że ścieżka krytyczna dla badanego problemu będzie zawsze rozpoczynać się w wierzchołku $(1, 1)$ oraz kończyć w (m, n). Rozwiązaniem problemu jest wyznaczenie permutacji π^* dla której długość uszeregowania będzie jak najmniejsza:

$$C_{\max}(\pi^*) = \min_{\pi \in \Pi} C_{\max}(\pi). \tag{2.26}$$

Pole γ podobnie jak parametr α nie może pozostać puste.

Innymi często badanymi kryteriami w literaturze są:



Rysunek 2.12: Ścieżka krytyczna dla klasycznego PFSSP z Przykładu 2.1

T – czas cyklu (*cycle time*) [1],
$\sum C_i$ – suma czasów zakończenia wykonywania zadań (*total completion time*) [2],
$\sum w_i C_i$ – ważona suma czasów zakończenia wykonywania zadań (*total weighted completion time*) [130],
T_{max} – maksymalne spóźnienie zadań(*maximum tardiness*) [26],
$\sum T_i$ – suma spóźnień zadań(*total tardiness*) [108],
$\sum w_i T_i$ – ważona suma spóźnień zadań(*total weighted tardiness*) [142],
TSC – całkowity koszt wykonywania przezbrojeń (*total setup cost*) [78],
TST – całkowity czas wykonywania przezbrojeń (*total setup time*) [140],
TNS – całkowita liczba przezbrojeń (*total number of setups*) [100].

W praktyce najczęściej rozważane są kryteria minimalizacji maksymalnego czasu $C_{\max}$, sumy czasów $\sum C_i$, minimalnego czasu cyklu T oraz ważonej sumy spóźnień $\sum w_i T_i$. Dla $m > 3$ wszystkie warianty PFSSP są silnie NP-trudne [45].

2.5 Wnioski i uwagi

W rozdziale przedstawiono wprowadzenie do teorii szeregowania zadań, a następnie zaprezentowano szczegółową klasyfikację problemów szeregowania zadań z uwzględnieniem notacji Grahama. Omówiono problemy przepływowe szeregowania zadań oraz kryteria optymalizacyjne, w tym kryterium maksymalnego czasu zakończenia wykonywania wszystkich operacji. Dla omawianych problemów przedstawiono modele matematyczne i grafowe oraz metodę wyznaczania wartości funkcji celu.

Rozdział 3

Algorytmy optymalizacyjne

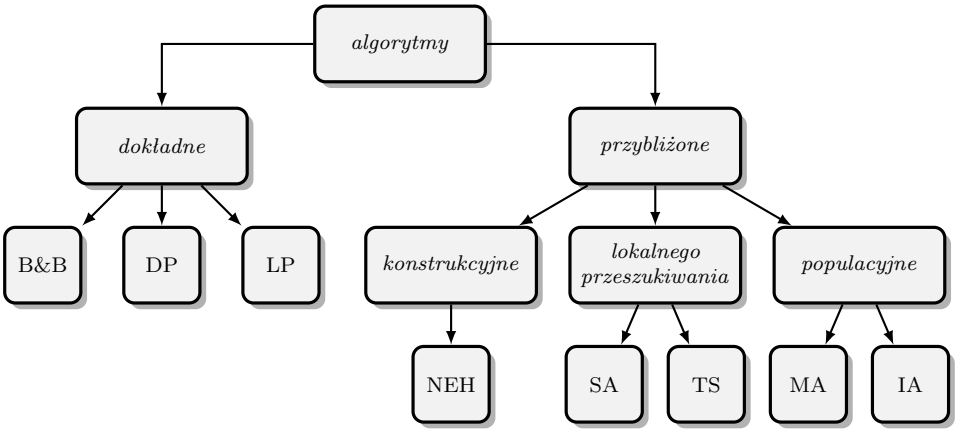
Problemy szeregowania zadań, niezależnie od przyjętych parametrów oraz formy funkcji celu, zazwyczaj się formułowane jako problemy optymalizacji dyskretnej. Problemy opisywanie w Rozdziale 2 w większości należą do klasy problemów NP-trudnych. Ze względu na mnogość występowania ekstremów lokalnych oraz rozmiar przestrzeni rozwiązań, nie są znane algorytmy dokładne (optymalne) ich rozwiązywania działające w czasie wielomianowym. Jeśli nie możemy lub nie mamy potrzeby znalezienia rozwiązania optymalnego, wtedy stosuje się metody przybliżone. Obie klasy metod zostaną opisane wraz z przykładami w niniejszym rozdziale. Klasyfikację wraz z przykładowymi algorytmami przedstawiono na Rysunku 3.1.

3.1 Metody dokładne

Algorytm dokładny zawsze wyznacza rozwiązanie optymalne dla badanego problemu. Najprostszą w implementacji metodą dokładną jest przegląd zupełny, czyli Metoda Siłowa (*Brute Force*, BF) polegająca na sprawdzeniu wszystkich możliwych rozwiązań. Stosowalność algorytmów dokładnych jest ograniczona do instancji problemu o małych rozmiarach, ponieważ metody te wymagają zbyt dużo czasu na wyznaczenie rozwiązania. W praktyce większość algorytmów dokładnych bazuje na schemacie: (1) Metody Podziału i Ograniczeń (*Branch and Bound*, B&B), (2) Programowania Dynamicznego (*Dynamic Programing*, DP) lub (3) modelu Programowania Liniowego (*Linear Programing*, LP).

Metoda Podziału i Ograniczeń

Metoda Podziału i Ograniczeń (B&B) [81] jest jedną z najczęściej stosowanych metod dokładnych dla różnych problemów optymalizacji dyskret-



Rysunek 3.1: Klasyfikacja algorytmów

nej. W algorytmie według drzewiastego schematu należy podzielić zbiór wszystkich rozwiązań na mniejsze, rozłączne podzbiory. Dla każdego podzbioru należy wyznaczyć wartość dolnego oszacowania (*lower bound*, LB) funkcji celu jaką można uzyskać w danym podzbiorze rozwiązań. Oprócz LB drugim równie istotnym elementem jest górne oszacowanie (*Upper Bound*, UB) funkcji celu, które można uzyskać. W praktyce dla problemów minimalizacji UB jest równy wartośći funkcji celu najlepszego znalezionego rozwiązania. Każdy podzbiór jest rekurencyjnie dzielony na kolejne mniejsze podzbiory jeśli wartość jego LB jest mniejsza niż UB, reszta zbiorów nie jest rozważana – następuje tak zwane odcięcie danej gałęzi rozwiązań. Podzbiory są dzielone do momentu uzyskania zbiorów jednoelementowych (liście drzewa), wtedy należy wyznaczyć dokładną wartość danego rozwiązania oraz zaktualizować UB jeśli nasępiła poprawa. Rekurencyjny charakter został przedstawiony w Algorytmie 1 na stronie 41.

W pesymistycznym przypadku mogą zostać sprawdzone wszystkie możliwe rozwiązania, dlatego złożoność obliczeniowa opisywanej metody zazwyczaj jest tożsama z BF. Istotnym elementem jest odpowiednie dobranie metody wyznaczania LB. Dla PFSSP funkcja wyznaczania LB zazwyczaj uwzględnia trzy główne składowe: (1) dokładny czas zakończenia wykonywania już uszeregowanych zadań, (2) czas wykonywania wszystkich jeszcze nieuszeregowanych operacji na konkretnych maszynach oraz (3) czas wykonywania najkrótszych, nieuszeregowanych operacji na kolejnych maszynach. Jeśli przez C^a_{sch} oznaczymy czas zakończenia wykonywania ostatniego uszeregowanego zadania na maszynie a oraz przez N zbiór zadań nieusze-

Algorytm 1: Metoda Podziału i Ograniczeń

Dane: N: zbiór zadań nieuszeregowanych
Wynik: π^{opt}: znalezione optymalne rozwiązanie.

```
1  N ← 𝒥, UB ← ∞, π ← ∅;
2  function BranchAndBound(j*, N, π):
3      π.append(j*);
4      N ← N \ {j*};
5      if N ≠ ∅ then
6          LB ← EstimateLowerBound (π, N);
7          if LB ⩽ UB then
8              for j ∈ N do
9                  BranchAndBound (j, N, π);
10     else
11         if f_max(π) < UB then
12             UB ← f_max(π);
13             π^opt ← π;
14 return π^opt
```

regowanych to LB możemy wyznaczyć według następujących wzorów:

$$LB_1(\pi, \mathrm{N}) = \max_{a \in \mathcal{M}} \left(C^a_{sch} + \sum_{i \in \mathrm{N}} p^a_i \right), \tag{3.1}$$

$$LB_2(\pi, \mathrm{N}) = \max_{a \in \mathcal{M}} \left(C^a_{sch} + \sum_{i \in \mathrm{N}} p^a_i + \sum_{b=a+1}^{m} \min_{i \in \mathcal{J}} p^b_i \right), \tag{3.2}$$

$$LB_3(\pi, \mathrm{N}) = \max_{a \in \mathcal{M}} \left(C^a_{sch} + \sum_{i \in \mathrm{N}} p^a_i + \sum_{b=a+1}^{m} \min_{i \in \mathrm{N}} p^b_i \right), \tag{3.3}$$

$$LB_4(\pi, \mathrm{N}) = \max_{a \in \mathcal{M}} \left(C^a_{sch} + \sum_{i \in \mathrm{N}} p^a_i + \min_{i \in \mathrm{N}} \sum_{b=a+1}^{m} p^b_i \right). \tag{3.4}$$

Funkcja (3.1) bierze pod uwagę jedynie czas zakończenia wykonywania już uszeregowanych zadań oraz sumę czasów wykonywania jeszcze nieprzydzielonych operacji na maszynach. Największą zaletą tej funkcji jest mała złożoność obliczeniowa $O(mn)$. W funkcji (3.2) uwzględniono wykonywanie operacji na kolejnych maszynach. Jeśli wcześniej wyznaczymy najkrótsze operacje na każdej maszynie to złożoność obliczeniowa tej metody również wynosi $O(mn)$, a będzie potrzeba zaledwie $O(m)$ dodatkowej pamięci.

Funkcja (3.3) podobnie jak (3.2) bierze pod uwagę najkrótsze pozostałe operacje, ale ze zbioru jeszcze nieuszeregowanych zadań (podobnie jak funkcja (3.4)). Złożoność obliczeniowa obu metod wynosi $O(nm^2)$.

Metoda B&B jest skutecznie implementowana również do rozwiązywana problemów optymalizacyjnych innych niż szeregowanie zadań, np. dla: (1) problemu komiwojażera (*Traveling Salesman Problem*, TSP) [9], (2) problemu marszrutyzacji pojazdów (*Vehicle Routing Problem*, VRP) [134], czy (3) problemu plecakowego (*Knapsack Problem*) [80].

Programowanie Dynamiczne

Programowanie Dynamicznie (PD) [69], podobnie jak B&B jest często stosowanym algorytmem dla problemów optymalizacji dyskretniej. Ideą DP jest rozwiązanie głównego problemu poprzez wykorzystanie wiedzy o jego podproblemach (*subproblems*). Rekurencyjna wersja algorytmu DP dla problemu $1||\sum w_i T_i$ została przedstawiona w Algorytmie 2 .

Algorytm 2: Programowanie Dynamiczne

Dane: $\boldsymbol{J}$: zbiór zadań
Wynik: memory: wartości funkcji celu

```
1  U ← J;
2  Initialize empty array memory;
3  function DynamicPrograming(U):
4      wiTi ← ∞;
5      for i ∈ U do
6          if !memory(U \ {i}) then
7              memory(U \ {i}) ← DynamicPrograming(U \ {i});
8          wiTi ← min{wiTi, max{ Σ_{j∈U} (p_j^1) − d_i, 0} w_i + memory(U \ {i})};
9      memory(U) ← wiTi;
10     return memory
```

W celu poprawnego działania należy zapewnić pamięć (`memory`) o rozmiarze $O(2^n)$, aby przechowywać wartości funkcji celu dla każdego podproblemu $\mathcal{U}$. Jeśli poprzez operacje bitowe oraz reprezentację zbioru przy użyciu jednej liczby dziesiętnej (bit „1" na pozycji x oznacza, że zadanie x jest w rozpatrywanym podproblemie $\mathcal{U}$) zapewni się dostęp do pamięci w czasie stałym $O(1)$, to złożoność obliczeniowa całej metody wyniesie $O(n2^n)$.

Programowanie Liniowe

Programowanie Liniowe (LP) [32] jest popularną metodą rozwiązywania problemów optymalizacyjnych w wypadku, gdy nie jest znany dedykowany algorytm dla badanego problemu. Podstawą LP jest zbudowanie liniowego modelu matematycznego zawierającego zbiór ograniczeń problemu w postaci układu równań lub nierówności. Rozwiązanie problemu należy opisać przy użyciu zmiennych decyzyjnych $\boldsymbol{x} = [x_i^a]_{m \times n}$ oraz funkcji kryterialnej $f(\boldsymbol{x})$, która będzie liniową kombinacją parametrów $\boldsymbol{x}$.

Jeśli wartości zmiennych decyzyjnych są ograniczone do liczb całkowitych, a w szczególności do liczb binarnych, w takim wypadku należy skorzystać z dedykowanej metody Programowania Liniowego Całkowitoliczbowego (*Mixed Integar Linear Programing*, MILP). Wszystkie metody LP znajdują zastosowanie do różnych problemów optymalizacji dyskretnej [5, 66] w tym do problemów szeregowania zadań [86, 124] oraz do problemów optymalizacji ciągłej. Metody LP wchodzą w skład popularnych pakietów optymalizacyjnych, takich jak `CPLEX` czy *Gurobi Optimizer*.

3.2 Metody przybliżone

Głównym zadaniem metod heurystycznych (przybliżonych) jest znalezienie rozwiązania w krótkim czasie kosztem jego jakości. Znajdują one szczególne zastosowanie w przypadkach, gdzie algorytm dokładny nie jest w stanie znaleźć rozwiązania w wyznaczonym czasie. Algorytmy przybliżone możemy zazwyczaj podzielić na 3 kategorie: (1) konstrukcyjne, (2) przeszukiwania lokalnego lub (3) populacyjne.

Istotnym wskaźnikiem efektowności algorytmu przybliżonego jest jakość uzyskanego wyniku. Najczęściej używa się procentowego względnego odchylenia rozwiązania względem rozwiązania referencyjnego (*Percentage Relative Deviation*, PRD):

$$\mathrm{PRD}(\pi) = \frac{f_{\max}(\pi) - f_{\max}(\pi_{\mathrm{ref}})}{f_{\max}(\pi_{\mathrm{ref}})} \cdot 100\%, \tag{3.5}$$

gdzie rozwiązaniem referencyjnym π_{ref} jest: (1) rozwiązanie optymalne, (2) najlepsze znalezione do tej pory lub (3) oszacowane.

3.2.1 Algorytmy konstrukcyjne

Algorytmy konstrukcyjne zwane również często zachłannymi zazwyczaj są metodami dedykowanymi do danego problemu. Ideą metody jest konsekwentne budowanie rozwiązania krok po kroku wybierając aktualnie najbardziej korzystny (zachłannie) ruch w danym momencie dla rozwiązania

częściowego. Przerwanie algorytmu podczas działania będzie skutkowało brakiem otrzymania rozwiązania. Najbardziej znaną metodą zachłanną jest algorytm Najbliższego Sąsiada (*Nearest Neighbor*, NN) [72] dla TSP. Dla innych problemów popularnym rozwiązaniem jest Szybkie Sortowanie (*Quick Sort*, QS) względem wybranego parametru, np.: dla problemów szeregowania zadań z żądanym czasem zakończenia wykonania zadań.

NEH

Algorytm NEH (Nawaz, Enscore i Ham) [94] jest algorytmem dedykowanym dla problemu $FP||C_{\max}$. Składa się z dwóch głównych etapów: (1) posortowanie zadań nierosnąco względem sumy czasów wykonywania operacji w ramach zadań oraz (2) dla każdego kolejnego zadania należy znaleźć lokalnie najlepszą pozycję poprzez metodę wstawień na wszystkich potencjalnych pozycjach. Złożoność przedstawionego algorytmu wynosi $O(n^2m)$, która jest dobrze widoczna na przedstawionym Algorytmie 3. Innym często spotykanym algorytmem w szeregowaniu zadań jest INSA [96] dla JSSP.

Algorytm 3: NEH

Dane: N: zbiór zadań nieuszeregowanych
Wynik: π^*: znalezione rozwiązanie.

```
1  N ← 𝒥;
2  function NEH(N):
3      π ← ∅;
4      N ← QuickSort(N, Σ_{a=1}^{m} p_i^a);
5      for i ∈ N do
6          for k ∈ {1, ..., Lenght(π)} do
7              if C_max(Insert(π, i, k)) < C_max(π*) then
8                  π* ← Insert(π, i, k);
9          π ← π*;
10     return π*;
```

3.2.2 Algorytmy przeszukiwania lokalnego

Algorytmy przeszukiwania lokalnego (*Local Search* LS) należą do większej klasy tzw. metaheurystyk. Przedstawione poniżej wersje algorytmów mogą być skutecznie stosownie nie tylko dla różnych problemów szeregowania

zadań, ale również innych problemów dyskretnych których rozwiązanie jest reprezentowanie przez permutację, np.: TSP lub VRP.

Ideą algorytmów przeszukiwania lokalnego jest iteracyjne przeszukiwanie przestrzeni rozwiązań poprzez polepszanie aktualnego rozwiązania. W przedstawionym Algorytmie 4 widać, że pierwszym krokiem jest wyznaczenie rozwiązania początkowego (naturalnego, losowego lub dostarczonego poprzez algorytm zachłanny). W każdej iteracji jest generowane nowe rozwiązanie z otoczenia aktualnego rozwiązania. Aktualne rozwiązanie zostaje nadpisane. Następnie należy porównać bieżące rozwiązanie z najlepszym znalezionym do tej pory, w przypadku poprawy należy je zaktualizować. Algorytm zazwyczaj wykonuje się dopóki nie zostanie osiągnięty limit iteracji (tak jak na załączonym przykładzie) lub inny warunek stopu (brak poprawy, limit czasowy). Przez otoczenie danego rozwiązania π rozumiemy

Algorytm 4: Przeszukiwanie Lokalne

Dane: π: rozwiązanie początkowe, maxIter: liczba iteracji.
Wynik: π^*: najlepsze znalezione rozwiązanie.

```
1 π ← InitSolution();
2 function LocalSearch(π, maxIter):
3     while it < maxIter do
4         π ← SearchNeihgborhood(π);
5         if f_max(π) < f_max(π*) then
6             π* ← π;
7         it ← it + 1;
8     return π*;
```

zbiór wszystkich permutacji zbioru $\mathcal{J}$, które możemy wygenerować z permutacji π za pomocą pojedynczego ruchu. W szeregowaniu zadań najczęściej spotyka się trzy rodzaje ruchów:

- Zamiana dowolnych par (*Swap*), rozwiązanie sąsiednie π' otrzymywane jest przez zamianę wartości $\pi(i)$ i $\pi(j)$, gdzie $i \in \mathcal{J}$ i $j \in \mathcal{J} \setminus \{i\}$. Rozmiar otoczenia wynosi dokładnie $\frac{n(n-1)}{2}$ dla PFSSP oraz SMSP.
- Zamiana przyległych par (*Adjacent Swap*), rozwiązanie sąsiednie π' otrzymywane jest przez zamianę wartości $\pi(i)$ oraz $\pi(i+1)$, gdzie $i \in \mathcal{J} \setminus \{n\}$. Rozmiar otoczenia wynosi dokładnie $(n-1)$ dla PFSSP oraz SMSP.
- Wytnij i wstaw (*Insert*), rozwiązanie sąsiednie π' otrzymywane jest przez wycięcie elementu na pozycji $\pi(i)$, następnie przesunięcie wszystkich elementów w lewo włącznie z elementem na pozycji $\pi(j)$, gdzie

zostanie wstawiony element $\pi(i)$, $i \in \mathcal{J}$ i $j \in \mathcal{J} \setminus \{i\}$. Rozmiar otoczenia wynosi dokładnie $n(n-1)$ dla PFSSP oraz SMSP.

Jednak najskuteczniejszymi otoczeniami są otoczenia blokowe [95], które korzystają z własności eliminacyjnych danego problemu. Zostały one zaproponowane przez Grabowskiego [52], następnie rozwijane przez Nowickiego i Smutnickiego [96]. Zmniejszając rozmiar sąsiedztwa nawet do $2m$ dla PFSSP, będzie możliwe dużo szybsze przeglądanie przestrzeni rozwiązań i otrzymanie dobrego jakościowo wyniku w bardzo krótkim czasie. Dla TSP możemy spotkać się dodatkowo z ruchem odwracania podciągów.

Przeszukiwanie z Zabronieniami

Przeszukiwanie z Zabronieniami (*Tabu Search, TS*) jest bardzo znanym algorytmem metaheurystycznym, który jest modyfikacją przeszukiwania lokalnego [46,47], który wykorzystuje pamięć krótkoterminową oraz niekiedy również długoterminową, aby uniknąć uwięzienia w lokalnym optimum. Ze względu na swój deterministyczny charakter i skuteczność. w praktyce jest to jedna z najczęściej stosowanych metod metaheurystycznych. Metoda jest wykorzystywana do różnych problemów od szeregowania zadań [14], poprzez problem plecakowy [79], wybór modelu [87], a nawet replikacja danych w środowiskach chmurowych [36].

Algorytm 5: Przeszukiwanie z zabronieniami

Dane: π: rozwiązanie początkowe, maxIter: liczba iteracji.
Wynik: π^*: najlepsze znalezione rozwiązanie.

```
1  π ← CreateInitialSolution();
2  function TabuSearch(π, maxIter):
3      π* ← π it ← 0;
4      while it < maxIter do
5          π ← SearchNeihgborhood(π);
6          UpdateTabuList();
7          if f_max(π) < f_max(π*) then
8              π* ← π;
9          it ← it + 1;
10     return π*;
```

Podstawowa wersja TS różni się kilkoma modyfikacjami od przedstawionego w Algorytmie 4 schematu LS. W fazie przeglądania sąsiedztwa zawsze wybierane jest najlepsze rozwiązanie z otoczenia, uwzględniając zakazane

ruchy umieszczone na liście zakazów (*tabu list*) nazywaną zazwyczaj listą tabu. Na liście tabu jest umieszczany ruch (a właściwie jego atrybuty), który doprowadził do utworzenia aktualnie wybranego rozwiązania. Dopuszcza się przyjęcie gorszego jakościowo rozwiązania, aby algorytm mógł „wyjść" z lokalnego optimum. Jednak w celu uniknięcia powrotu do optimum lokalnego a w konsekwencji do „zapętlenia", należy umieścić atrybuty takiego ruchu na liście tabu.

Na przestrzeni lat zostało zaproponowane wiele modyfikacji dla TS. Do najczęstszych zmian należą:

- Własności blokowe – zmniejszenie rozmiaru sąsiedztwa w celu znalezienia dobrego rozwiązania w krótkim czasie.
- Adaptacyjna długość listy tabu – skrócenie długości list, aby dokładniej przeszukać określony region przestrzeni rozwiązań (intensyfikacja) lub wydłużenie długości listy, aby przeszukać większy obszar przestrzeni rozwiązań (dywersyfikacja).
- Kryterium aspiracji – przyjęcie zakazanego rozwiązania jeśli spełnia dodatkowe warunki.
- Multistart – równoległe, niezależne wykonywanie algorytmu dla różnych rozwiązań początkowych.
- Restart – wylosowanie nowego rozwiązania po zadanej liczbie iteracji bez poprawy.
- Skoki powrotne – powrót do wcześniejszego rozwiązania i skierowanie przeszukiwania w inną stronę.

Złożoność obliczeniowa metody TS zależy od wyboru poszczególnych elementów. Zależy zarówno od wybory ruchu, warunku stopu czy sposobu przechowywania i przeglądania listy tabu jak i jej długości. W Części II. dysertacji zostaną przedstawione szczegółowe warianty algorytmów opartych na TS dla rozważanych problemów.

Symulowane Wyżarzanie

Symulowane Wyżarzanie (*Simulated Annealing*, SA) [71] podobnie jak TS również jest modyfikacją metody LS. SA jest metaheurystyką o probabilistycznym charakterze inspirowaną metalurgicznym procesem wyżarzania metali. Najpierw metal jest rozgrzewany do wysokiej temperatury, aby był bardziej podatny na zmiany. Następnie jest powoli schładzany, z przejściowym podgrzewaniem.

Konstrukcję omawianej metody przedstawiono w Algorytmie 6. W każdej iteracji wybierany jest losowo sąsiad z danego otoczenia. Lepsze rozwiązanie (w przypadku minimalizacji o niższej wartości funkcji celu) zawsze

jest akceptowane (prawdopodobieństwo akceptacji wynosi 1). W przeciwnym wypadku rozwiązanie może być zaakceptowane z pewnym prawdopodobieństwem. Funkcję prawdopodobieństwa przyjęcia nowego rozwiązania nazywamy funkcją akceptacji (*acceptance function*). Funkcja akceptacji zależy od różnicy wartości funkcji celu obu rozwiązań oraz od wartości parametru, tzw. aktualnej temperatury. Prawdopodobieństwo akceptacji maleje wraz ze spadkiem temperatury oraz rosnącą różnicą między rozwiązaniami. Temperatura spada po każdej iteracji według przyjętego schematu chło-

Algorytm 6: Symulowane Wyżarzanie

Dane: π: rozwiązanie początkowe, $\mathtt{T}_0$: temperatura początkowa, cr: współczynnik wychładzania.
Wynik: π^*: najlepsze znalezione rozwiązanie.

```
1  π ← InitSolution();
2  function SimulatedAnnealing(π, T_0, cr):
3      currTemp ← T_0;
4      while currTemp > 0.01 do
5          π' ← RandomNeihgbor(π);
6          if AcceptanceFunction(Δf_max(π, π'), currTemp) then
7              π ← π';
8              if f_max(π) < f_max(π*) then
9                  π* ← π;
10         currTemp ← currTemp * cr;
11     return π*;
```

dzenia (*cooling scheme*) zazwyczaj: (1) geometrycznego, (2) liniowego lub (3) logarytmicznego. W przypadku zbyt szybkiego spadku temperatury dodaje się w algorytmie powtórzenia dla zadanej temperatury.

Tak samo jak w przypadku TS trudno jest precyzyjnie określić złożoność obliczeniową algorytmu, ponieważ zależy od wyboru ruchu, schematu chłodzenia wraz z parametrami, oraz od sposobu implementacji powtórzeń. SA znajduje z sukcesem zastosowanie w różnych problemach optymalizacji dyskretnej [99, 105].

3.2.3 Algorytmy Populacyjne

Algorytmy Populacyjne (*Population-based Algorithms* PA) w przeciwieństwie do algorytmów przeszukiwania lokalnego operują na zbiorze rozwiązań zwanym populacją $\mathcal{P}_k$. Znacząca grupa PA to Algorytmy Ewolucyjne

(*Evolutionary Algorithm*, EA), a w szczególności Algorytmy Genetyczne (*Genetic Algorithm* GA). Wspólną cechą wszystkich PA jest mechanizm rozwiązywania problemu inspirowany biologicznym procesem doboru naturalnego i ewolucji [6]. Twórcą podstawowego GA jest Holland [59, 60].

Algorytm 7 przedstawia ogólny schemat PA. Na początku należy wygenerować populację początkową $\mathcal{P}_0$. Najczęściej populacja początkowa jest losowa, dopuszczalna, tj.: zgodna z ograniczeniami problemu. W celu zapobiegnięcia szybkiej zbieżności wszystkich osobników do jednego minimum, warto zapewnić odpowiedni poziom dywersyfikacji rozwiązań początkowych. Częstym zabiegiem jest dołączenie do populacji początkowej rozwiązań otrzymanych z innych algorytmów heurystycznych zachłannych lub przeszukiwania lokalnego. W każdym pokoleniu (iteracji) trzeba przepro-

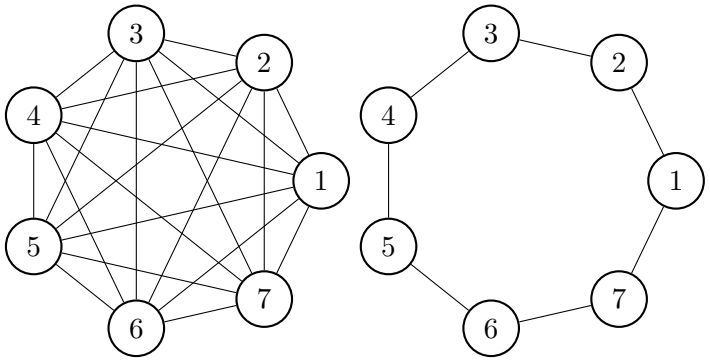
Algorytm 7: Algorytm Populacyjny

Dane: maxIter: liczba iteracji/pokoleń, size: rozmiar populacji
Wynik: π^*: najlepsze znalezione rozwiązanie.

```
1  function PopulationBasedAlgorithms(maxIter,size):
2      P_it ← GenerateInitPopulation(size);
3      while it < maxIter do
4          π' ← arg min_{π∈P_it} f_max(π);
5          if f_max(π') < f_max(π*) then
6              π* ← π';
7          P'_it ← GenerateNewPopulation(P_it) ;
8          P_{it+1} ← Selection(P_it, P'_it) ;
9          it ← it + 1;
10     return π*;
```

wadzić ewaluacje wszystkich osobników. Następnie w przypadku EA i GA nowe pokolenie jest tworzone przy użyciu operatorów genetycznych:

- Selekcja – proces doboru osobników do kolejnej populacji lub dobór rodziców z bieżącej populacji. W literaturze najczęściej spotyka się trzy typy selekcji: (1) ruletka – wybór przez użycie koła ruletki, gdzie premiowane są osobniki lepiej przystosowane (mają przypisany większy wycinek koła) (2) turniej – wybór najlepiej przystosowanego osobnika z grupy o określonym rozmiarze oraz (3) losowy, gdzie wszystkie osobniki mają równą szansę zostania wybranym.



Rysunek 3.2: Przykładowe schematy połączeń między wyspami

- Krzyżowanie – wymiana informacji pomiędzy parą osobników, dzięki której powstaje jeden lub dwa nowe osobniki potomne. Wyróżnia się krzyżowania jedno- lub wielo-punktowe [74].
- Mutacja – losowa zmiana (ruch) w genotypie osobnika, aby zapobiegać stagnacji algorytmu. Zazwyczaj stosuje się niskie prawdopodobieństwo zajścia mutacji u osobnika.

W celu poprawienia efektywności algorytmów tworzy się operatory dedykowane do rozwiązywania badanego problemu [93]. Złożoność obliczeniowa opisanych metod zależy od doboru konkretnych operatorów oraz ich parametrów.

Algorytm Wyspowy

W metodach opartych na schemacie Algorytmu Wyspowego (*Island Algorithm* IA) [138] korzysta się z operatorów genetycznych typowych dla algorytmów EA oraz GA. Istotną różnicą jest podział głównej populacji na mniejsze rozłączne podpopulacje (*subpopulations*) nazywane wyspami. Wewnątrz odseparowanych wysp działają operatory genetyczne (selekcja, krzyżowanie i mutacja). Główną ideą IA jest duża dywersyfikacja przeszukiwania przestrzeni rozwiązań, ponieważ rozwiązania wewnątrz każdej wsypy powinny zbiegać do innego ekstremum lokalnego.

W celu uniknięcia przedwczesnej zbieżności na jednej z wysp stosuje się wymianię informacji genetycznych między wyspami w określonych cyklach. Połączenia między wyspami są opisane grafem. Przykładowe graf połączeń pokazano na Rysunku 3.2. Pierwsze pokazane połączenie jest typu gwiazda (po lewej na rysunku), gdzie każda wyspą wymienia informacje ze wszystkimi innymi. Po prawej przedstawiono połączenie typu pierścień,

gdzie każda wyspa wymienia informacje tylko z sąsiednimi wyspami. Struktura grafu może być dowolna i nie musi mieć regularnej postaci. Zaletą algorytmu jest prosta możliwość zrównoleglenia obliczeń, jeśli każdej wyspie zostanie przypisany osobny wątek procesora.

Algorytm Memetyczny

W praktyce okazuje się, że klasyczne algorytmy EA i GA skutecznie radzą sobie z „szerokim przeglądaniem" (dywersyfikacją poszukiwań, eksploracją) przestrzeni rozwiązań i zlokalizowaniem obszaru, w którym znajduje się lokalne ekstremum. Jednak mają problem z eksploatacją danego obszaru. Na przestrzeni lat powstał pomysł, żeby każde rozwiązanie z wybranej grupy osobników spośród populacji poprawić algorytmem LS. Moscato w pracy [91] zaproponował, aby takie połączenie algorytmów nazwać Algorytmem Memetycznym (*Memetic Algorithm* MA). MA są zaliczane do grupy hybrydowych algorytmów ewolucyjnych, a w teorii opierają się na hipotezie ewolucji Lamarcka.

Metody bazujące na MA charakteryzują się lepszą zbieżnością niż standardowe metody EA lub GA, kosztem czasu wykonywania oraz przedwczesnej stagnacji algorytmu. Najczęściej do poprawy pojedynczego osobnika stosuje się SA [92] lub Algorytm Wspinaczkowy (*Hill Climbing*, HC) [85] – drobna modyfikacja LS, która w momencie osiągnięcia ekstremum lokalnego kończy działanie lub „przeskakuje" w losowe miejsce w przestrzeni rozwiązań, aby ponownie „wspiąć się" do optimum. Dla TSP skuteczną metodą lokalnej poprawy jest algorytm 2-opt [118], który radzi sobie skutecznie z „rozplątywaniem" przecinających się ścieżek.

Algorytm Rojowy

Drugą dużą klasą PA są algorytmy bazujące na inteligencji rozproszonej zwanej również inteligencją roju (*Swarm Intelligence*, SI) zwane też ogólnie Algorytmami Rojowymi (*Swarm-based Optimization Algorithm*, SOA). W tym przypadku zbiorem rozwiązań może być rój, grupa czy stado:

- cząstek (*Particle Swarm Optimization)* [70]),
- mrówek (*Ant Algorithm* [28] lub *Ant Colony Optimization* [35]),
- pszczół (*Bees Algorithm [104]* lub *Artificial Bee Colony* [68]),
- nietoperzy (*Bat Algorithm* [144]),
- wielbłądów (*Camel Algorithm* [76]),
- wilków (*Wolf Search Algorithm* [131], *Grey Wolf Optimizer* [90]).

Cechą wspólną wymieniony algorytmów jest możliwość przemieszczania się cząstek/osobników w przestrzeni według wyznaczonej trajektorii z ustaloną

prędkością. Cząstki/osobnicy mogą wchodzić w pośrednie lub bezpośrednie interakcje między sobą. Zazwyczaj w podstawowych wersjach SOA nie stosuje się operatorów genetycznych. Jednak w literaturze można znaleźć np.: algorytm sztucznej koloni pszczół z dodanym krzyżowaniem [77].

3.3 Wnioski i uwagi

Przedstawiony przegląd metod wraz zaproponowaną klasyfikacją nie wyczerpuje listy znanych w literaturze algorytmów optymalizacyjnych. Omówione metody dokładne (w szczególności Metoda Podziału i Ograniczeń oraz Programowanie Liniowe Całkowitoliczbowe) będą w późniejszych rozdziałach pracy służyć do wyznaczania rozwiązań referencyjnych, lecz wyłącznie dla małych rozmiarów instancji, dla proponowanych problemów ze sprzężeniami czasowymi. Dla dużych rozmiarów instancji rozwiązania referencyjne będą wyznaczane przy użyciu podstawowych wersji różnych algorytmów przybliżonych. Głównym badanym algorytmem będzie Przeszukiwanie z Zabronieniami wykorzystujące między innymi własności eliminacyjne proponowanych problemów. Algorytmy konstrukcyjne będą używane do generowania rozwiązań początkowych dla algorytmów lokalnego poszukiwania lub wyznaczania górnego ograniczenia w metodach dokładnych.

Część II

Własności i algorytmy dla problemów przepływowych

Rozdział 4

Problemy ze sprzężeniami czasowymi

Pomimo dużej różnorodności wariantów problemów przepływowych szeregowania zadań opisanych w Rozdziale 2, w praktyce często są one niewystarczające. Zazwyczaj kłopot pojawia się przy rozważaniu maszyn o nietypowych własnościach. Interesującym przypadkiem są procesy związanie z betonowaniem w ramach harmonogramowania procesów budowlanych.

Dobrym przykładem takiego zagadnienia jest proces betonowania, gdzie jednym z frontów budowlanych (maszyn) jest wylewanie mieszanki betonowej na budowie. Betoniarka (pojazd) musi załadować następną partię mieszkanki przed dojazdem na kolejne stanowisko (zadanie). Ponadto mieszankę przed wylaniem należy mieszać przez odpowiedni interwał czasu – zbyt długie lub zbyt krótkie mieszanie może wpłynąć na właściwości mieszanki lub nawet uszkodzić pojazd. Jednak mieszanie nie może być wliczone w czas operacji, ponieważ prace na budowie mogą być kontynuowane, gdy tylko betoniarka opuści stanowisko. W procesie szeregowania zadań możemy opisać to jako dodatkową zależność między czasem rozpoczęcia i zakończenia operacji na konkretnej maszynie (w tym przypadku betoniarce).

W efekcie w harmonogramie powstaną przerwy między operacjami o minimalnej i maksymalnej długości wynikające z dodatkowej wzajemnej relacji między tymi operacjami. Ograniczania te będą nazywane **sprzężeniami czasowymi** (*time couplings*). W Podrozdziale 4.1 zostanie opisany permutacyjny problem szeregowania zadań ze sprzężeniami czasowymi [19, 20] (*Permutation Flow Shop Scheduling Problem with Time Coupling*, PFSSP-TC) a w Podrozdziale 4.2 zostanie opisany niepermutacyjny problem szeregowania zadań ze sprzężeniami czasowymi [22, 63](*non-permutation Flow Shop Scheduling Problem with Time Coupling*, FSSP-TC).

4.1 Permutacyjny problem przepływowy

W tym rozdziale zostanie przedstawiony permutacyjny problem przepływowy ze sprzężeniami czasowymi. Następnie zostanie sformułowany jego model matematyczny oraz grafowy. Będzie zidentyfikowane kilka własności, w tym metoda wyznaczania wartości funkcji celu. Zaproponowane zostanie kilka metod rozwiązania: (1) Metoda Podziału i Ograniczeń, (2) Przeszukiwanie Snopowe, (3) Przeszukiwanie z Zabronieniami oraz (3) Sztuczna Kolonia Pszczół. Na koniec zostaną omówione wyniki eksperymentów obliczeniowych.

4.1.1 Definicja problemu

Przez $\mathcal{J} = \{1, 2, \ldots, n\}$ oraz $\mathcal{M} = \{1, 2, \ldots, m\}$ oznaczono odpowiednio zbiory n zadań oraz m maszyn. Każde zadanie i składa się z m operacji z zbioru $\mathcal{O}_i = \{l_i+1,\ l_i+2,\ \ldots,\ l_i+m\}$, gdzie $l_i = m(i-1)$. Zatem łącznie jest nm operacji. Przez $\mathcal{O}$ oznaczono zbiór wszystkich operacji, w którego skład wchodzą wszystkie operacje z poszczególnych zadań:

$$\mathcal{O} = \bigcup_{i \in \mathcal{J}} \mathcal{O}_i. \tag{4.1}$$

Każde zadanie i musi zostać wykonane na maszynach w kolejności technologicznej: $1 \to 2 \to \cdots \to m$. Czas wykonywania zadania i na maszynie a (tzn. operacji $l_i + a$) został oznaczony przez $p_i^a > 0$. Ponadto, niech $\hat{r}_a \geqslant 0$ oraz $\hat{d}_a \geqslant \hat{r}_a$ oznaczają minimalny i maksymalny czas przestoju (*idle time*) maszyny a dozwolony między przetwarzaniem kolejnych operacji na tej maszynie.

Kolejność wykonywania operacji w ramach każdego zadania jest stała, ponieważ wynika z ograniczeń technologicznych procesu. Przez π oznaczono n-elementową kolejność zadań. Zatem $\pi(i)$ oznacza indeks zadania wykonywanego jako i-tego w kolejności π. Rozwiązaniem będzie nazywana kolejność π.

Na podstawie rozwiązania π można zbudować odpowiadający mu *harmonogram* $(\boldsymbol{S}, \boldsymbol{C})$ dosunięty w lewo. Harmonogram składa się z dwóch macierzy: (1) macierzy momentów rozpoczęcia wykonywania operacji $\boldsymbol{S} = [S_i^a]^{m \times n}$, gdzie S_i^a jest czasem rozpoczęcia wykonywania i-tej operacji na maszynie a oraz (2) macierzy momentów zakończenia wykonywania operacji $\boldsymbol{C} = [C_i^a]^{m \times n}$, gdzie C_i^a jest czasem zakończenia wykonywania i-tej operacji na maszynie a. Aby harmonogramu $(\boldsymbol{S}, \boldsymbol{C})$ był dopuszczalny, musi

Tablica 4.1: Instancja problemu PFSSP-TC z Przykładu 4.1

a	p_1^a	p_2^a	p_3^a	p_4^a	p_5^a	$\hat{r}_a$	$\hat{d}_a$
1	1	3	3	2	1	1	5
2	2	3	2	1	4	0	0
3	1	3	2	6	1	1	2

spełniać następujące ograniczenia:

$$\forall a \in \mathcal{M} \quad \forall i \in \mathcal{J} \quad S_i^a, C_i^a \geqslant 0 \tag{4.2}$$

$$\forall a \in \mathcal{M} \quad \forall i \in \mathcal{J} \quad C_i^a = S_i^a + p_i^a, \tag{4.3}$$

$$\forall a \in \mathcal{M} \quad \forall i \in \mathcal{J} \setminus \{1\} \quad \hat{r}_a \leqslant S_i^a - C_{i-1}^a \leqslant \hat{d}_a, \tag{4.4}$$

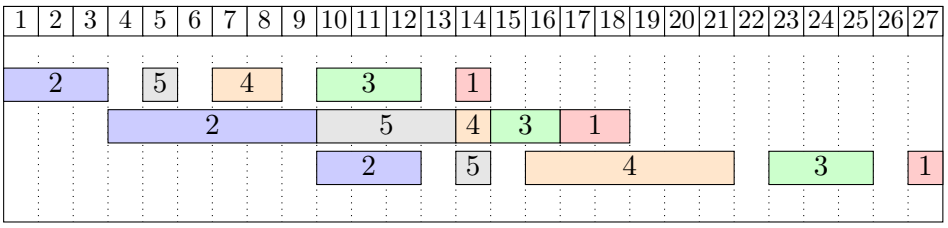
$$\forall a \in \mathcal{M} \setminus \{1\} \quad \forall i \in \mathcal{J} \quad S_i^a \geqslant C_i^{a-1}, \tag{4.5}$$

Bez utraty ogólności można przyjąć czas rozpoczęcia wykonywania pierwszej operacji z zadania pierwszego w momencie 0, w rezultacie $S_1^1 = 0$. Ograniczenie (4.3) zapewnia ciągłość wykonywania operacji. Ograniczenie (4.4) wymusza przestrzeganie minimalnego oraz maksymalnego czasu przestoju maszyny, ale również wymaga, aby operacja i-ta w kolejności była wykonywana dopiero po zakończeniu wykonywania poprzedniej operacji $i-1$ (poprzednik maszynowy) na tej samej maszynie a. Podobnie, ograniczenie (4.5) wymusza rozpoczęcie wykonywania operacji i-tej na maszynie a dopiero po zakończeniu wykonywania operacji i-tej na maszynie poprzedniej $a-1$ (poprzednika technologicznego).

Na podstawie ograniczenia (4.3) można zaobserwować, że dopuszczalny harmonogram może być jednoznacznie opisany za pomocą $\boldsymbol{S}$ lub $\boldsymbol{C}$. Harmonogram będzie nazywany dosuniętym w lewo (*left-shifted*), jeśli ani jedna operacja nie może rozpocząć wykonywać się wcześniej bez naruszenia ograniczeń lub zmiany kolejności wykonywania zadań. W Podrozdziale 4.1.2 pokazano metodę wyznaczania harmonogramu dosuniętego w lewo, włącznie z uwzględnieniem złożoności obliczeniowej.

Przykład 4.1 W Tabeli 4.1 pokazano instancję problemu FSSP-TC. Na Rysunku 4.1 przedstawiono w lewo dosunięty harmonogram dla tej instancji oraz permutacji $\pi = (2, 5, 4, 3, 1)$.

Badanym kryterium optymalizacyjnym będzie długość uszeregowania, czyli czas zakończenia wykonywania wszystkich operacji. Przez $C_{\max}(\pi)$ oznaczono czas zakończenia wykonywania operacji, która zakończyła się wykonywać jako ostatnia dosuniętym w lewo harmonogramie dla kolejności



Rysunek 4.1: Harmonogram dla rozwiązania π z Przekładu 4.1

π. Z ograniczeń (4.3)–(4.5) wynika, że będzie to czas zakończenia wykonywania ostatniej operacji w ostatnim zadaniu $C_{\max}(\pi) = C^m_{\pi(n)}$. Celem będzie znalezienie takiego rozwiązania π, by:

$$C_{\max}(\pi^{\text{opt}}) = \min_{\pi \in \Pi} C_{\max}(\pi), \tag{4.6}$$

gdzie Π jest zbiorem wszystkich dopuszczalnych rozwiązań. Rozwiązanie π^{opt} będzie nazywane rozwiązaniem optymalnym.

Według rozszerzonej notacji Grahama [57] problem PFSSP-TC będzie oznaczany jako $FP|\texttt{minimal idle, maximal idle}|C_{\max}$. W celu rozpatrywania klasycznego wariantu PFSSP, tzn. $FP||C_{\max}$ należy przyjąć $\hat{r}_a = 0$ oraz $\hat{d}_a = \infty$ (dla wszystkich a). W przepadku PFSSP bez przestojów $FP|\texttt{no idle}|C_{\max}$ należy przyjąć $\hat{r}_a = 0$ oraz $\hat{d}_a = 0$. Dla PFSSP wyłącznie z minimalnym czasem przestoju (*inserted idle*) $FP|\texttt{minimal idle}|C_{\max}$ należy przyjąć $\hat{d}_a = \infty$. Dla PFSSP wyłącznie z maksymalnym czasem przestoju (*limited idle*) $FP|\texttt{maximal idle}|C_{\max}$ należy przyjąć $\hat{r}_a = 0$. Proponowane własności oraz metody pokazane w kolejnych rozdziałach można stosować w każdym z wymienionych przypadków, a zwłaszcza dla $FP|\texttt{minimal idle}|C_{\max}$ oraz $FP|\texttt{maximal idle}|C_{\max}$ dla których w literaturze jeszcze nie sformułowano dedykowanych metod, według najlepszej wiedzy autora.

Model grafowy

Jedną z klasycznych technik szeregowania zadań jest budowa modelu grafowego dla danego rozwiązania π. Należy utworzyć graf rozwiązania $G(\pi) = (V,\ E_1 \cup E_2)$, gdzie:

$$V = \bigcup_{\substack{a \in \mathcal{M} \\ i \in \mathcal{J}}} \Big\{ (a, i) \Big\}. \tag{4.7}$$

Przez V został oznaczony zbiór wierzchołków reprezentujących operacje (dla uproszczenia wizualizacji ustawione w formie siatki). Operacja z zadania i-tego na maszynie a jest reprezentowana przez obciążony wierzchołek w i-tej kolumnie oraz a-tym wierszu oznaczony parą (a, i) z wagą p_i^a.

$$E_1 = \bigcup_{\substack{a \in \mathcal{M} \\ i \in \mathcal{J} \setminus \{n\}}} \Big\{ \Big((a,i),\ (a,i+1) \Big) \Big\}, \tag{4.8}$$

$$E_2 = \bigcup_{\substack{a \in \mathcal{M} \setminus \{m\} \\ i \in \mathcal{J}}} \Big\{ \Big((a,i),\ (a+1,i) \Big) \Big\}. \tag{4.9}$$

Łuki grafu $G(\pi)$ reprezentują ograniczenia problemu: (1) łuki poziome E_1 – ograniczenia maszynowe oraz (2) łuki pionowe E_2 – ograniczenia technologiczne. Długość najdłużej ścieżki dochodzącej do danego wierzchołka (a, i) jest równa odpowiedniemu czasowi zakończenia wykonywania C_i^a. Przez długość ścieżki rozumie się sumę wag wierzchołków i krawędzi należących do ścieżki. Przez $\mathrm{len}(p)$ oznaczono długość ścieżki p.

Kolejnym krokiem jest rozszerzenie grafu rozwiązania $\mathcal{G}(\pi)$. Minimalny czas przestoju maszyny został uwzględniany, podobnie jak w przypadku przezbrojeń niezależnych od kolejności [17], poprzez przypisanie każdemu łukowi „poziomemu" $((a,i),\ (a,i+1))$ w zbiorze E_1 wagi $\hat{r}_i$. Po drugie, w celu uwzględnienia maksymalnego czas przestoju maszyny należy wprowadzić nowy zbiór łuków:

$$E_3 = \bigcup_{\substack{a \in \mathcal{M} \\ i \in \mathcal{J} \setminus \{n\}}} \Big\{ \Big((a,i+1),\ (a,i) \Big) \Big\}. \tag{4.10}$$

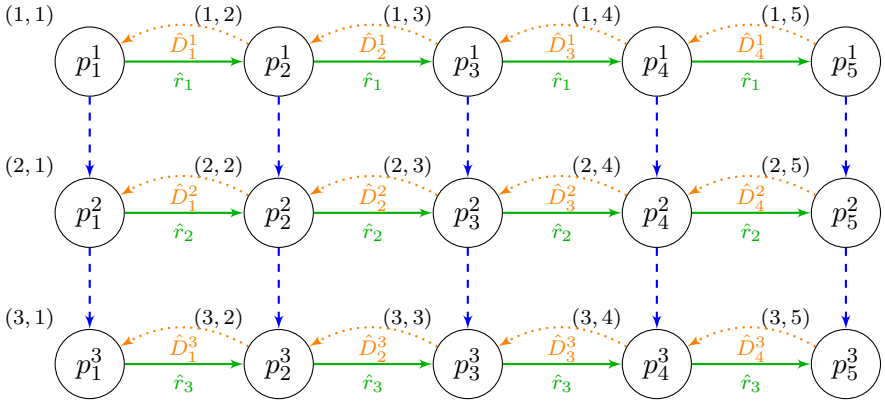
Każdy łuk „powrotny" $((a,i+1),\ (a,i))$ w zbiorze E_3 ma wagę $\hat{D}_i^a$, zdefiniowaną jako:

$$\hat{D}_i^a = -\rho_{a,i} - \rho_{a,i+1} - \hat{d}_a. \tag{4.11}$$

Strukturę grafu $\mathcal{G}(\pi)$ dla problemu o rozmiarze 5×3 oraz kolejności $\pi = (1,2,3,4,5)$ przedstawiono na Rysunku 4.2. Dla ułatwienia czytelności łuki E_1, E_2 oraz E_3 oznaczono kolejno zieloną linią ciągła, niebieską przerywaną oraz pomarańczowa kropkowaną.

4.1.2 Własności problemu

Poniżej zostaną przedstawione oraz omówione nowe własności dla problemu $FP|\texttt{minimal idle, maximal idle}|C_{\max}$ w tym: dopuszczalność rozwiązań, złożoność obliczeniowa metody wyznaczania wartości funkcji celu.



Rysunek 4.2: Przykładowy graf $\mathcal{G}(\pi)$ dla problemu PFSSP-TC

Własność 4.1 *Dla dowolnego rozwiązania $\pi \in \Pi$, graf rozwiązania $\mathcal{G}(\pi)$ dla problemu $FP|$`minimal idle, maximal idle`$|C_{max}$ nie zawiera cykli o dodatniej długości.* □

Dowód. Należy zauważyć, że graf $\mathcal{G}(\pi)$ nie zawiera łuków idących „w górę" (tzn. nie istnieje łuk wychodzący z wierzchołka (b, i) do (a, i)) dla $a < b$. Zatem cykl może powstać tylko za pomocą łuków „poziomych" i „powrotnych".

Rozważmy cykl składający się z dwóch sąsiednich wierzchołków (a, i) oraz $(a, i+1)$. Wierzchołki mają wagi p_i^a i p_{i+1}^a, podczas gdy łuki łączące te dwa wierzchołki mają wagi $\hat{r}_a$ oraz $\hat{D}_i^a = -p_i^a - p_{i+1}^a - \hat{d}_a$. Po zsumowaniu wag długość takiego cyklu wynosi $\hat{r}_a - \hat{d}_a \leqslant 0$. Teraz zostanie rozważony przypadek bardziej ogólny dla cyklu składającego się z $k > 2$ wierzchołków w jednym „rzędzie" (na tej samej maszynie). Taki cykl zawiera (dla pewnych a i i) wierzchołki od (a, i) do $(a, i + k - 1)$, a także $k - 1$ łuków „poziomych" z wagami $\hat{r}_a$ oraz $k - 1$ łuków „powrotnych" z wagami $\hat{D}_i^a$. Kolejnym krokiem jest obliczenie długości takiego cyklu. Bez utraty ogólności można założyć, że cykl zaczyna się w wierzchołku $(i, j + k - 1)$. Rozważmy wewnętrzne wierzchołki tego cyklu, czyli wierzchołki od $(a, i+1)$ do $(a, i + k - 2)$. Niech (a, j) będzie oznaczać jeden z wewnętrznych wierzchołków. W trakcie przejścia przez cykl, trzeba przejeść przez każdy wierzchołek dwa razy dodając w ten sposób $2p_j^a$ do długości cyklu za każdy wierzchołek wewnętrzny. Ponadto przechodząc przez wierzchołek (a, j) dodajemy dwukrotnie wagę $-p_j^a$: (1) raz z łuku „powrotnego" kończącego się w wierzchołku (a, j) oraz (2) raz z łuku „powrtonego" zaczynającego się w wierzchołku (a, j). Zatem jak dotąd długość cyklu wynosi 0, ponie-

waż wszystkie wagi $2p_j^a$ oraz $-2p_j^a$ sumują się do 0. Powyższe wyliczenie jest równoważne sytuacji, w której wierzchołki od $(a, i+1)$ do $(a, i+k-2)$ mają wagi 0, a łuki „powrotne" składają się jedynie ze składowej $\hat{d}_a$ w ich wadze. Jedynymi wyjątkami są łuk „powrotny" rozpoczynający się w wierzchołku początkowym $(a, i+k-1)$ oraz łuk „powrotny" kończący się w wierzchołku końcowym $(a, i+1)$. W wagach tych łuków dalej są wartości $-p_{j+k-1}^a$ oraz $-p_j^a$. Jednak łatwo zauważyć, że te dwie wartości zostaną „zniwelowane" przez wagi wierzchołków (a, i) oraz $(a, i+k-1)$, gdy ścieżka przejdzie przez te wierzchołki tylko raz w trakcie cyklu. W ten sposób można obliczyć długość cyklu tak, jakby wszystkie wierzchołki cyklu miały wagi 0, a jego łuki „powrotne" miały wagi uwzględniające tylko $-\hat{d}_i$. Długość cyklu wynosi wtedy dokładnie $(k-1)(\hat{r}_i - \hat{d}_i) \leqslant 0$. ■

W związku z tym, wszystkie cykle w grafie $\mathcal{G}(\pi)$ mają długość niedodatnią. Obliczając czas zakończenia wykonywania wszystkich zadań, będzie można zignorować cykle zgodnie z następującą obserwacją.

Wniosek 4.1 Dla każdego rozwiązania π, jeśli graf rozwiązania $\mathcal{G}(\pi)$ dla $FP|\texttt{minimal idle, maximal idle}|C_{\max}$ zawiera ścieżkę p o długości $\text{len}(p)$ takiej, że p zawiera cykl, to graf $\mathcal{G}(\pi)$ zawiera również ściekę p' bez cyklu, taką że:

$$\text{len}(p') \geqslant \text{len}(p). \tag{4.12}$$

□

Zatem, jeśli ścieżka p w grafie $\mathcal{G}(\pi)$ zawiera cykl, to: (1) nie jest to najdłuższa ścieżka w grafie $\mathcal{G}(\pi)$ (przynajmniej jeden cykl w ścieżce p ma ujemną długość) lub (2) możemy skonstruować inną ścieżkę zaczynającą się i kosząca dokładnie w tych samych wierzchołkach, ale niezawierającej cyklu (wszystkie cykle w ścieżce p mają długość zero).

Twierdzenie 4.1 *Dla każdego rozwiązania $\pi \in \Pi$ permutacyjnego problemu $FP|\texttt{minimal idle, maximal idle}|C_{max}$, istnieje dopuszczalny oraz jednocześnie dosunięty w lewo harmonogram (S, C). Ponadto, wartość kryterium $C_{max}(\pi)$ dla harmonogramu można wyznaczyć w czasie $O(nm)$.* □

Dowód. Dowód ma charakter konstrukcyjny i jest oparty na Algorytmie 8. Poniżej przedstawiono szczegółowy opis algorytmu włącznie z jego złożonością obliczeniową. Algorytm jest podzielony na m faz. W każdej fazie należy określić czasy wykonywania wszystkich operacji na maszynie a. Za każdym razem, gdy jest obliczany lub aktualizowany czas zakończania C_i^a, trzeba równocześnie zmienić czas rozpoczęcia wykonywania operacji $S_i^a = C_i^a - p_i^a$. Trzymając się tej zasady zostanie zachowanie ograniczenie (4.3).

Algorytm 8: Konstruowanie harmonogramu dosuniętego w lewo

Dane: $\boldsymbol{\pi}$: kolejność wykonywania zadań
Wynik: $\boldsymbol{C}$: macierz momentów zakończenia wykonywania operacji

```
1  function Evaluate(π):
2      C_1^1 ← p_1^1;
3      for i = 2, 3, ..., n do
4          C_i^1 ← C_{i-1}^1 + r̂_0 + p_i^1;
5      for a = 2, 3, ..., m do
6          C_1^a ← C_i^{a-1} + p_1^a;
7          for i = 2, 3, ..., n do
8              C_i^a ← max{C_{i-1}^a + r̂_a, C_i^{a-1}} + p_i^a;
9          for i = n − 1, n − 2, ..., 1 do
10             C_i^a ← max{C_i^a, C_{i+1}^a − p_{i+1}^a − d̂_a};
11     return C;
```

W liniach 2–4 pokazano fazę pierwszą algorytmu. Pierwszym krokiem jest ustawienie czasu zakończenia wykonywania pierwszej operacji $C_i^a = p_1^1$, ponieważ wspomniana operacja nie posiada poprzedników maszynowych ani technologicznych. Następnie można ustalić minimalny czas zakończenia wykonywania pozostałych operacji na pierwszej maszynie uwzględniając jedynie czas ich wykonywania oraz minimalny czas przestoju na tej maszynie. W ten sposób będzie spełnione organicznie (4.4) dla tej maszyny. Ograniczenie (4.5) nie dotyczy pierwszej maszyny, ponieważ wszystkie operacje na tej maszynie nie posiadają poprzedników technologicznych. Ustalony w ten sposób harmonogram dla pierwszej maszyny jest dopuszczalny oraz dosunięty w lewo, ponieważ przesunięcie dowolnej operacji w lewo złamałoby ograniczenie (4.4) lub spowodowałoby ujemny czas rozpoczęcia wykonywania pierwszej operacji S_1^1. Na podstawie pseudokodu łatwo zauważyć, że opisana faza jest wykonywana w czasie $O(n)$.

Teraz zostaną omówione fazy $a > 1$ (linie 5 to 10). Każda faza a musi zostać zakończona przed rozpoczęciem fazy $a+1$ (lina 5). Każda faza przebiega w podobny sposób. Najpierw (w linii 6) należy wstępnie przypisać czas zakończenia wykonywania pierwszej operacji na maszynie C_1^a, analogicznie jak w linii 2, ale tym razem z uwzględnieniem poprzednika technologicznego. Następnie w liniach 7–8 trzeba wstępnie ustawić czas zakończenia wykonywania kolejnych operacji na tej maszynie biorąc pod uwagę poprzednik technologiczny, poprzednika maszynowy i minimalny czas przestoju $\hat{r}_a$.

Po wstępnym ustawieniu operacji, część z nich może naruszać prawą stronę ograniczenia (4.4). W celu naprawienia harmonogramu należy w liniach 9–10 wykonać analogiczną procedurę jak w liniach 7-8, ale tym razem w odwrotnej kolejności (od operacji $n-1$ do 1). W praktyce jeśli czasy C_i^a oraz C_{i+1}^a łamią organicznie (4.4) to należy przesunąć czas C_i^a w prawo na tyle, aby spełniał ograniczenie, po czym wartość C_i^a jest już ostateczna. Podczas tej procedury część operacji może zostać przełożona, aby zaczynały i kończyły się później, i taka zmiana nie naruszy innych ograniczeń. Ponadto zostanie zagwarantowane ograniczenie (4.4).

Harmonogram na maszynie a jest dopuszczalny oraz dosunięty w lewo. Jeśli żadna operacja w procedurze korekty (linie 9–10) nie została przesunięta w harmonogramie na później (w prawo), to nie można przesunąć żadnej operacji na wcześniej (w lewo) aby nie naruszać lewej strony ograniczenia (4.4) lub ograniczenia (4.5). Analogicznie, jeśli operacja została przesunięta podczas korekty, to przesunięcie było najmniejsze z takich, które spełniały prawą stronę ograniczenia (4.4). Przesunięcie dowolniej operacji w lewo w harmonogramie nie jest możliwe i skutkowałoby wygenerowaniem harmonogramu niedopuszczalnego.

W załączonym Algorytmie 8 zauważyć można, że każda faza $a > 1$ może zostać zakończona w czasie $O(n)$. Zatem wszystkie m faz można obliczyć w czasie $O(mn)$, pozwalając na uzyskanie dopuszczalnego oraz dosuniętego w lewo harmonogramu dla dowolnego rozwiązania π z $C_{\max} = C_n^m$.

4.1.3 Algorytmy optymalizacyjne

W tym rozdziale zostaną przedstawione następujące metody rozwiązywania problemu $FP|\texttt{minimal idle, maximal idle}|C_{\max}$: (1) Metoda Podziału i Ograniczeń (B&B) (2) Przeszukiwanie Snopowe, (3) Przeszukiwanie z Zabronieniami oraz (4) Sztuczna Kolonia Pszczół. Metoda dokładna B&B posłuży jako odniesienie do oceny jakości metod przybliżonych.

Metoda Podziały i Ograniczeń

Metoda Podziału i Ograniczeń (*Branch and Bound*, B&B) [12] w zastosowaniu do minimalizacji polega na efektywnym przeglądaniu drzewa rozwiązań i „odcinaniu" części gałęzi, w których mamy gwarancje, że nie znajduje się optymalne rozwiązanie. W każdym etapie algorytmu, w momencie podjęcia decyzji, należy ocenić potencjał danej gałęzi. Służy do tego dolne ograniczenie (*Lower Bound*, LB). Wartość najlepszego rozwiązania w danej gałęzi nie może być lepsza niż lepsza niż wartość LB (może być równa lub gorsza).

Odcięcie gałęzi następuje w momencie jeśli wartość LB jest większa niż wartość górnego ograniczenia (*Upper Bound*, UB). W praktyce za wartość UB przyjmowana jest wartość funkcji celu najlepszego znanego do tej pory rozwiązania. Dodatkowo, w celu przyśpieszenia algorytmu B&B, wartość początkowa UB wylicza się jeszcze przed rozpoczęciem przeglądu drzewa w oparciu o dodatkowe algorytmy. Zazwyczaj stosuje w tym celu rozwiązanie uzyskane algorytmem zachłannym, w tym wypadku algorytmem NEH [65].

Zaproponowania w niniejszej dysertacji implementacja B&B opiera się na iteracyjnym przeglądzie węzłów umieszczonych w kolejce priorytetowej. Każdy element kolejki ν zawiera: (1) rozwiązanie częściowe (ν_π), (2) zbiór ν_N zadań które nie zostały jeszcze uszeregowane oraz (3) oraz wartość dolnego ograniczenia $LB(\nu)$. Dolne ograniczenie oszacowano według wzoru:

$$LB(\nu) = \max_{a \in \mathcal{M}} \left\{ C^a_{|\mathcal{J} \setminus \nu_\mathrm{N}|} + \sum_{i \in \nu_\mathrm{N}} (\hat{r}_a + p^a_i) + \sum_{b=a+1}^{m} \min_{i \in \mathcal{J}} p_{b,i} \right\}, \tag{4.13}$$

gdzie pierwsza cześć odpowiada za czas zakończenia wykonywania zadań już uszeregowanych w rozwiązaniu częściowym, druga za operację na a-tej maszynie, trzecia za operacje na maszynach późniejszych.

Przeszukiwanie Snopowe

Metoda B&B jest dokładna, ale jej czas działania jest bardzo często nieakceptowalnie długi, szczególnie dla dużych instancji. Powstało wiele modyfikacji B&B [132, 139], które co prawda nie dają gwarancji uzyskania rozwiązania optymalnego, ale ich czas działania jest dużo krótszy dodatkowo z możliwością jego kontroli.

Jedną z metod przybliżonych bazujących na idei B&B jest tzw. poszukiwanie snopowe (*Beam Search*, BS) [113, 127]. Główną różnicą względem schematu B&B jest moment dodania nowych węzłów-dzieci do kolejki. Metoda B&B dodaje zawsze wszystkie takie węzły, zaś metoda BS dodaje tylko określoną liczbę węzłów, która jest parametrem algorytmu. Zwykle do dodania wybiera się najbardziej obiecujące węzły (z najmniejszą wartością LB). Pozostałe węzły są odrzucane.

W eksperymentach ograniczono liczbę sprawdzanych węzłów do 33% najlepszych na danym poziomie drzewa. Ponadto jako dodatkowy warunek zatrzymania, wprowadzono limit czasowy w celu łatwiejszego porównania z innymi metodami przybliżonymi.

Przeszukiwanie z Zabronieniami

Przeszukiwanie z Zabronieniami (*Tabu Search*, TS) [16] jest zaawansowaną wersją metody poszukiwania lokalnego zaproponowany przez F. Glovera. W celu znalezienia najlepszego sąsiada jest przeglądane całe sąsiedztwo. Sąsiedztwo $\mathcal{N}(\pi)$ rozwiązania π jest zdefiniowane jako wszystkie rozwiązania π', które można otrzymać z π poprzez predefiniowaną funkcją *ruchu*, w tym wypadku zamiany wartości $\pi_a(i)$ oraz $\pi_a(j)$, gdzie $a \in \mathcal{M}$, $i \in \mathcal{J}$ oraz $j \in \mathcal{J} \setminus \{i\}$.

Najlepszy sąsiad może być gorszy niż aktualne rozwiązanie, dlatego w celu zapobiegnięcia wejścia w nieskończony cykl, w algorytmie implementuję się pamięć krótkotrwałą, tak zwaną listę zakazów. Na liście przechowuję się zazwyczaj atrybuty ruchu, przy pomocy którego został wygenerowany najlepszy sąsiad (z pewnymi wyjątkami: tzw. kryterium aspiracji). Rozwiązania uzyskane poprzez zabronione ruchy są ignorowane przy wyborze najlepszego sąsiada. W celu szybkiego przeglądania listy zakazów należy zaimplementować ją w formie dwuwymiarowej macierzy o wymiarach $n \times n$, gdzie wartość w macierzy będzie zawierać informację od której iteracji algorytmu dany ruch będzie już dostępny.

Algorytm wyposażono dodatkowo w mechanizm restartu, jeśli przez określoną liczbę iteracji nie została osiągnięta poprawa. Atrybuty ruchów są przechowywane przez n iteracji na liście zakazów. Jako warunek zatrzymania algorytmu zastosowano limit czasowy.

Sztuczna Kolonia Pszczół

Sztuczna Kolonia Pszczół (*Artificial Bee Colony*, ABC) [67] jest metodą bazującą na zachowaniu pszczół miodnych. Robotnice (*employed bees*) po wyszukaniu źródła nektaru przekazują informacje o nim (azymut, odległość od ula, jakość lub wielkość) innym pszczołom z wykorzystaniem serii akrobacji (tańca). Inne pszczoły skupiają się na „zareklamowanych" miejscach. Znajdywane są również nowe źródła nektaru. Całość pozwala na zmaksymalizowanie zbiorów nektaru.

Metoda ABC jest algorytmem populacyjnym [114] opartym o inteligencję roju [89]. Schemat metody przedstawiono w Algorytmie 9. Na samym początku (linia 2) należy wygenerować populacje początkową. Poprzez populację rozumiemy zbiór rozwiązań, w tym wypadku pożywienie (nektar), które będzie odwiedzane przez pszczoły. Pszczoły formalnie dzieli się na 3 grupy: robotnicy (*employes*), obserwatorów (*onlookers*) i zwiadowców (*scouts*). Liczba pszczół w każdej grupie odpowiada liczbie rozwiązań.

Algorytm 9: Sztuczna Kolonia Pszczół

Dane: maxIter: liczba iteracji/pokoleń, size: rozmiar populacji
Wynik: π^*: najlepsze znalezione rozwiązanie.

```
1  function ArtificialBeeColony(maxIter,size):
2      P_it ← GenerateInitPopulation(size);
3      Evaluate(P_it);
4      while it < maxIter do
5          EmployedBeesPhase(P_it);
6          CalculateProbability(P_it);
7          OnlookersBeesPhase(P_it);
8          ScoutBeesPhase(P_it);
9          π' ← arg min_{π∈P_it} f_max(π);
10         if f_max(π') < f_max(π*) then
11             π* ← π';
12         it ← it + 1;
13     return π*;
```

Algorytm bazuje na lokalnym przeszukiwaniu oraz dzieli się na trzy główne fazy widocznie w liniach 5-8, gdzie każda odpowiada innej grupie pszczół. W fazie pierwszej jest wysyłany dokładnie jeden robotnik do jednego rozwiązania. Robotnik ma za zadanie wygenerować losowego sąsiada, w tym wypadku przy użyciu ruchu zamień. Faza obserwatorów jest podobna do fazy robotników, jednak istotną różnica jest wymiana informacji (tak zwany taniec pszczół) między tymi dwiema fazami. Każdy obserwator losuje na podstawie metody ruletki (prawdopodobieństwo wyboru zależy od jakości rozwiązania) rozwiązanie, którym będzie się zajmował. Kilku obserwatorów może wybrać to samo rozwiązanie. W obu fazach lepsze rozwiązanie jest zawsze akceptowane, w przeciwnym wypadku rozwiązanie jest odrzucane, a dla danego źródła jest zwiększany licznik. W ostatniej fazie zwiadowcy są wysyłani do znalezienia nowych rozwiązań w miejsce rozwiązań, które mają przekroczony limit.

W testach ograniczono ustaloną stałą liczbę pszczół w każdej grupie oraz wielkość populacji równą 10. Limit zmian bez poprawy dla danego rozwiązania wynosił 30. Jako warunek zatrzymania, wprowadzono limit czasowy w celu łatwiejszego porównania z innymi metodami przybliżonymi.

Tablica 4.2: Średnie czasy działania T (metod B&B, BS, TS oraz ABC, w sekundach) oraz średnie PRD (metod BS, TS oraz ABC) dla instancji ze zbioru A

$n \times m$	$T_{\text{B\&B}}$	PRD_{BS}	T_{BS}	PRD_{TS}	T_{TS}	PRD_{ABC}	T_{ABC}
6×3	0,00	0,23	0,00	2,09	0,20	0,00	0,20
6×5	0,00	1,50	0,00	4,29	0,20	0,00	0,20
8×3	0,03	0,00	0,00	0,00	0,20	0,00	0,20
8×5	0,04	0,35	0,00	0,31	0,20	0,00	0,20
10×3	3,77	0,03	0,04	0,00	0,20	0,00	0,20
10×5	4,13	1,17	0,08	0,74	0,20	0,00	0,20
12×3	78,39	0,03	0,15	0,04	0,20	0,00	0,20
średnia	-	0,47	-	1,07	-	0,00	-

4.1.4 Eksperymenty obliczeniowe

Celem eksperymentów było porównanie efektywności metod przybliżonych: (1) Przeszukiwania Snopowego, (2) Przeszukiwania z Zabronieniami oraz (3) Sztucznej Koloni Pszczół, w zastosowaniu do rozważanego problemu ze sprzężeniami czasowymi. Przygotowano dwa zestawy instancji testowych A i B, wygenerowanych przy użyciu generatora instancji zaproponowanego przez Taillarda [129]. Zbiór A zawierał instancje o niewielkich rozmiarach, które zostały użyte do przetestowania skuteczności algorytmów przybliżonych względem metody dokładnej B&B. Zestaw B składał się z instancji o większych rozmiarach, został wykorzystany do testowania metaheurystyk w bardziej praktycznych warunkach. Dla każdego rozmiaru wygenerowano po 10 instancji.

Jako miarę jakości zastosowano procentowe względne odchylenie (*Percentage Relative Deviation*, PRD). Niech π oznacza rozwiązanie uzyskane przez badany algorytm dla konkretnej instancji oraz π^* rozwiązane referencyjne, które zazwyczaj jest najlepszym znanym rozwiązaniem. Wtedy PRD definiuje się następująco.

$$\text{PRD} = \frac{C_{\max}(\pi) - C_{\max}(\pi^*)}{C_{\max}(\pi^*)} \cdot 100\%. \tag{4.14}$$

Wszystkie eksperymenty zostały przeprowadzone na komputerze z procesorem Intel Core i7-6700K 4,0 GHz oraz 16 GB pamięci RAM. Program został napisany języku w `C++`, skompilowany przy użyciu IDE Microsoft Visual Studio 2019 pod systemem MS Windows 10.

W pierwszym eksperymencie zostały porównane wszystkie cztery metody dla instancji z zestawu A. Wyniki zostały przedstawione w Tabeli 4.2. Dla wszystkich instancji danego rozmiaru warunkiem zatrzymania algorytmu TS oraz ABC był limit czasowy w tabeli oznaczony odpowiednio jako T_{TS} oraz T_{ABC}. Przez $T_{\mathrm{B\&B}}$ oraz T_{BS} oznaczono czas działania odpowiednio algorytmów B&B oraz BS. Na podstawie analizy wyników można zaobserwować, że zarówno czas działania B&B i BS rośnie wraz z rozmiarem instancji, w tym dla BS znacząco wolniej. Metody przybliżone dla instancji o rozmiarze większym niż 8 zadań znalazły bardzo dobre jakościowo rozwiązania w znacząco krótszym czasie (poniżej 0,2 s) niż metoda dokładna. Algorytm BS mimo odrzucania średnio 30% rozwiązań na każdym poziomie drzewa decyzyjnego znalazł dla wszystkich instancji o rozmiarze 8×3 rozwiązania optymalne, a w pozostałych przypadkach średnio tylko 0,47% gorsze od rozwiązań optymalnych. Metodą TS znaleziono zawsze rozwiązania optymalne w przypadku wszystkich instancji w grupach rozmiarów 8×3 i 10×3, a w pozostałych przypadkach średnio tylko 1,07% gorsze od rozwiązania optymalnego. Metaheurystyka ABC dla badanych instancji zawsze znalazła rozwiązanie optymalne w wyznaczonym limicie czasowym. W badanym przypadku jako rozwiązanie referencyjne π^* było przyjęte rozwiązanie optymalne dostarczone przez metodę dokładną B&B,

W drugim eksperymencie zostały porównane trzy różne metody przybliżone dla zbioru instancji B. Warunkiem zatrzymania algorytmu we wszystkich trzech metodach był limit czasowy. Ponadto rozwiązanie π^* zostało wybrane jako najlepsze z trzech badanych metod dla danej instancji w danym interwale czasowym ($10\ s$ lub $60\ s$). Wyniki przedstawiono w Tabeli4.3.

Warto zwrócić uwagę na to, że w największej liczbie grup (7) najlepsze rozwiązanie zawsze zostało znalezione przez algorytm ABC. Ponadto, wyniki średnie PRD dla ABC i TS były bardzo zbliżone (poniżej 0,1%). Największe PRD wyszło dla algorytmu BS, ale dalej średnio poniżej 3%. Przedstawione wyniki wskazują, że badane metody generują zbliżone rezultaty w tym samym czasie. Ponadto sześciokrotne zwiększenie czasu nieznacznie wpłynęło na różnice między jakością znajdowanych rozwiązań.

4.1.5 Wnioski i uwagi

W tym rozdziale został przedstawiony permutacyjny problem przepływowy ze sprzężeniami czasowymi dla kryterium czasu zakończenia wykonywania wszystkich zadań. Opisany problem bazuje na rzeczywistym procesie betonowania. Sformułowano model matematyczny oraz pewne nowe własności oparte na reprezentacji grafowej, w tym zaproponowano szybką metodę liczenia funkcji celu w czasie $O(nm)$.

Tablica 4.3: Średnie PRD oraz czas działania T (w sekundach) dla metod przybliżonych BS, TS i ABC

$n \times m$	$T = 10$			$T = 60$		
	$\mathrm{PRD}_{\mathrm{BS}}$	$\mathrm{PRD}_{\mathrm{TS}}$	$\mathrm{PRD}_{\mathrm{ABC}}$	$\mathrm{PRD}_{\mathrm{BS}}$	$\mathrm{PRD}_{\mathrm{TS}}$	$\mathrm{PRD}_{\mathrm{ABC}}$
25 × 5	2,95	0,01	0,00	2,95	0,07	0,00
25 × 15	5,17	0,05	0,14	5,27	0,18	0,17
50 × 5	3,51	0,03	0,00	3,51	0,14	0,00
50 × 15	3,37	0,17	0,11	3,48	0,17	0,06
100 × 5	2,70	0,04	0,00	2,71	0,09	0,00
100 × 15	2,71	0,04	0,18	2,79	0,04	0,17
200 × 5	1,08	0,00	0,00	1,08	0,00	0,00
200 × 5	1,12	0,00	0,09	1,14	0,00	0,04
średnia	2,83	0,04	0,07	2,87	0,09	0,05

Pokazano cztery metody rozwiązania: dokładną Metodę Podziału i Ograniczeń oraz trzy metody przybliżone: (1) Poszukiwanie Snopowe, (2) Przeszukiwanie z Zabronieniami oraz (3) Sztuczną Kolonię Pszczół. Eksperymenty komputerowe na instancjach bazujących na generatorze Taillarda wykazały podobne zachowanie algorytmów w tym samym czasie, więc metody można stosować wymiennie.

4.2 Niepermutacyjny problem przepływowy

W niniejszym rozdziale skupiono się na niepermutacyjnym problemie przepływowym ze sprzężeniami czasowymi. Zostanie sformułowany model matematyczny oraz grafowy. Następnie będą zidentyfikowane nowe własności, w szczególności własności eliminacyjne oparte na blokach. Zaproponowane zostaną dwie metody rozwiązania: (1) Metoda Podziału i Ograniczeń oraz (2) Przeszukiwanie z Zabronieniami uwzględniające własności blokowe. Dalej zostaną omówione wyniki eksperymentów.

4.2.1 Definicja problemu

Ponownie przez $\mathcal{J}$, $\mathcal{M}$ oraz $\mathcal{O}$ oznaczono odpowiednio zbiory n zadań, m maszyn oraz mn operacji. Każde zadanie i musi zostać wykonane na maszynach w kolejności technologicznej: $1 \to 2 \to \cdots \to m$. Czas wykonywania zadania i na maszynie a został oznaczony przez $p_i^a > 0$. Ponadto,

niech $\hat{r}_a \geqslant 0$ oraz $\hat{d}_a \geqslant \hat{r}_a$ oznaczają odpowiednio minimalny i maksymalny czas przestoju (*idle*) maszyny a dozwolony między przetwarzaniem kolejnych operacji na tej maszynie.

Kolejność wykonywania operacji w ramach każdego zadania jest stała, ponieważ wynika z ograniczeń technologicznych problemu. Jednak kolejność wykonywania poszczególnych zdań (operacji z zadania na konkretnej maszynie) na różnych maszynach może być inna. Przez π_a oznaczono n-elementową kolejność zadań wykonywanych na maszynie a. Zatem $\pi_a(i)$ oznacza i-te zadanie wykonywane w kolejności π_a na maszynie a. Rozwiązaniem będzie nazywana sekwencja kolejności na wszystkich maszynach $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_m)$.

Na podstawie rozwiązania $\boldsymbol{\pi}$ można zbudować odpowiadający mu *harmonogram* $(\boldsymbol{S}, \boldsymbol{C})$. Harmonogram składa się z dwóch macierzy: (1) macierzy momentów rozpoczęcia wykonywania operacji $\boldsymbol{S} = [S_i^a]^{m \times n}$, gdzie S_i^a jest czasem rozpoczęcia wykonywania i-tej operacji na maszynie a oraz (2) macierzy momentów zakończenia wykonywania operacji $\boldsymbol{C} = [C_i^a]^{m \times n}$, gdzie C_i^a jest czasem zakończenia wykonywania i-tej operacji na maszynie a. Aby harmonogramu $(\boldsymbol{S}, \boldsymbol{C})$ był dopuszczalny, musi spełniać następujące ograniczenia:

$$\forall a \in \mathcal{M} \;\; \forall i \in \mathcal{J} \quad S_i^a, C_i^a \geqslant 0 \tag{4.15}$$

$$\forall a \in \mathcal{M} \;\; \forall i \in \mathcal{J} \quad C_i^a = S_i^a + p_i^a, \tag{4.16}$$

$$\forall a \in \mathcal{M} \;\; \forall i \in \mathcal{J} \setminus \{1\} \quad \hat{r}_a \leqslant S_i^a - C_{i-1}^a \leqslant \hat{d}_a, \tag{4.17}$$

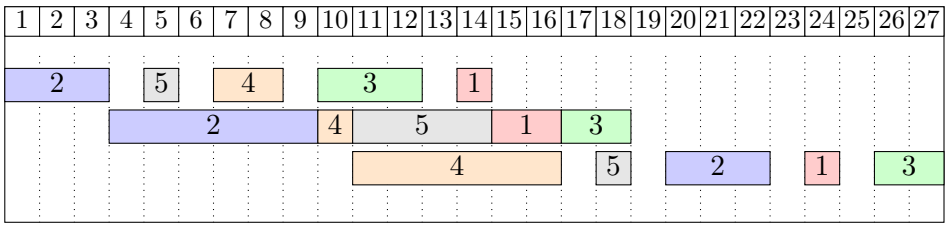
$$\forall a \in \mathcal{M} \setminus \{1\} \;\; \forall i \in \mathcal{J} \quad S_i^a \geqslant C^{a-1}_{\pi_{a-1}^{-1}(\pi_a(i))}, \tag{4.18}$$

gdzie $\pi_a^{-1}(i)$ oznacza pozycje operacji z zadania i-tego w kolejności π_a, czyli: $\pi_a^{-1}(j) = i \iff \pi_a(i) = j$. Bez utraty ogólności można przyjąć czas rozpoczęcia pierwszej operacji z zadania pierwszego w kolejności na pierwszej maszynie jako 0, czyli $S_1^1 = 0$. Ograniczenie (4.16) zapewnia ciągłość wykonywania operacji. Ograniczenie (4.17) wymusza przestrzeganie minimalnego oraz maksymalnego czasu przestoju maszyny, ale również wymaga, aby operacja i-ta w kolejności była wykonywana dopiero po zakończeniu poprzedniej operacji $i-1$ (poprzednik maszynowy) na tej samej maszynie a. Podobnie, ograniczenie (4.18) wymusza rozpoczęcie operacji i-tej na maszynie a dopiero po zakończeniu operacji i-tej na maszynie poprzedniej $a-1$ (poprzednika technologicznego).

Harmonogram będzie nazywany w lewo dosuniętym (*left-shifted*), jeśli żadne operacje nie mogą rozpocząć wykonywać się wcześniej bez naruszenia ograniczeń lub zmiany kolejności wykonywania zadań. W Pod-

Tablica 4.4: Instancja problemu FSSP-TC z Przykładu 4.2

a	p_1^a	p_2^a	p_3^a	p_4^a	p_5^a	$\hat{r}_a$	$\hat{d}_a$
1	1	3	3	2	1	1	5
2	2	3	2	1	4	0	0
3	1	3	2	6	1	1	2



Rysunek 4.3: Harmonogram dla rozwiązania $\boldsymbol{\pi}$ z Przekładu 4.2

rozdziale 4.2.2 oraz 4.2.5 pokazano metodę wyznaczania harmonogramu dosuniętego w lewo odpowiednio w wersji sekwencyjnej oraz równoległej.

Przykład 4.2 W Tabeli 4.4 pokazano instancje problemu FSSP-TC. Na Rysunku 4.3 przedstawiono w lewo dosunięty harmonogram dla tej instancji oraz permutacji $\boldsymbol{\pi} = ((2,5,4,3,1),\ (2,4,5,1,3),\ (4,5,2,1,3))$.

Badanym kryterium optymalizacyjnym będzie długość uszeregowania, czyli czas zakończenia wykonywania wszystkich operacji. Przez $C_{\max}(\boldsymbol{\pi})$ oznaczono czas zakończenia operacji, która zakończyła się wykonywać jako ostatnia w harmonogramie dosuniętym w lewo dla kolejności $\boldsymbol{\pi}$. Z ograniczeń (4.16)–(4.18) wynika, że $C_{\max} = C_n^m$. Celem będzie znalezienie takiego rozwiązania $\boldsymbol{\pi}$, że:

$$C_{\max}(\boldsymbol{\pi}^{\text{opt}}) = \min_{\boldsymbol{\pi} \in \Pi} C_{\max}(\boldsymbol{\pi}), \tag{4.19}$$

gdzie Π jest zbiorem wszystkich dopuszczalnych rozwiązań. Rozwiązanie π^{opt} będzie nazywane rozwiązaniem optymalnym.

Według rozszerzonej notacji Grahama [57] problem FSSP-TC jest oznaczany jako $F|\texttt{minimal idle, maximal idle}|C_{\max}$. Za pomocą proponowanego problemu, analogicznie jak to było w wypadku problemu PFSSP-TC, można rozważać klasyczne warianty problemu FSSP poprzez odpowiedni dobór parametrów $\hat{r}_a$ oraz $\hat{d}_a$. Proponowane metody i własności można skutecznie stosować między innymi dla następujących problemów: $F||C_{\max}$, $FP|\texttt{no idle}|C_{\max}$, $F|\texttt{minimal idle}|C_{\max}$ oraz $F|\texttt{maximal idle}|C_{\max}$.

Model grafowy

Jedną z klasycznych technik szeregowania zadań jest budowa modelu grafowego dla danego rozwiązania π. Należy utworzyć graf rozwiązania $G(\pi) = (V,\ E_1 \cup E_2)$, gdzie:

$$V = \bigcup_{\substack{a\in\mathcal{M}\\ i\in\mathcal{J}}} \Big\{(a,i)\Big\}. \tag{4.20}$$

Przez V został oznaczony zbiór wierzchołków reprezentujących operacje (dla uproszczenia wizualizacji ustawione w formie siatki). Operacja z zadania i-tego w kolejności na maszynie a jest reprezentowana przez obciążony wierzchołek w i-tej kolumnie oraz a-tym wierszu oznaczony parą (a,i) z wagą p_i^a.

$$E_1 = \bigcup_{\substack{a\in\mathcal{M}\\ i\in\mathcal{J}\setminus\{n\}}} \Big\{\Big((a,i),\ (a,i+1)\Big)\Big\}, \tag{4.21}$$

$$E_2 = \bigcup_{\substack{a\in\mathcal{M}\setminus\{1\}\\ i\in\mathcal{J}}} \Big\{\Big((a-1,\pi_{a-1}^{-1}(\pi_a(i))),\ (a,i)\Big)\Big\}. \tag{4.22}$$

Łuki grafu $G(\pi)$ reprezentują ograniczenia problemu: (1) łuki poziome E_1 – ograniczenia maszynowe oraz (2) łuki pionowe E_2 – ograniczenia technologiczne. Długość najdłużej ścieżki do danego wierzchołka (a,i) jest równa odpowiedniemu czasowi zakończenia C_i^a. Przez długość ścieżki rozumie się sumę wag wierzchołków i krawędzi należących do ścieżki. Przez $\mathrm{len}(p)$ oznaczono długość ścieżki p.

Kolejnym krokiem jest rozszerzenie grafu $\mathcal{G}(\pi)$. Minimalny czas przestoju maszyny został odzwierciedlony analogicznie do czasów przezbrojeń [17]. Każdemu łukowi „poziomemu" $((a,i),\ (a,i+1))$ w zbiorze E_1 przypisano wagę $\hat{r}_i$. Po drugie, aby odzwierciedlić maksymalny czas przestoju maszyny należy wprowadzić nowy zbiór łuków:

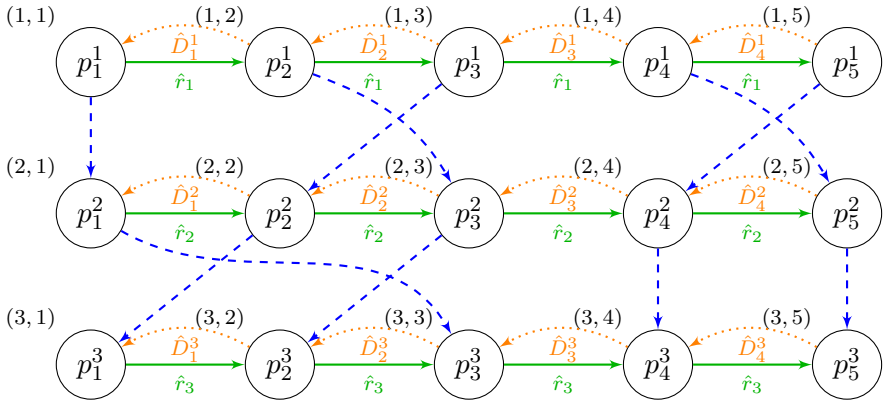
$$E_3 = \bigcup_{\substack{a\in\mathcal{M}\\ i\in\mathcal{J}\setminus\{n\}}} \Big\{\Big((a,i+1),\ (a,i)\Big)\Big\}. \tag{4.23}$$

Każdy łuk „powrotny" $((a,i+1),\ (a,i))$ w zbiorze E_3 ma wagę $\hat{D}_i^a$, zdefiniowaną jako:

$$\hat{D}_i^a = -\rho_{a,i} - \rho_{a,i+1} - \hat{d}_a. \tag{4.24}$$

Strukturę grafu $\mathcal{G}(\boldsymbol{\pi})$ dla problemu o rozmiarze 5×3 oraz kolejności $\boldsymbol{\pi} = ((2,5,4,3,1),\ (2,4,5,1,3),\ (4,5,2,1,3))$ przedstawiono na Rysunku 4.4.

Dla ułatwienia czytelności łuki E_1, E_2 oraz E_3 oznaczono kolejno zieloną linią ciągła, niebieską przerywaną oraz pomarańczowa kropkowaną.



Rysunek 4.4: Przykładowy graf $\mathcal{G}(\boldsymbol{\pi})$ dla problemu FSSP-TC

4.2.2 Własności problemu

Poniżej zostaną przedstawione oraz omówione nowe własności problemu $F|\texttt{minimal idle, maximal idle}|C_{\max}$ w tym: dopuszczalność rozwiązań, złożoność obliczeniowa metody wyznaczania wartości funkcji celu oraz własność eliminacyjną opartą na blokach.

Własność 4.2 *Dla dowolnego rozwiązania $\boldsymbol{\pi} \in \Pi$, graf rozwiązania $\mathcal{G}(\boldsymbol{\pi})$ dla problemu $F|\texttt{minimal idle, maximal idle}|C_{max}$ nie zawiera cykli o dodatniej długości.* □

Dowód. Dowód jest analogiczny jak dowód Własności 4.1 dla problemu $FP|\texttt{minimal idle, maximal idle}|C_{\max}$ na stronie 60, ponieważ również w tym wypadku graf nie zawiera krawędzi idących w górę. Podobnie w tym przypadku cykl może powstać tylko za pomocą łuków „poziomych" lub „powrotnych", których struktura jest niezmienna. ■

W związku z tym, wszystkie cykle w grafie $\mathcal{G}(\boldsymbol{\pi})$ mają długość niedodatnia. Obliczając czas zakończenia wszystkich zadań, będzie można zignorować cykle zgodnie z następującą obserwacją.

Wniosek 4.2 Dla każdego rozwiązania $\boldsymbol{\pi}$, jeśli graf rozwiązania $\mathcal{G}(\boldsymbol{\pi})$ dla $F|\texttt{minimal idle, maximal idle}|C_{\max}$ zawiera ścieżkę p o długości $\text{len}(p)$

Algorytm 10: Konstruowanie harmonogramu dosuniętego w lewo

Dane: $\boldsymbol{\pi}$: kolejność wykonywania zadań na wszystkich maszynach
Wynik: $\boldsymbol{C}$: macierz momentów zakończenia operacji

1 **function** `Evaluate(`$\boldsymbol{\pi}$`):`
2 $\quad C_1^1 \leftarrow p^1_{\pi(1)}$;
3 $\quad$ **for** $i = 2, 3, \ldots, n$ **do**
4 $\quad\quad C_i^1 \leftarrow C_{i-1}^1 + \hat{r}_0 + p^1_{\pi(i)}$;
5 $\quad$ **for** $a = 2, 3, \ldots, m$ **do**
6 $\quad\quad C_1^a \leftarrow C^{a-1}_{\pi_{a-1}^{-1}(\pi_a(1))} + p^a_{\pi(1)}$;
7 $\quad\quad$ **for** $i = 2, 3, \ldots, n$ **do**
8 $\quad\quad\quad C_i^a \leftarrow \max\{C_{i-1}^a + \hat{r}_a, C^{a-1}_{\pi_{a-1}^{-1}(\pi_a(i))}\} + p^a_{\pi(i)}$;
9 $\quad\quad$ **for** $i = n-1, n-2, \ldots, 1$ **do**
10 $\quad\quad\quad C_i^a \leftarrow \max\{C_i^a, C_{i+1}^a - p^a_{\pi(i+1)} - \hat{d}_a\}$;
11 $\quad$ **return** $\boldsymbol{C}$;

takiej, że p zawiera cykl, to graf $\mathcal{G}(\boldsymbol{\pi})$ zawiera również ścieżkę p' bez cyklu, taką że:

$$\text{len}(p') \geqslant \text{len}(p). \tag{4.25}$$

□

Zatem, jeśli ścieżka p w grafie $\mathcal{G}(\boldsymbol{\pi})$ zawiera cykl, to: (1) nie jest to najdłuższa ścieżka w grafie $\mathcal{G}(\boldsymbol{\pi})$ (przynajmniej jeden cykl w ścieżce p ma ujemną długość) lub (2) możemy skonstruować inną ścieżkę zaczynającą się i kosząca dokładnie w tych samych wierzchołkach, ale niezawierającej cyklu (wszystkie cykle w ścieżce p mają długość zero).

Twierdzenie 4.2 *Dla każdego rozwiązania $\boldsymbol{\pi} \in \Pi$ niepermutacyjnego problemu $F|$`minimal idle, maximal idle`$|C_{max}$, istnieje dopuszczalny oraz dosunięty w lewo harmonogram $(\boldsymbol{S}, \boldsymbol{C})$. Ponadto, wartość kryterium $C_{max}(\boldsymbol{\pi})$ dla harmonogramu $(\boldsymbol{S}, \boldsymbol{C})$ można wyznaczyć w czasie $O(nm)$.* □

Dowód. Dowód jest analogiczny jak dowód Twierdzenia 4.2 dla problemu $FP|$`minimal idle, maximal idle`$|C_{\max}$. Należy jednak uwzględnić inne liczenie poprzedników technologicznych zgodnie z ograniczaniem (4.18). Modyfikację algorytmu dla $F|$`minimal idle, maximal idle`$|C_{\max}$ pokazano w Algorytmie 10, dokładnie w linii 6 oraz 8. Zmiana nie wpływa na złożoność obliczeniową algorytmu jeśli wykorzysta się dodatkowo $O(mn)$ pamięci do przechowywania pozycji poprzedników przy użyciu permutacji

odwrotnej pokazanej w Przykładzie 4.3. Czas wyznaczenia od zera permutacji odwrotnej również wynosi $O(mn)$.

Wniosek 4.3 Dla dowolnej kolejności wykonywania $\pi \in \Pi$ dla problemu $F|\texttt{minimal idle, maximal idle}|C_{\max}$ istnieje co najmniej jeden dopuszczalny harmonogram $(\boldsymbol{S}, \boldsymbol{C})$ (zatem wszystkie kolejności są dopuszczalne).□

Dowód. Metodę liczenia harmonogramu przedstawioną w Twierdzeniu 4.2 można zastosować dla dowolnej kolejności π, ponieważ przy zmianie kolejności π zmianie ulegają są jedynie aktualnie wartości p_i^a, które są używanie do ustalania czasów C_i^a bazujących na aktualnych wartościach S_i^a. W ten sposób przy użyciu algorytmy tworzony jest dopuszczalny harmonogram $(\boldsymbol{S}, \boldsymbol{C})$ dla dowolnego π. ■

Przykład 4.3 W Tabeli 4.5 pokazano reprezentację permutacji $\boldsymbol{\pi}$. Dla $\boldsymbol{\pi} = ((2,5,4,3,1),\ (2,4,5,1,3),\ (4,5,2,1,3))$ w formie dwuwymiarowej tabeli oraz w Tabeli 4.6 reprezentację permutacji $\boldsymbol{\pi}^{-1}$ do niej odwrotnej. □

Tablica 4.5: Reprezentacja permutacji $\boldsymbol{\pi}$ w pamięci komputera

a	$\pi_a(1)$	$\pi_a(2)$	$\pi_a(3)$	$\pi_a(4)$	$\pi_a(5)$
1	2	5	4	3	1
2	2	4	5	1	3
3	4	5	2	1	3

Tablica 4.6: Reprezentacja permutacji odwrotnej $\boldsymbol{\pi}^{-1}$ w pamięci komputera

a	$\pi_a^{-1}(1)$	$\pi_a^{-1}(2)$	$\pi_a^{-1}(3)$	$\pi_a^{-1}(4)$	$\pi_a^{-1}(5)$
1	5	1	4	3	2
2	4	1	5	2	3
3	4	3	5	1	2

W praktyce wiele metod rozwiązywania problemów optymalizacyjnych bazuje na algorytmach przeszukiwania lokalnego, gdzie przestrzeń rozwiązań jest przeszukiwania poprzez modyfikowanie jednego (aktualnego) rozwiązania używając koncepcji `ruchu` (*move*). Ruch typu zamień dla problemu $F|\texttt{minimal idle, maximal idle}|C_{\max}$ można zdefiniować w następujący sposób:

$$\boldsymbol{\pi}' = \mathtt{S}(\boldsymbol{\pi}, a, i, j), \tag{4.26}$$

gdzie rozwiązanie $\boldsymbol{\pi}'$ jest tworzone z $\boldsymbol{\pi}$ przez zamianę wartości $\pi_a(i)$ oraz $\pi_a(j)$. Rozwiązanie $\boldsymbol{\pi}'$ będzie nazwane sąsiadem rozwiązania $\boldsymbol{\pi}$. Podobnie, zbiór wszystkich rozwiązań utworzonych z $\boldsymbol{\pi}$ przez ten ruch będzie nazywany sąsiedztwem $\boldsymbol{\pi}$. We Wniosku 4.3 udowodniono, że wszystkie rozwiązania $\boldsymbol{\pi}$ są dopuszczalne. Jednak nie ma potrzeby sprawdzania wszystkich rozwiązań w celu znalezienia rozwiązania optymalnego.

Poniżej zostanie sformułowana własność eliminacyjna, z której będzie wynikać, że niektóre ruchy nie skutkują natychmiastową poprawą długości uszeregowania. Analogicznie do własności eliminacyjnych bazujących na blokach dla JSSP zaproponowanych przez Grabowskiego [51] i dalej rozwiniętych w pracy [95] przez Nowickiego i Smutnickiego.

Poprzez pojęcie `bloku` (*block*) jest rozumiana sekwencja operacji $B = (b_1, b_2, \ldots, b_c)$ dla $c > 0$. Taka sekwencja będzie nazywana blokiem wtedy i tylko wtedy, gdy spełnione są wszystkie z poniższych warunków:

1. Wszystkie operacje z B znajdują się na tej samej ścieżce krytycznej.
2. Operacje z B są wykonywane pod rząd na maszynie a.
3. Operacje z B są wykonywane na (maszynie) a w kolejności ich występowania w B.
4. Długość B jest maksymalna, tzn. operacje przed i po bloku nie mogą być do niego dodane bez naruszenia pierwszych trzech warunków.

Ponadto dla bloku $B = (b_1, b_2, b_3, \ldots, b_{c-2}, b_{c-1}, b_c)$ oraz $c > 2$ operacji zdefiniowano jego `blok wewnętrzny` (*inner block*) jako sekwencję $B' = (b_2, b_3, \ldots, b_{c-2}, b_{c-1})$.

Niech blok $B = (b_1, b_2, \ldots, b_c)$ będzie częścią ścieżki krytycznej p^c. Blok B będzie nazywany blokiem prawostronnym (lub R-blokiem), jeśli pierwsza operacja z bloku tego na ścieżce p^c to operacja b_1. Analogicznie B będzie nazywany blokiem lewostronnym (lub L-blokiem), jeśli pierwsza operacja z bloku to operacja b_c. Dzięki takiej definicji można zdefiniować blokową własność eliminacyjną dla problemu $F|\texttt{minimal idle, maximal idle}|C_{\max}$.

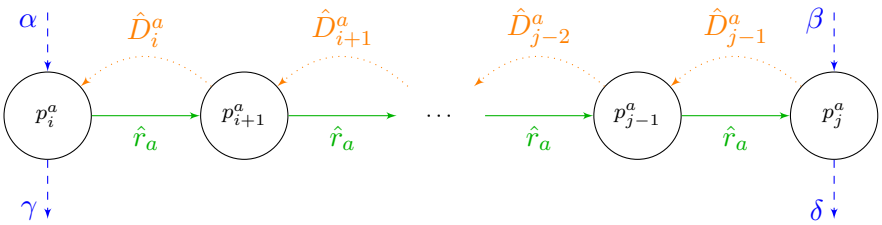
Twierdzenie 4.3 *Niech* $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_m)$ *będzie rozwiązaniem problemu* $F|\texttt{minimal idle, maximal idle}|C_{max}$ *oraz niech* $B = (b_1, b_2, \ldots, b_c)$, $c > 2$ *będzie blokiem prawostronnym na ścieżce krytycznej* p^c *w grafie* $\mathcal{G}(\boldsymbol{\pi})$ *składającym się z operacji na maszynie* a. *Następnie, niech* $\boldsymbol{\pi}'$ *będzie rozwiązaniem utworzonym z* $\boldsymbol{\pi}$ *przez zamianę dowolnej pary operacji wewnątrz bloku* B*:*

$$\boldsymbol{\pi}' = \mathrm{swap}(a, i, j) \quad i, j \in \{b_2, b_3, \ldots, b_{c-2}, b_{c-1}\},\ i \neq j, \tag{4.27}$$

wtedy:

$$C_{\max}(\boldsymbol{\pi}') \geqslant C_{\max}(\boldsymbol{\pi}). \tag{4.28}$$

□



Rysunek 4.5: Ogólna struktura L- oraz R-bloku

Dowód. Na podstawie Własności 4.2 oraz Wniosku 4.2 istnieje co najmniej jedna ścieżka krytyczna bez cyklu, która zawiera blok B. Niech p^c oznacza dokładnie taką ścieżkę krytyczną. Z tego wynika, że ścieżka krytyczna przechodzi przez blok B w jednym kierunku (ścieżka krytyczna nie zmienia kierunku wewnątrz bloku B). Zatem B jest blokiem prawostronnym lub lewostronnym.

Łatwo zauważyć, że łuk „pionowy" wchodzący do bloku B z poprzedniej maszyny (jeśli $a > 1$) zawsze wchodzi do B na jego początku (w operacji b_1) lub na jego końcu (w operacji b_c). Analogicznie jest dla łuków „pionowych" na następną maszynę (dla $a < m$). Dla R-bloku ścieżka krytyczna p^c wchodzi do bloku B z poprzedniej maszyny w operacji b_1 oraz wychodzi na kolejną maszynę w operacji b_c.

Przy założeniu, że operacje $(b_1, b_2, \ldots, b_c)$ z bloku B są odpowiednio operacjami z zadań $i, i+1, \ldots, j-1, j$. Niech α będzie długością najdłuższej ścieżki kończącej się w wierzchołku (a, i), ale bez uwzględnienia wagi tego wierzchołka. Podobnie δ niech będzie długością najdłuższej ścieżki zaczynającej się w wierzchołku (a, j), ale również bez uwzględnienia wagi wierzchołka. Stosowny fragment grafu $\mathcal{G}(\boldsymbol{\pi})$, w tym blok B oraz ścieżki α i δ pokazano na Rysunku 4.5. Długość ścieżki krytyczniej w przypadku R-bloku będzie wynosić:

$$\alpha + \rho_{a,i} + \rho_{a,i+1} + \ldots + \rho_{a,j-1} + \rho_{a,j} + \hat{r}_a(j-i)\hat{r}_a + \delta. \tag{4.29}$$

Wewnętrzny blok dla B to $B' = (b_2, b_3, \ldots, b_{c-1})$. W przypadku zamiany dwóch dowolnych operacji $b_x, b_y \in B'$ takich, że $x \neq y$, taka zamiana nie wpłynie na długość ścieżek α oraz δ. Nie zmieni również wartości p^a_i oraz p^a_j, ponieważ wierzchołki (a, i) oraz (a, j) znajdują się poza blokiem wewnętrznym. Wartości $\hat{r}_a$ również nie ulegną zmianie. Zmiana operacji b_x oraz b_y zmieni kolejność wartości od p^a_{i+1} do p^a_{j-1}, ale ich suma pozostanie niezmieniona. W ten sposób długość rozważanej ścieżki krytycznej pozostanie taka sama. Zatem graf $\mathcal{G}(\boldsymbol{\pi}')$ będzie zawierał ścieżkę o długości co

najmniej takiej, jak długość ścieżki p^c w $\mathcal{G}(\boldsymbol{\pi})$, więc:

$$C_{\max}(\pi') \geqslant C_{\max}(\pi). \tag{4.30}$$

■

Twierdzenie 4.4 *Niech $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_m)$ będzie rozwiązaniem problemu $F|$`minimal idle, maximal idle`$|C_{max}$ oraz niech $B = (b_1, b_2, \ldots, b_c)$, $c > 2$ będzie blokiem lewostronnym na ścieżce krytyczniej p^c w grafie $\mathcal{G}(\boldsymbol{\pi})$ składającym się z operacji na maszynie a. Następnie, niech $\boldsymbol{\pi}'$ będzie rozwiązaniem utworzonym z $\boldsymbol{\pi}$ przez zamianę dowolnej pary operacji wewnątrz bloku B:*

$$\boldsymbol{\pi}' = \text{swap}(a, i, j) \quad i, j \in \{b_2, b_3, \ldots, b_{c-2}, b_{c-1}\},\ i \neq j, \tag{4.31}$$

wtedy:

$$C_{\max}(\boldsymbol{\pi}') \geqslant C_{\max}(\boldsymbol{\pi}). \tag{4.32}$$

□

Dowód. Dowód jest analogiczny jak w przypadku Twierdzenia 4.3 dla bloku prawostronnego. W tym przypadku ścieżka krytyczna p^c przechodzi przez blok B odwrotnie niż dla R-bloku. Ścieżka krytyczna wchodzi do B w operacji b_c (dla $a > 1$), a wychodzi w operacji b_1 (dla $a < m$).

Przy założeniu, że operacje $(b_1, b_2, \ldots, b_c)$ z bloku B są odpowiednio operacjami z zadań $i, i+1, \ldots, j-1, j$, niech β będzie długością najdłuższej ścieżki kończącej się w wierzchołku (a, j), ale bez uwzględnienia wagi tego wierzchołka. Podobnie γ niech będzie długością najdłuższej ścieżki zaczynającej się w wierzchołku (a, i), ale również bez uwzględnienia wagi tego wierzchołka. Stosowny fragment grafu $\mathcal{G}(\boldsymbol{\pi})$, w tym blok B oraz ścieżki βoraz γ pokazano na Rysunku 4.5. Długość ścieżki krytycznej wynosi zatem:

$$\beta + p_i^a + p_{i+1}^a + \ldots + p_{j-1}^a + p_j^a + \hat{D}_i^a + \hat{D}_{i+1}^a + \ldots + \hat{D}_{i-1}^a + \gamma. \tag{4.33}$$

Ponownie w przypadku zamiany dwóch dowolnych operacji $b_x, b_y \in B'$ takich, że $x \neq y$, taka zamiana nie wpłynie na długość ścieżek β oraz γ. Nie zmieni również wartości p_i^a oraz p_j^a, ponieważ wierzchołki (a, i) oraz (a, j) znajdują się poza blokiem wewnętrznym. Zamiana b_x i b_y również nie wpłynie na sumę wag wierzchołków $(a, i+1)$ na $(a, j-1)$, ponieważ wszystkie one nadal będą częścią wewnętrznego bloku po zamianie, tylko w innej kolejności.

Należy rozważyć jeszcze wagi łuków od $\hat{D}_i^a$ do $\hat{D}_{j-1}^a$. Częścią wag są wartości $\hat{d}_a$, które nie ulegną zmianie, ponieważ są niezależnie od rozwiązania $\boldsymbol{\pi}$ (lub $\boldsymbol{\pi}'$). Wartości p_i^a oraz p_j^a, które również są częścią wag $\hat{D}_i^a$ oraz

$\hat{D}^a_{j-1}$ nie zmienią się, ponieważ są zależne od wag wierzchołków (a, i) oraz (a, j), na które zamiana nie ma wpływu. Ostatnią częścią wag $\hat{D}^i_k$ dla k od i do $j-1$ są wartości $-p^a_k$ oraz $-p^a_{k+1}$. Te wartości są zależne od wierzchołka (a, k) oraz jego wagi p^a_k. Jednak wszystkie wagi p^a_k pozostaną w bloku wewnętrznym (tylko ze zmienioną kolejnością). Zatem zmiana operacji nie zmieni ich całkowitego wkładu w wagi $\hat{D}^a_k$ oraz końcowej sumy wag od $\hat{D}^a_i$ do $\hat{D}^a_{j-1}$.

W rezultacie graf $\mathcal{G}(\boldsymbol{\pi}')$ będzie zawierał ścieżkę o takiej samej długości jak ścieżka krytyczna p^c w $\mathcal{G}(\boldsymbol{\pi})$, więc:

$$C_{\max}(\boldsymbol{\pi}') \geqslant C_{\max}(\pi). \tag{4.34}$$

■

Z powyższych dwóch Twierdzeń (4.3 oraz 4.4) wynika bezpośrednio, że zamiana operacji wewnątrz wewnętrznego bloku nie spowoduje natychmiastowej poprawy $C_{\max}$. Podobnie wygląda sytuacja przy zamianie dwóch operacji poza blokiem (operacji nieleżących na ścieżce krytycznej). Dlatego najbardziej obiecującą metodą przeglądania otoczenia jest wykonywanie ruchów typy zamień, w których jedna operacja należy do bloku wewnętrznego, a druga jest spoza całego bloku.

Twierdzenie 4.5 *Niech* $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_m)$ *będzie rozwiązaniem problemu* $F|\texttt{minimal idle, maximal idle}|C_{max}$ *oraz* p^c *będzie ścieżką krytyczną w grafie* $\mathcal{G}(\boldsymbol{\pi})$ *to istnieje dokładnie* m *bloków. Ponadto, bloki można wyznaczyć w czasie* $O(mn)$. □

Dowód. Dowód ma charakter konstrukcyjny oraz bazuje na Algorytmie 11. Przed rozpoczęciem obliczeń należy wyznaczyć dokładne czasy zakończenia wykonywania wszystkich operacji, najlepiej przy użyciu metody pokazanej wcześniej w Algorytmie 10. Wierzchołkiem początkowym, od którego należy zacząć, jest wierzchołek (m, n). W pierwszej kolejności należy sprawdzić czy ścieżka krytyczna nie idzie „w górę"(linia 3), ponieważ zabieg ten zagwarantuje znalezienie ścieżki bez cyklu (na podstawie wcześniejszych twierdzeń wiadomo, że ścieżka krytyczna nie zawiera cykli). Jeśli warunek jest spełniony, to wykryto blok na maszynie a. Należy przechować atrybuty jednoznacznie identyfikujące blok w odpowiedniej strukturze (linia 5). Jeśli kolejne wykrywane bloki będą dodawane zawsze na początku struktury przechowującej dane, to wystarczy zapamiętać wyłącznie początek i koniec bloku. Po wykryciu bloku oraz przed rozpoczęciem wykrywania bloku na maszynie $a-1$ należy najpierw wyzerować kierunek sprawdzania (linia 6) oraz numer zadania j (linia 7), które jest początkiem bloku.

Algorytm 11: Wyznaczanie bloków

Dane: $\boldsymbol{C}$: macierz momentów zakończenia operacji
Wynik: Blocks: zbiór bloków

```
1  function DetectedBlocks(C):
2      a ← m, i ← n, j ← n, direction ← 0;
3      while a > 1 ∨ i > 1 do
4          if C_i^a = C_{π_{a-1}^{-1}(π_a(i))}^{a-1} + p_{π(i)}^a then
5              Blocks.PushFront(block(i, j));
6              direction ← 0;
7              j ← i;
8              a ← a − 1;
9          else
10             if C_i^a = C_{i+1}^a − p_{π(i+1)}^a − d̂_a ∧ direction ≠ −1 then
11                 direction ← 1;
12                 i ← i + 1;
13             if C_i^a = C_{i−1}^a + p_{π(i)}^a + r̂_a ∧ direction ≠ 1 then
14                 direction ← −1;
15                 i ← i − 1;
16     return Blocks;
```

Jeśli ścieżka krytyczna nie idzie „w górę" to należy określić jej kierunek: (1) w lewo (linie 10-12) lub (2) w prawo (linie 13-15). W trakcie sprawdzania ścieżki nie ma możliwości zawrócenia. Na podstawie przedstawionego pseudokodu dobrze widać, że na każdej maszynie wykrywanie bloku odbędzie się dokładnie raz. Zostanie znalezionych dokładnie m bloków (w tym również jednoelementowe) po jednym dla każdej maszyny. Każda maszyna zostanie sprawdzona w czasie $O(n)$. Zatem wszystkie bloki można wykryć w czasie $O(mn)$.

4.2.3 Algorytmy optymalizacyjne

W tym rozdziale zostaną przedstawione dwie metody rozwiązania problemu $F|\texttt{minimal idle, maximal idle}|C_{\max}$: (1) Metoda Podziału i Ograniczeń, oraz (2) Przeszukiwanie z Zabronieniami. Metoda dokładna B&B posłuży jako jako odniesienie do oceny jakości metody przybliżonej TS.

Metoda Podziału i Ograniczeń

Metoda Podziału i Ograniczeń (B&B) [49] jest metodą dokładną. Metoda wykorzystuje dolne i górne ograniczenie do pominięcia przeglądu części przestrzeni rozwiązań, w których na pewno nie zawiera się optymalne rozwiązanie.

Przedstawiona poniżej implementacja B&B opiera się na iteracyjnym przeglądzie węzłów umieszczonych w kolejce priorytetowej. Każdy element kolejki ν zawiera: (1) rozwiązanie częściowe (ν_π), (2) nr maszyny na której aktualnie algorytm wykonuje rozgałęzienie (*branch*, ν_L), (3) zbiór ν_N zadań które nie zostały jeszcze uszeregowane na aktualnej maszynie (4) oraz wartość dolnego ograniczenia $LB(\nu)$.

Metoda rozgałęziania jest podobna do używanej w metodzie B&B dla JSSP [24]. Najpierw rozważane są wszystkie zadania na bieżącym poziomie (permutacji, na bieżącej maszynie) zanim przejdzie się na kolejny poziom. Zabieg ten stosowany jest w celu uniknięcia ponownego przetwarzania tych samych węzłów. Jeśli $\nu_\mathrm{L} = m$ oraz $\nu_\mathrm{N} = \varnothing$ to węzeł ν jest liściem (ν_π zawiera pełne rozwiązanie), więc należy policzyć dla niego dokładną wartość C_max. Jeśli wynikowa wartość C_max dla węzła jest lepsza od wartości najlepszego znalezionego rozwiązania do tej pory, to należy zaktualizować bieżące górne ograniczenie UB. Początkowe UB jest generowane przy użyciu zachłannej heurystyki podobnej do NEH [94] dla $FP||C_\mathrm{max}$ lub INSA [97] dla $J||C_\mathrm{max}$, które bazują na technice wstawiania zadań w rozwiązywaniu częściowym oraz wyborze lokalnie najlepszej pozycji dającej najmniejszą długość uszeregowania. Dolne ograniczenie LB dla node jest obliczany w następujący sposób:

$$LB(\nu) = \max_{a \in \mathcal{M}} \left\{ PRV + \sum_{i \in SET} (\hat{r}_a + p_i^a) + \sum_{b=a+1}^{m} \min_{i \in \mathcal{J}} p_{b,i} \right\}, \tag{4.35}$$

gdzie:

$$PRV = \begin{cases} C_{a,\Theta_i(\nu_\pi)} & \text{jeśli } \Theta_a(\nu_\pi) > 0, \\ C_{a-1,1} & \text{jeśli } \Theta_a(\nu_\pi) = 0 \wedge a > 1, \\ 0 & \text{jeśli } \Theta_a(\nu_\pi) = 0 \wedge a = 1, \end{cases} \quad SET = \begin{cases} \varnothing & \text{jeśli } a < \nu_\mathrm{L}, \\ \nu_\mathrm{N} & \text{jeśli } a = \nu_\mathrm{L}, \\ \mathcal{J} & \text{w p.p.}, \end{cases} \tag{4.36}$$

gdzie funkcja $\Theta_a(\nu_\pi)$ zwraca ostatnie uszeregowanie zadanie na maszynie a w częściowym rozwiązaniu. Algorytm kończy działanie w momencie, gdy kolejka Q będzie pusta lub dolne ograniczenie najlepszego węzła w kolejce Q będzie gorsze niż aktualne UB.

Przeszukiwanie z Zabronieniami

W każdej iteracji algorytmu Przeszukiwania z Zabronieniami [56] jest przeglądane *sąsiedztwo* aktualnego rozwiązania. Sąsiedztwo $\mathcal{N}(\pi)$ rozwiązania π jest zdefiniowane jako wszystkie rozwiązania π', które można otrzymać z π po przez predefiniowaną funkcją *ruchu*. Dla badanego problemu zaproponowano trzy typy ruchów:

1. Ruch zamień $\mathtt{S}(\pi, a, i, j)$. Sąsiednie rozwiązanie π' otrzymuje się przez zamianę wartości $\pi_a(i)$ oraz $\pi_a(j)$, gdzie $a \in \mathcal{M}$, $i \in \mathcal{J}$, $j \in \mathcal{J} \setminus \{i\}$. Sąsiedztwo typu zamień ma dokładnie $\frac{n(n-1)}{2}m$ rozwiązań.
2. Ruch zamień sąsiedni $\mathtt{A}(\pi, a, i)$. Sąsiednie rozwiązanie π' uzyskuje się poprzez jednoczesną zamianę sąsiednich elementów $\pi_a(i)$ oraz $\pi_a(i+1)$, gdzie $a \in \mathcal{M}$, $i \in \mathcal{J} \setminus \{n\}$. Sąsiedztwo typu zamień sąsiedni ma dokładnie $(n-1)m$ rozwiązań.
3. Ruch zamień blokowy $\mathtt{B}(a, i, j)$. Ruch jest podobny do $\mathtt{S}(\pi, a, i, j)$, ale z uwzględnianiem dowodów Twierdzeń 4.3 oraz 4.4. W szczególności, jeśli $\mathcal{B}_a$ jest wewnętrznym blokiem na maszynie a, to ruch jest wykonywany tylko wtedy, gdy $i \in \mathcal{B}_a \wedge j \notin \mathcal{B}_a$ lub $i \notin \mathcal{B}_a \wedge j \in \mathcal{B}_a$. Rozmiar sąsiedztwa typu zamień blokowy zmienia się dynamicznie, zależy od rozmiarów bloków w rozwiązaniu π, ale jego rozmiar mieści się miedzy rozmiarami sąsiedztw generowanych ruchami `A` oraz `S` ze średnio $\frac{n^2m}{8}$ rozwiązaniami.

Wszystkie powyższe typy ruchów są inwolucjami.

W każdej iteracji it algorytmu TS jest generowane sąsiedztwo aktualnego rozwiązania π^{it} (zgodnie z wybranym typem ruchu). Następnie wszystkie otrzymane w ten sposób rozwiązania sąsiednie są oceniane (obliczany jest $C_{\max}$). Najlepsze z tych rozwiązań stanie się bieżącym rozwiązaniem w następnej iteracji π^{it+1}. Ponadto, jeśli najlepszy sąsiad $\pi^{\mathcal{N}}_{\text{best}}$ jest lepszy niż najlepsze znalezione do tej pory rozwiązanie globalne π^*, należy zamienić π^* na $\pi^{\mathcal{N}}_{\text{best}}$. Początkowe rozwiązanie π^1 jest albo losowe, albo generowane przy użyciu prostej metody zachłannej.

Jedną z najbardziej znanych wad takiej strategii przeszukiwania sąsiedztwa jest to, że algorytm szybko osiąga lokalne optimum, a cykl rozwiązań jest powtarzany w nieskończoność. Aby temu zaradzić, TS ma pamięć krótkotrwałą nazywaną zazwyczaj listą tabu lub zakazów. Lista ma za zadanie określić, które ruchy są zabronione. Rozwiązania generowalne poprzez zabronione ruchy są generalnie ignorowane przy wyborze najlepszego sąsiada $\pi^{\mathcal{N}}$. Jedynym wyjątkiem od tej reguły jest sytuacja, gdy zabroniony sąsiad poprawia globalnie najlepsze rozwiązanie π^*. W takiej sytuacji może zo-

stać wybrane takie rozwiązanie sąsiednie jako najlepsze z otoczenia, mimo że jest zabronione.

Kiedy wykonywany jest ruch (generujący sąsiednie rozwiązanie $\pi_{\text{best}}^{\mathcal{N}}$), należy przechować jego atrybuty a, i oraz j na liście tabu dla kolejnych `C` iteracji, gdzie `C` jest parametrem zwanym kadencja (*cadence*). W celu przyśpieszenia przeglądania listy tabu, implementuje się listę zakazów jako trójwymiarową macierz. Zabieg ten pozwala na wykonywania wszystkich operacji na liście tabu w czasie stałym $O(1)$. Jednak mimo utworzenia listu tabu, algorytm nadal może utknąć w cyklu. Zatem algorytm TS jest restartowany (następuje zmiana bieżącego rozwiązania na losowe) po tym jak przez `K` kolejnych iteracji nie zostało poprawione globalnie najlepsze rozwiązanie π^*, gdzie `K` jest parametrem algorytmu.

Ze względu na różny rozmiar sąsiedztwa w implementacji TS dla rozważanego problemu, wartość parametru `C` została ustawiona osobno dla każdego typu sąsiedztwa: nm dla ruchu zamień, $\lfloor\sqrt{nm}\rfloor$ dla ruchu zamień sąsiedni i $\lfloor nm/2\rfloor$ dla ruchu zamień blokowy. Po wstępnych badaniach wartość `K` została ustawiona na 20.

4.2.4 Eksperymenty obliczeniowe

Przygotowano dwa zestawy instancji testowych A i B, wygenerowane przy użyciu generatora instancji zaproponowanego przez Taillarda [129]. Zbiór A zawiera instancje o niewielkich rozmiarach, które zostaną użyte do przetestowania skuteczności metaheurystyki TS w porównaniu do metody dokładnej B&B. Zestaw B składa się z instancji o większych rozmiarach, został wykorzystany do testowania kilku wersji TS w bardziej praktycznych warunkach. Ponadto każdy zestaw został podzielony na 5 podgrup z różnym zakresem czasów wykonywania oraz minimalnym i maksymalnym czasem przestoju: (1) 1–99, (2) 10–90, (3) 30–70, (4) 40–60, (5) 45–55. Dla każdej z podgrup oraz każdego rozmiaru wygenerowano po 5 instancji.

Jako miarę jakości zastosowano procentowe względne odchylenie (*Percentage Relative Deviation*, PRD). Niech π oznacza rozwiązanie uzyskane przez badany algorytm dla konkretnej instancji oraz π^* rozwiązanie referencyjne, które zazwyczaj jest najlepszym znanym rozwiązaniem. Wtedy PRD można zdefiniować następująco.

$$\text{PRD} = \frac{C_{\max}(\pi) - C_{\max}(\pi^*)}{C_{\max}(\pi^*)} \cdot 100\%. \tag{4.37}$$

Wszystkie eksperymenty zostały przeprowadzone na komputerze z procesorem Intel Core i7-8650U 1,9 GHz oraz 16 GB pamięci RAM. Program

Tablica 4.7: Średnie czasy działania T (metod B&B oraz TS, w sekundach) oraz średnie PRD (metody TS) dla instancji ze zbioru A

$n \times m$	$T^1_{\text{B\&B}}$	$T^2_{\text{B\&B}}$	$T^3_{\text{B\&B}}$	$T^4_{\text{B\&B}}$	$T^5_{\text{B\&B}}$	T_{TS}	PRD_{TS}
4×4	0,54	0,16	0,07	0,05	0,05	0,004	0,54
4×5	1,71	0,77	0,10	0,07	0,07	0,010	0,58
5×4	46,33	18,13	2,15	0,50	0,51	0,015	1,28
5×5	213,56	65,61	1,37	0,85	0,75	0,034	2,30

został napisany języku w `C++`, skompilowany przy użyciu `g++` 7.5.0 i uruchomiony pod systemem Linux Mint (jądro 5.4.0).

W pierwszym eksperymencie zostały porównane metody TS oraz B&B dla instancji z zestawu A. Wyniki zostały przedstawione w Tabeli 4.7. Dla wszystkich instancji danego rozmiaru czas działania algorytmu TS oznaczono jako T_{TS}. Podczas gdy przez $T^g_{\text{B\&B}}$ został oznaczany czas działania metody B&B dla konkretnej podgrupy g, gdzie $T^1_{\text{B\&B}}$ to czas działania B&B dla podgrupy nr 1 o największym zakresie czasów wykonywania (1-99). Na podstawie wyników można zaobserwować, że podczas gdy czas działania B&B rośnie wraz z rozmiarem problemu, to również maleje z numerem podgrupy (zmniejszeniem zakresu danych). Dla grupy o rozmiarze 5×5 metoda B&B w podgrupie 5 jest ponad 280 razy szybszy niż grupie 1. Oznacza to, że instancje z mniejszym zakresem czasów wykonywania i przestojów maszyny są znacznie łatwiejsze do rozwiązania. Należy zauważyć również, że czasy działania metaheurystyki TS są znacznie krótsze niż czasy B&B oraz mieszczą się w czasie poniżej 35 ms. W badanym przypadku jako rozwiązanie referencyjne π^* było rozwiązanie optymalne dostarczone przez metodę dokładną B&B. Zatem, algorytm TS dostarczył średnio rozwiązania odległego mniej niż $2,3\%$ od wartości optymalnej $C_{\max}$ nawet dla instancji o rozmiarze 5×5 w krótkim czasie.

W drugim eksperymencie zostały porównane trzy wersje algorytmu TS dla zbioru instancji B. Każda wersja algorytmu przeglądała inne sąsiedztwo (`S` dla ruchu zamień, `A` dla ruchu zamień sąsiedni oraz `B` dla ruchu zamień blokowy). Warunkiem zatrzymania algorytmu we wszystkich trzech wersjach był limit czasowy. Ponadto rozwiązanie π^*_{best} zostało wybrane jako najlepsze z badanych trzech wersji TS dla danej instancji. Wyniki przedstawiono w Tabeli 4.8.

Warto zwrócić uwagę na to, że dla 6 grup najlepsze rozwiązanie zawsze zostało znalezione przy użyciu metody TS wykorzystującej sąsiedztwo typu zamień blokowy. Ponadto, PRD dla tego otoczenia nie przekraczało $0,8\%$,

Tablica 4.8: Średnie PRD oraz czas działania T (w sekundach) dla metody TS z sąsiedztwem zamień, zamień sąsiedni oraz zamień blokowy dla instancji ze zbioru B

n	$m = 10$				$m = 15$			
	PRD_{B}	PRD_{A}	PRD_{S}	T	PRD_{B}	PRD_{A}	PRD_{S}	T
10	0,58	12,95	6,67	0,01	0,07	15,04	8,95	0,02
20	0,00	17,52	14,27	0,09	0,00	14,77	13,14	0,26
30	0,00	8,25	11,13	0,38	0,00	9,02	12,78	1,12
40	0,03	4,43	11,98	1,18	0,00	4,77	9,71	3,03
50	0,08	3,36	10,74	2,16	0,10	2,71	9,43	6,23
60	0,66	1,42	10,31	3,87	0,08	1,74	8,60	11,38
80	0,00	4,61	11,16	10,14	0,01	1,65	9,12	30,28
90	0,08	2,65	9,60	17,35	0,02	2,61	9,87	46,23
100	0,76	1,06	9,60	22,93	0,40	0,55	8,19	66,77
średnia	0,24	6,24	10,65	-	0,08	5,87	9,98	-

przy średniej wartości $0,24\%$ oraz $0,08\%$ odpowiednio dla $m = 10$ oraz $m = 15$ maszyn. Średnie wartości PRD dla pozostałych dwóch sąsiedztw były dziesiątki razy większe. Przedstawione wyniki wskazują, że własności eliminacyjne blokowe z Twierdzenia 4.3 oraz Twierdzenia 4.4 pozwoliły metodzie TS na bardzo szybką zbieżność do minimum, znajdując dobre rozwiązania w około 60 sekund nawet dla instancji o rozmiarze 100×15.

4.2.5 Akceleracja funkcji celu

W Algorytmie 10 będącym podstawą dowodu (konstrukcyjnego) Twierdzenia 4.2 pokazano sekwencyjną metodą wyznaczania wartości funkcji celu w czasie $O(nm)$. Poniżej zostanie zaproponowany równoległy algorytm wyznaczania wartości funkcji celu dla rozważanego problemu ze sprzężeniami czasowymi, który może być zastosowany do obliczeń bazując na modelu maszyny równoległej o dostępie swobodnym z jednoczesnym odczytem oraz wyłącznym zapisem (*Concurrent Read, Exclusive Write Parallel Random Access Machine*, CREW PRAM [42]). Model PRAM jest rozszerzeniem sekwencyjnego modelu maszyny o dostępie swobodnym (*Random Access Machine* RAM). Cormen i inni [29] zaznaczają, że większość algorytmów równoległych dla podstawowych struktur danych (list czy grafów) można łatwo przedstawić w modelu PRAM. Ponadto własności oraz sposób działania algorytmów dla modelu PRAM są zbliżone do rzeczywistych (np. GPU)

mimo pominięcia części praktycznych aspektów maszyn równoległych. Jeśli jeden z zamodelowanych algorytmów będzie lepszy niż drugi w sensie złożoności obliczeniowej, to istnieje bardzo mała szansa, że po zaimplementowaniu obu algorytmów na rzeczywistym komputerze ich zachowanie ulegnie istotnej zmianie.

Jedną z elementarnych metod przedstawianych dla modelu PRAM jest obliczenie minimum z ciągu liczb, którą można sformułować jako następujący fakt fakt:

Fakt 1 *Minimalna wartość z n-elementowego ciągu liczbowego może być wyznaczona na maszynie CREW PRAM w czasie* $O(\log n)$ *przy użyciu* $O(n/\log n)$ *procesorów.* □

Dowód. W fazie pierwszej należy podzielić ciąg na $O(n/\log n)$ bloków (podciągów) o długości $O(\log n)$. Każdy procesor oblicza sekwencyjnie minimum z każdego bloku w czasie $O(\log n)$. Zatem, otrzymujemy $O(n/\log n)$ wartości. W drugim etapie należy znaleźć minimum z otrzymanych wartości. Można to zrobić w czasie $O(\log n)$, ponieważ mamy do dyspozycji $O(n/\log n)$ procesorów. Zatem obie fazy łącznie zajmują czas $O(\log n)$ przy użyciu $O(n/\log n)$ procesorów. ■

Na podstawie Faktu 1 oraz grafu $\mathcal{G}(\boldsymbol{\pi})$ można sformułować następujące twierdzenie o złożoności czasowej równoległego algorytmy wyznaczania wartości funkcji celu dla problemu $F|\texttt{minimal idle, maximal idle}|C_{\max}$.

Twierdzenie 4.6 *Dla ustalonej kolejności wykonywania zadań na wszystkich maszynach* $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_m)$ *wartość funkcji* $C_{\max}(\boldsymbol{\pi})$ *dla problemu* $F|\texttt{minimal idle, maximal idle}|C_{max}$ *można wyznaczyć w czasie* $O(\log^2(nm))$ *przy zastosowaniu modelu obliczeniowego CREW PRAM oraz użyciu* $O\left(\frac{(nm)^3}{\log(nm)}\right)$ *procesorów.* □

Dowód. W celu obliczenia wartości $C_{\max}(\boldsymbol{\pi})$ należy obliczyć długość najdłuższej ścieżki w grafie $\mathcal{G}(\boldsymbol{\pi})$. Proponowana równoległa metoda obliczania najdłuższej ścieżki jest oparta na sekwencyjnym algorytmie Floyda-Warshalla do znajdowania najkrótszych ścieżek w grafach [30]. Algorytm dopuszcza ujemne wagi krawędzi oraz cykle, dopóki nie ma cyklu ujemnego (cyklu z ujemną sumą wag krawędzi).

Znalezienie najdłuższej ścieżki w grafie $\mathcal{G}(\boldsymbol{\pi}) = (V, E)$ jest równoważne znalezieniu najkrótszej ścieżki w grafie $\mathcal{G}'(\boldsymbol{\pi}) = (V, E')$. Struktura grafu $\mathcal{G}'(\boldsymbol{\pi})$ (układ wierzchołków oraz krawędzi) jest taka sama jak $\mathcal{G}(\boldsymbol{\pi})$, z wyjątkiem wag krawędzi, które są zanegowane:

$$\forall (u,v) \in E: \quad (u,v) \in E' \wedge \psi'(u,v) = -\psi(u,v), \tag{4.38}$$

gdzie $\psi(u,v)$ oraz $\psi'(u,v)$ są wagami krawędzi (u,v) odpowiednio w grafie $\mathcal{G}(\boldsymbol{\pi})$ oraz $\mathcal{G}'(\boldsymbol{\pi})$, a wartość $\psi'(u,v)$ jest liczbą przeciwną do $\psi(u,v)$. Na podstawie Własności 4.2 wiadomo, że graf $\mathcal{G}(\boldsymbol{\pi})$ dla rozważanego problemu $F|\texttt{minimal idle, maximal idle}|C_{\max}$ nie ma dodatnich cykli, więc $\mathcal{G}'(\boldsymbol{\pi})$ nie będzie miał cykli ujemnych. Zatem w badanym przypadku można użyć algorytmu Floyda-Warshalla.

W celu poprawienia czytelności notacji, graf $\mathcal{G}'(\boldsymbol{\pi})$ zostanie przekształcony w graf $\mathcal{G}^*(\boldsymbol{\pi})$ poprzez zmianę numeracji wierzchołków. W rezultacie wierzchołki $\mathcal{G}^*(\boldsymbol{\pi})$ będą indeksowane przy użyciu pojedynczej wartości u zamiast pary (a,i) jak dla $\mathcal{G}^*(\boldsymbol{\pi})$. Tłumaczenie numeracji z (a,i) na u wygląda następująco:

$$u = (a-1)n + i. \tag{4.39}$$

W ten sposób wierzchołki są numerowane począwszy od lewego górnego rogu, numerując wszystkie wierzchołki w rzędzie przed przejściem do następnego rzędu. Transformacja odwrotna jest następująca:

$$a = \left\lfloor \frac{u-1}{n} \right\rfloor + 1, \quad i = u - \left\lfloor \frac{u-1}{n} \right\rfloor n. \tag{4.40}$$

Wynikiem będzie graf $\mathcal{G}^*(\boldsymbol{\pi}) = (V^*, E^*)$, gdzie $V^* = \{1, 2, \ldots, nm\}$ jest zbiorem nm wierzchołków. Waga wierzchołka $u \in V^*$ została oznaczona jako p_u i jest równa wadze $p_u = -p_i^a$ odpowiedniego wierzchołka z $\mathcal{G}'(\boldsymbol{\pi})$. Zbiór krawędzi E^* można podzielić na zbiory E_1^*, E_2^* and E_3^* reprezentujące pionowe (technologiczne), poziome w przód oraz poziome powrotne krawędzie z grafu $\mathcal{G}^*(\boldsymbol{\pi})$:

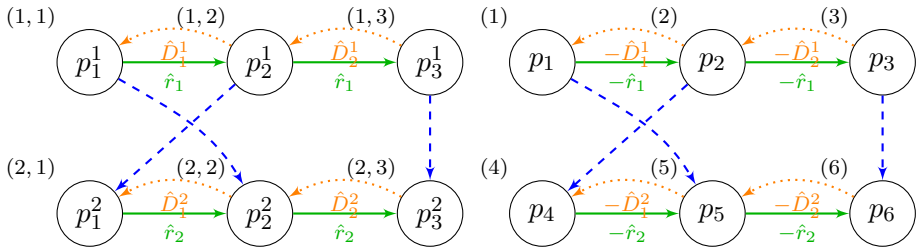
$$E_1^* = \bigcup_{u=1}^{nm-n} \{(u, u+n)\}, \tag{4.41}$$

$$E_2^* = \bigcup_{a=1}^{m} \bigcup_{u=(a-1)n}^{an-1} \{(u, u+1)\}, \tag{4.42}$$

$$E_3^* = \bigcup_{a=1}^{m} \bigcup_{u=(a-1)n}^{an-1} \{(u+1, u)\}. \tag{4.43}$$

Wagi $\psi(u,v)$ krawędzi (u,v) w grafie $\mathcal{G}^*(\boldsymbol{\pi})$ zostały przypisane następująco:

$$\psi(u,v) = \begin{cases} 0 & \text{jeśli} \quad (u,v) \in E_1^*, \\ -\hat{r}_a & \text{jeśli} \quad (u,v) \in E_2^*, \\ -\hat{D}_i^a & \text{jeśli} \quad (u,v) \in E_3^*. \end{cases} \tag{4.44}$$



(a) Graf $\mathcal{G}(\boldsymbol{\pi})$ (b) Graf $\mathcal{G}^*(\boldsymbol{\pi})$

$$\begin{bmatrix} 0 & -\hat{r}_1 - p_2 & \infty & \infty & -p_5 & \infty \\ -\hat{D}_1^1 - p_1 & 0 & -\hat{r}_1 - p_3 & -p_4 & \infty & \infty \\ \infty & -\hat{D}_2^1 - \rho_2 & 0 & \infty & \infty & -\rho_6 \\ \infty & \infty & \infty & 0 & -\hat{r}_2 - p_5 & \infty \\ \infty & \infty & \infty & -\hat{D}_1^2 - p_4 & 0 & -\hat{r}_2 - p_6 \\ \infty & \infty & \infty & \infty & -\hat{D}_2^2 - p_5 & 0 \end{bmatrix}$$

(c) Macierz A

Rysunek 4.6: Przykład transformacji grafu rozwiązania $\mathcal{G}(\boldsymbol{\pi})$ na graf $\mathcal{G}^*(\boldsymbol{\pi})$ oraz początkowe wartości macierzy A

gdzie a oraz i można przetransformować z u za pomocą Wzoru (4.40). Przykład prostego grafu $\mathcal{G}(\boldsymbol{\pi})$ oraz odpowiadającego mu grafu $\mathcal{G}^*(\boldsymbol{\pi})$ pokazano kolejno na Rysunkach 4.6a oraz 4.6b.

Dla grafu $\mathcal{G}^*(\boldsymbol{\pi})$ można zdefiniować macierz $A = [a_{u,v}]$ o rozmiarze $nm \times nm$, gdzie $a_{u,v}$ będzie reprezentować długość najdłuższej ścieżki między wierzchołkami u oraz v w grafie $\mathcal{G}^*(\boldsymbol{\pi})$. Wartości macierzy A należy początkowo wypełnić w następujący sposób:

$$a_{u,v} = \begin{cases} 0 & \text{if} \quad u = v, \\ \psi(u,v) - p_u & \text{if} \quad u \neq v \wedge (u,v) \in E^*, \\ \infty & \text{if} \quad u \neq v \wedge (u,v) \notin E^*. \end{cases} \tag{4.45}$$

Powodem uwzględnienia wartości p_u jest to, że algorytm Floyda-Warshalla rozpoznaje jedynie ważone krawędzie, a nie ważone wierzchołki. Zatem waga p_u jest dodawana do każdej krawędzi, która zaczyna się w wierzchołku u. Początkowa zawartość macierzy A dla grafu z Rysunku 4.6b jest pokazana na Rysunku 4.6c.

Macierz A będzie użyta do obliczenia najkrótszej ścieżki w grafie $\mathcal{G}^*(\boldsymbol{\pi})$, której negacja będzie najdłuższa ścieżka w oryginalnym grafie $\mathcal{G}(\boldsymbol{\pi})$. Każda

z $(nm)^2$ wartości początkowych macierzy A zostanie obliczona niezależnie od pozostałych. Zatem początkowe wypełnienie macierzy A można wykonać w czasie $O(1)$ na maszynie CREW PRAM przy użyciu $O((nm)^2)$ procesorów, z których każdy wykonuje pojedyncze przypisanie.

Ponadto należy zdefiniować trójwymiarową tablicę $H = [h_{u,w,v}]$ o rozmiarze $nm \times nm \times nm$, która zostanie użyta do obliczenia przechodniego domknięcia grafu $\mathcal{G}^*(\boldsymbol{\pi})$. Innymi słowy, wartość $h_{u,w,v}$ będzie używana do przechowywania oraz aktualizowania długości najkrótszej ścieżki od wierzchołka u do wierzchołka v, który przechodzi przez wierzchołek w.

Podstawowa część algorytmu opiera się na powtarzaniu następujących dwóch kroków w pętli:

1. Zaktualizuj $h_{u,w,v}$ dla wszystkich trójek (u, w, v) zgodnie z następującym wzorem:
$$h_{u,w,v} = a_{u,w} + a_{w,v}, \tag{4.46}$$
2. Zaktualizuj $a_{u,v}$ dla wszystkich par (u, v) na podstawie następującego wzoru:
$$a_{u,v} = \min\{a_{u,v}, \min_{1 \leqslant w \leqslant nm} h_{u,w,v}\}. \tag{4.47}$$

Celem jest określenie wartości $a_{1,nm}$ (długość ścieżki od wierzchołka 1 do wierzchołka nm), aby go osiągnąć wystarczy wykonać powyższe dwa kroki $\lceil \log(nm-1) \rceil$ razy, ponieważ na każdym kroku algorytm znajduje najkrótszą ścieżkę między wierzchołkami położonymi dalej od siebie. Po pierwszej iteracji algorytmu obliczone zostaną najkrótsze ścieżki składające się z jednej krawędzi. Po drugiej iteracji algorytmu zostaną obliczone najkrótsze ścieżki złożone z maksymalnie dwóch krawędzi. Po trzeciej iteracji zostaną obliczone najkrótsze ścieżki złożone z maksymalnie 4 krawędzi i tak dalej. Zatem po k-tej iteracji algorytm obliczy najkrótsze ścieżki złożone z maksymalnie 2^{k-1} krawędzi.

Najdłuższa możliwa ścieżka (pod względem liczby krawędzi wchodzących w skład grafu) na grafie $\mathcal{G}^*(\boldsymbol{\pi})$ będzie przebiegać od lewej do prawej przez całą pierwszą maszynę, a następnie cofać się od prawej do lewej przez całą drugą maszynę, potem od lewej do prawej na trzeciej maszynie i tak dalej. Ścieżka ta będzie więc przebiegać się po „zygzakowatym" wzorze, przechodząc przez wszystkie wierzchołki grafu. Łatwo zauważyć, że taka ścieżka będzie miała nie więcej niż $nm - 1$ krawędzi. W ten sposób algorytm obliczy długość tej ścieżki po wykonaniu $\lceil \log(nm-1) \rceil$ iteracji. Oczywiście, ścieżka najdłuższa względem liczby krawędzi nie musi być najdłuższą względem sumy wag, jest tylko ilustracją spełnialności stwierdzenia o wystarczającej liczbie $\lceil \log(nm-1) \rceil$ iteracji.

Poniżej zostanie omówiony równoległy sposób wykonania kroków opisanych wzorami (4.46) oraz (4.47). Pierwszy krok może zostać wykonany w czasie $O(1)$ na modelu maszyny CREW PRAM, gdy zostanie przypisane do niego $O((nm)^3)$ procesorów, z których każdy wykona pojedyncze przypisanie dla jednej trójki (u, w, v). Jednak w przypadku $\lceil (nm)^3 / \log(nm) \rceil$ procesorów, każdy procesor musi przetworzyć nie jedną trójkę (u, w, v), ale dokładnie $\lceil \log(nm) \rceil$ takich trójek. Złożoność czasowa tego kroku będzie zatem wynosić $O(\log(nm))$.

Zadaniem kroku 2 jest obliczenie minimum z $nm + 1$ wartości, które można wykonać (zgodnie z Faktem 1) na modelu CREW PRAM w czasie $O(\log(nm))$ przy użyciu $O(nm/\log(nm))$ procesorów. Takie minimum należy wyznaczyć dla $(nm)^2$ różnych par (u, v), a obliczenia dla każdej pary można wykonać niezależne od pozostałych. Zatem krok ten można wykonać w czasie $O(log(nm))$ używając łącznie $(nm)^2 O(nm/\log(nm)) = O((nm)^3/\log(nm))$ procesorów.

Jak już wspomniano wcześniej, kroki 1 i 2 aktualizujące wartości odpowiednio macierzy H oraz A muszą zostać powtórzone $\lceil \log(nm - 1) \rceil$ razy w pętli, stąd całkowita złożoność czasowa tej pętli wynosi:

$$\lceil \log(nm - 1) \rceil O(\log(nm)) = O(\log^2(nm)), \tag{4.48}$$

przy użyciu $(nm)^2 O(nm/\log(nm)) = O((nm)^3/\log(nm))$ procesorów.

Należy zwrócić również uwagę, że waga p_u wierzchołka u jest używana tylko we wzorze (4.45), jeśli u ma krawędzie wychodzące. Dotyczy to wszystkich wierzchołków z wyjątkiem wierzchołka nm, który nie ma żadnych krawędzi wychodzących do rozpatrzenia, ponieważ generowałby cykl. Z dowodu Wniosku 4.2 wynika, że ścieżka krytyczna nie może zawierać cykli. Zatem wartość p_{nm} nie jest brana pod uwagę przez algorytm, a wartość $a_{1,nm}$ nie jest ostateczną wartością $C_{\max}$. Wiadomo jednak, że wierzchołek nm zawsze musi być zawarty w ścieżce krytycznej, ale jest odwiedzany tylko raz i to jako ostatni. Zatem ostateczną wartość $C_{\max}(\boldsymbol{\pi})$ można wyznaczyć w czasie $O(1)$ następująco:

$$C_{\max}(\pi) = -a_{1,nm} + p_{nm}. \tag{4.49}$$

Zatem końcowa złożoność czasowa całego algorytmu (w tym inicjalizacja macierzy A, pętla oraz końcowa korekta) wynosi $O(\log^2(nm))$ przy wykorzystaniu $O(\frac{(nm)^3}{\log(nm)})$ procesorów. ■

Tablica 4.9: Teoretyczne przyspieszenia proponowanej metody równoległej w porównaniu z podejściem sekwencyjnym

$n \times \ m$	T_s	T_p	$\mathtt{S}_p$	$T_s^{\mathcal{N}}$	$T_p^{\mathcal{N}}$	$\mathtt{S}_p^{\mathcal{N}}$
$10 \times \ 5$	50	36	1,39	2250	40	56,25
20×10	200	64	3,13	38000	69	550,72
30×15	450	81	5,56	195750	86	2276,16
40×20	800	100	8,00	624000	106	5886,79
50×20	1000	100	10,00	980000	106	9245,28

Dyskusja

W tym podrozdziale zostaną omówione możliwe wyniki zastosowania proponowanej metody obliczeń równoległych w algorytmach optymalizacyjnych dla problemu $F|\texttt{minimal idle, maximal idle}|C_{\max}$.

Pierwszym interesującym punktem jest możliwe przyspieszenie $\mathtt{S}_p$ (*speedup*), które możemy osiągnąć wykorzystując równoległą metodę wyznaczania pojedynczej wartości $C_{\max}$. Przyspieszenie definiuje się jako:

$$\mathtt{S}_p = \frac{T_s}{T_p}, \tag{4.50}$$

gdzie przez T_s oznaczono czas potrzebny do wyznaczenia $C_{\max}$ tradycyjną metodą (sekwencyjnie) używając tylko jednego procesora, przez przez T_p czas potrzebny do wyznaczenia $C_{\max}$ wykorzystując zaproponowaną powyżej metodę równoległą, używając $\frac{(nm)^3}{\log(nm)}$ procesorów. Teoretyczne wartości przyspieszenia dla kilku rozmiarów problemów powszechnie rozważanych w literaturze zostały pokazane w Tabeli 4.9. Można łatwo zauważyć, że proponowana, równoległa metoda może zapewnić znaczne przyspieszenie (do 10 dla rozważanych rozmiarów problemu).

Wyznaczanie pojedynczej wartości $C_{\max}$ jest elementem większości algorytmów rozwiązywania problemu $F|\texttt{minimal idle, maximal idle}|C_{\max}$. Istnieje jednak grupa metod, zwana metodami lokalnymi przeszukiwania, polegająca na przeszukiwaniu całego otoczenia danego rozwiązania w celu znalezienia najlepszego rozwiązania. Jednym z takich algorytmów jest metoda TS opisana w rozdziale 4.2.3. W przypadku badanego problemu jednym z zaproponowanych sąsiedztw jest sąsiedztwo otrzymane poprzez ruch tupu zamień sąsiedni. Na każdej maszynie istnieje $n-1$ możliwych sąsiednich par zadań oraz jest m maszyn, więc w skład sąsiedztwa wchodzi dokładnie $(n-1)m$ rozwiązań. Zatem możliwe jest dalsze skrócenie obliczeń poprzez równoległe obliczanie każdego rozwiązania z sąsiedztwa. Można

zdefiniować przyspieszenie $\mathtt{S}_p^{\mathcal{N}}$ podobne do poprzedniego następująco:

$$\mathtt{S}_p^{\mathcal{N}} = \frac{T_s^{\mathcal{N}}}{T_p^{\mathcal{N}}}, \tag{4.51}$$

gdzie $T_s^{\mathcal{N}}$ to czas przeszukiwania wszystkich $(n-1)m$ rozwiązań jedno po drugim przy użyciu pojedynczego procesora, a $T_p^{\mathcal{N}}$ to czas przeszukiwania równolegle sąsiedztwa przy użyciu $\lceil \frac{(nm)^3}{\log(nm)} \rceil$ procesorów dla każdego rozwiązania w sąsiedztwie. Wartości $S^{\mathcal{N}}$ zostały również pokazane w Tabeli 4.9. Można zaobserwować, że teoretycznie możliwe do uzyskania przyspieszenie jest bardzo wysokie (do kilku tysięcy dla typowych rozmiarów problemów). Wynika to głównie z rozmiaru otoczenia, ale jest dodatkowo wzmocnione przez proponowaną równoległą metodę wyznaczania $C_{\max}$.

Zaproponowana metoda oferuje znaczne przyspieszenie, jednak wymaga bardzo dużej liczby procesorów. Już dla problemu o rozmiarze 10×5 metoda wymaga 25 000 procesorów równoległych do wyznaczenia wartości $C_{\max}$ i liczba potrzebnych procesorów rośnie wraz ze wzrostem rozmiaru problemu. Zatem, chociaż wynik ten dowodzi, że algorytm Floyda-Warshalla może być używany do wspomagania wyznaczania funkcji celu dla szerokiego zakresu problemów szeregowania zadań ze sprzężeniami czasowymi, to jednak proponowana metoda nie jest jeszcze wykonalna przy obecnym stanie technologii obliczeń równoległych takich jak Nvidia CUDA (oferująca aktualnie kilka tysięcy rdzeni CUDA). Jednakże, chociaż proponowana metoda pozostaje obecnie w większości teoretyczna, nadal można ją stosować w równoległych systemach (klastry obliczeniowe) wyposażonych w tysiące oddzielnych maszyn. Co więcej, ze względu na prawo Brenta [23] możliwe jest również zastosowanie proponowanej metody w systemie z mniejszą liczbą procesorów, ale w takim przypadku metoda będzie działać wolniej (przy czym przeskalowanie jest liniowe).

4.2.6 Wnioski i uwagi

W tym rozdziale został przedstawiony niepermutacyjny problem przepływowy ze sprzężeniami czasowymi dla kryterium czasu zakończenia wszystkich zadań. Opisany problem bazuje na rzeczywistym procesie betonowania w przemyśle. Sformułowano model matematyczny oraz kilka własności opartych na reprezentacji grafowej, w tym metodę wyznaczania wartości funkcji celu wraz z jej złożonością. Co ważniejsze, pokazano oraz udowodniono własności eliminacyjne wykorzystującą koncepcję bloków prawostronnych oraz lewostronnych w modelu grafowym.

Zaproponowno dwie metody rozwiązania: dokładną Metodę Podziału i Ograniczeń (B&B) oraz metaheurystykę Przeszukiwania z Zabronieniami (TS). Eksperymenty komputerowe na instancjach bazujących na generatorze Taillarda wykazały małe (poniżej $2,3\%$) wartości odchylenia od optymalnego rozwiązania dla metody TS. Ponadto zaobserwowano duży wpływ danych wejściowych (rozrzut czasów wykonywania, minimalnego oraz maksymalnego przestoju) na czas działania metody B&B, który malał wraz ze zmniejszam możliwych zakresów danych wejściowych. Co więcej, wyniki pokazały, że otoczenie wykorzystujące proponowane własności blokowe przewyższa dwa pozostałe sąsiedztwa nieblokowe pod względem jakości rozwiązania i szybkości zbieżności oraz pozwala uzyskać wysokiej jakości rozwiązania w ciągu kilku sekund.

Dodatkowo zaproponowano metodę wyznaczania długości uszeregowania $C_{\max}$ na modelu obliczeniowym CREW PRAM przy użyciu modyfikacji algorytmu Floyda-Warshalla znajdowania najkrótszych ścieżek w grafach. Metoda ta, dla algorytmów bazujących na lokalnym przeszukiwaniu, pozwala na bardzo duże przyspieszenie (nawet w tysiącach) podczas procesu przeszukiwana sąsiedztwa rozwiązania. Jednak ze względu na bardzo dużą liczbę wymaganych równoległych procesorów metoda ta może być aktualnie stosowana tylko w dużych rozproszonych systemów komputerowych.

Rozdział 5

Problemy z przezbrojeniami

Sprzężenia czasowe będące tematem dysertacji obejmują dział szeregowana zadań związany z przezbrojeniami (*setups*), ponieważ przezbrojenia również opisują relacje między operacjami z różnych zadań na tej samej maszynie. W literaturze przezbrojenia pojawiają się jako jedna z pierwszy modyfikacji dla problemów przepływowych szeregowania zadań [64]. Przezbrojenia mają na celu modelowanie dodatkowych czynności wykonywanych na maszynie pomiędzy przetwarzaniem kolejnych zadań. Takie czynności mogą obejmować czyszczenie, tankowanie, kontrolę, zmianę ustawień maszyny (w tym wymianę różnych końcówek narzędzi itp.) lub po prostu usunięcie obrabianego elementu oraz włożenie kolejnego.

Ze względu na rzeczywistą potrzebę modelowania modernizacji, czyszczenia lub innego przygotowania maszyny między zadaniami, przezbrojenia stały się powszechnie stosowaną koncepcją w harmonogramowaniu. Przegląd najczęściej rozważanych problemów z przezbrojeniami został przedstawiony w Części I. dysertacji.

Zwykle zakłada się, że przezbrojenia mogą być wykonywane niezależnie, czyli w systemie w skład którego wchodzi m maszyn, można wykonać do m przezbrojeń w tym samym czasie. Pod warunkiem, że każde z nich jest wykonywane na innej maszynie. Dzieje się tak zwykle dlatego, że maszyny są bezobsługowe lub istnieją oddzielne załogi dla każdej dostępnej maszyny. Jednak w niektórych procesach produkcyjnych napotykanych w praktyce nie jest możliwe jednoczesne przeprowadzanie przezbrojeń.

Przykład takiego ograniczenia znaleziono w procesie produkcyjnym w firmie zlokalizowanej w Polsce. Proces obejmuje kształtowanie elementów metalowych do różnych urządzeń AGD przy użyciu dużych pras hydraulicznych. Ze względu na wielkość pras do zmiany ich konfiguracji (tzw. sztance ważą około 1 tony) dla różnych rodzajów produktów wymagane są ciężkie

specjalistyczne wózki. Ze względu na koszty oraz załogę wymaganą do obsługi wózka firma posiada tylko jeden taki pojazd. W rezultacie tylko jedna maszyna może być przezbrojona w danym momencie, tzn. przezbrojenia są rozłączne (*disjoint*). Chociaż ograniczenie ograniczające liczbę jednoczesnych przezbrojeń jest realistycznym założeniem, to zwróciło ono niewielką uwagę w literaturze.

W Podrozdziale 5.1 zostanie przedstawiony PFSSP z ciągła pracą maszyn oraz klasycznymi przezbrojeniami [18] (*No idle Permutation Flow Shop Scheduling Problem with Setups*, NPFSSP-S). Natomiast kolejny rozdział zostanie w całości poświęcony problemom szeregowania rozłącznych przezbrojeń w problemie przepływowym (*Flow Shop Scheduling Problem with Disjoint Setups*, FSSP-DS) [50].

5.1 Problem przepływowy bez przestojów

W niniejszym rozdziale przedstawiono problem przepływowy z ciągła pracą maszyn oraz przezbrojeniami zależnymi od kolejności wykonywania zadań. Sformułowano model matematyczny, model grafowy oraz własności, twierdzenia i definicję dedykowanego sąsiedztwa. W celu weryfikacji wyników przeprowadzono badania eksperymentalne z wykorzystaniem różnych algorytmów: (1) Przeszukiwania z Zabronieniami z trzema typami otoczeń oraz (2) Algorytmu Genetycznego.

5.1.1 Definicja problemu

Przez $\mathcal{J} = \{1, 2, \ldots, n\}$ oraz $\mathcal{M} = \{1, 2, \ldots, m\}$ oznaczono kolejno zbiory n zadań oraz m maszyn. Każde zadanie i składa się z m operacji ze zbioru $\mathcal{O}_i = \{l_i + 1,\ l_i + 2,\ \ldots,\ l_i + m\}$, gdzie $l_i = m(i-1)$. Zatem łącznie jest nm operacji. Przez $\mathcal{O}$ oznaczono zbiór wszystkich operacji, w którego skład wchodzą wszystkie operacje z poszczególnych zadań:

$$\mathcal{O} = \bigcup_{i \in \mathcal{J}} \mathcal{O}_i. \tag{5.1}$$

Każde zadanie i musi zostać wykonane na maszynach w kolejności technologicznej: $1 \to 2 \to \cdots \to m$. Czas wykonywania zadania i na maszynie a (tzn. operacji $l_i + a$) został oznaczony przez $p_i^a > 0$. Ponadto, niech $s_{i,j}^a \geqslant 0$ oznacza czas przezbrojenia (*setup*) maszyny a między przetwarzaniem operacji i oraz j na tej maszynie. Dodatkowo, przezbrojenie należy wykonać natychmiast po zakończeniu wykonywania operacji oraz kolejną operację niezwłocznie po zakończeniu wykonywania przezbrojenia.

Kolejność wykonywania operacji w ramach każdego zadania jest stała, ponieważ wynika z ograniczeń technologicznych procesu. Przez π oznaczono n-elementową kolejność zadań. Zatem $\pi(i)$ oznacza wykonywanie zadania jako i-tego w kolejności π. Rozwiązaniem będzie nazywana kolejność π.

Na podstawie rozwiązania π można zbudować odpowiadający mu *harmonogram* $(\boldsymbol{S}, \boldsymbol{C})$ dosunięty w lewo. Harmonogram składa się z dwóch macierzy: (1) macierz momentów rozpoczęcia wykonywania operacji $\boldsymbol{S} = [S_i^a]^{m \times n}$, gdzie S_i^a jest czasem rozpoczęcia wykonywania i-tej operacji na maszynie a oraz (2) macierz momentów zakończenia wykonywania operacji $\boldsymbol{C} = [C_i^a]^{m \times n}$, gdzie C_i^a jest czasem zakończenia wykonywania i-tej operacji na maszynie a. Aby harmonogramu $(\boldsymbol{S}, \boldsymbol{C})$ był dopuszczalny, musi spełniać następujące ograniczenia:

$$\forall a \in \mathcal{M} \quad \forall i \in \mathcal{J} \quad S_i^a, C_i^a \geqslant 0 \tag{5.2}$$

$$\forall a \in \mathcal{M} \quad \forall i \in \mathcal{J} \quad C_i^a = S_i^a + p_i^a, \tag{5.3}$$

$$\forall a \in \mathcal{M} \quad \forall i \in \mathcal{J} \setminus \{1\} \quad s_{i,j}^a \leqslant S_i^a - C_{i-1}^a \leqslant s_{i,j}^a, \tag{5.4}$$

$$\forall a \in \mathcal{M} \setminus \{1\} \quad \forall i \in \mathcal{J} \quad S_i^a \geqslant C_i^{a-1}, \tag{5.5}$$

Bez utraty ogólności można przyjąć moment rozpoczęcia wykonywania pierwszej operacji z zadania pierwszego jako 0, $S_1^1 = 0$. Ograniczenie (5.3) zapewnia ciągłość wykonywania operacji. Ograniczenie (5.4) wymusza ciągła pracę maszyny, ale również wymaga, aby operacja i-ta w kolejności była wykonywana dopiero po zakończeniu wykonania przezbrojenia po poprzedniej operacji $i-1$ (poprzednik maszynowy) na tej samej maszynie a. Podobnie, ograniczenie (5.5) wymusza rozpoczęcie wykonywania operacji i-tej na maszynie a dopiero po zakończeniu wykonywania operacji i-tej na maszynie poprzedniej $a-1$ (poprzedniku technologicznym).

Z ograniczenia (5.3) wynika, że dopuszczalny harmonogram można jednoznacznie opisać za pomocą $\boldsymbol{S}$ lub $\boldsymbol{C}$. Jednak dzięki zastosowaniu zarówno $\boldsymbol{S}$ jaki i $\boldsymbol{C}$, ograniczenia (5.4) i (5.5) są bardziej czytelne oraz łatwiej zauważyć ich wpływ w reprezentacji graficznej rozwiązania. Harmonogram będzie nazywany w lewo dosuniętym (*left-shifted*), jeśli ani jedna operacja nie może rozpocząć się wykonywać wcześniej bez naruszenia ograniczeń lub zmiany kolejności wykonywania zadań. W Podrozdziale 5.1.2 zaproponowano metodę wyznaczającą harmonogram dosunięty w lewo włącznie z uwzględnieniem złożoności obliczeniowej.

Przykład 5.1 W Tabeli 5.1 oraz 5.2 pokazano instancje problemu NPFSSP-S. Na Rysunku 5.1 przedstawiono dosunięty w lewo harmonogram dla tej instancji oraz permutacji $\pi = (2, 5, 4, 3, 1)$.

Tablica 5.1: Czasy wykonywania operacji dla problemu NPFSSP-S z Przykładu 5.1

a	p_1^a	p_2^a	p_3^a	p_4^a	p_5^a
1	1	3	3	2	1
2	2	3	2	1	4
3	1	3	2	6	1

Tablica 5.2: Czasy wykonywania przezbrojeń dla problemu NPFSSP-S z Przykładu 5.1

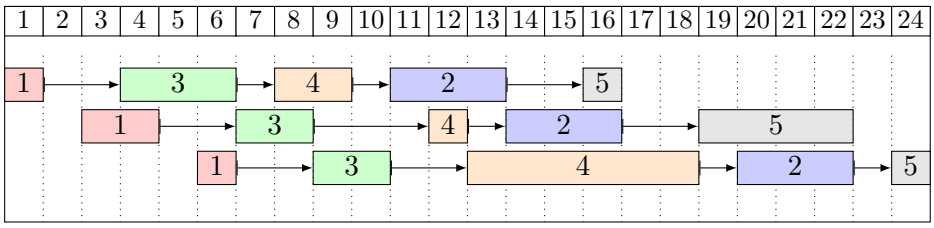
i	$s_{i,1}^1$	$s_{i,2}^1$	$s_{i,3}^1$	$s_{i,4}^1$	$s_{i,5}^1$	$s_{i,1}^2$	$s_{i,2}^2$	$s_{i,3}^2$	$s_{i,4}^2$	$s_{i,5}^2$	$s_{i,1}^3$	$s_{i,2}^3$	$s_{i,3}^3$	$s_{i,4}^3$	$s_{i,5}^3$
1	0	2	**2**	1	1	0	2	**2**	1	1	0	2	**2**	1	2
2	3	0	1	2	**2**	3	0	2	1	**2**	3	0	1	2	**1**
3	3	2	0	**1**	1	3	2	0	**3**	1	3	2	0	**2**	1
4	2	**1**	1	0	3	2	**1**	1	0	1	2	**1**	1	0	1
5	1	4	3	1	0	1	4	1	1	0	3	4	1	2	0

Badanym kryterium optymalizacyjnym będzie długość uszeregowania, czyli czas zakończenia wykonywania wszystkich operacji wraz z przezbrojeniami. Przez $C_{\max}(\pi)$ oznaczono czas zakończenia wykonywania operacji, która zakończyła się wykonywać jako ostatnia w lewo dosuniętym harmonogramie dla kolejności π. Z ograniczeń (5.3)–(5.5) wynika, że będzie to czas zakończenia wykonywania ostatniej operacji w ostatnim zadaniu $C_{\max}(\pi) = C_{\pi(n)}^m$. Celem będzie znalezienie takiego rozwiązania π^{opt}, że:

$$C_{\max}(\pi^{\text{opt}}) = \min_{\pi \in \Pi} C_{\max}(\pi), \tag{5.6}$$

gdzie Π jest zbiorem wszystkich dopuszczalnych rozwiązań. Rozwiązanie π^{opt} będzie nazywane rozwiązaniem optymalnym.

Według rozszerzonej notacji Grahama [57] problem NPFSSP-S (*No idle Permutation Flow Shop Scheduling Problem with Setups*) jest oznaczany jako $FP|\texttt{no idle, setups}|C_{\max}$. W przypadku przezbrojeń niezależnych od kolejności oraz równym czasie wykonywania przezbrojenia w ramach a-tej maszyny $s_{i,j}^a = s_{j,k}^a$, można potraktować problem NPFSSP-S jako PFSSP-TC (*Permutation Flow Shop Scheduling Problem with Time Couplings*), gdzie dla każdej maszyny minimalny oraz maksymalny czas przestoju jest sobie równy $\hat{r}_a = \hat{d}_a$.



Rysunek 5.1: Harmonogram dla rozwiązania π z Przekładu 5.1

Model grafowy

Jedną z klasycznych technik szeregowania zadań jest budowa modelu grafowego dla danego rozwiązania π. Należy utworzyć graf rozwiązania $G(\pi) = (V,\ E_1 \cup E_2)$, gdzie:

$$V = \bigcup_{\substack{a \in \mathcal{M} \\ i \in \mathcal{J}}} \Big\{ (a, i) \Big\}. \tag{5.7}$$

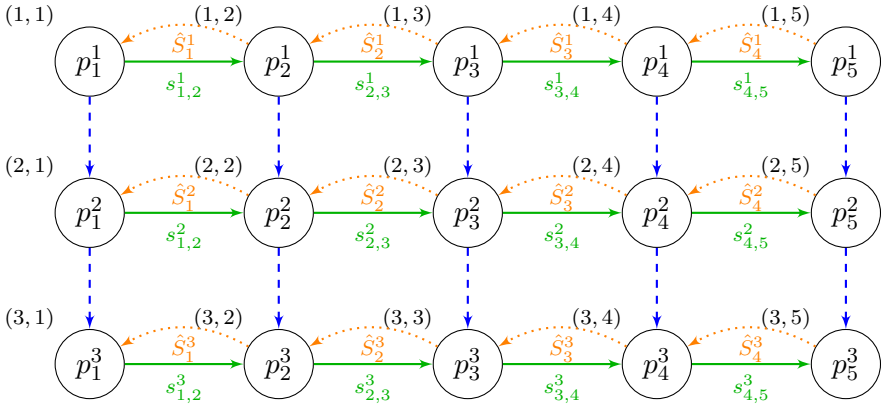
Niech przez V będzie oznaczony zbiór wierzchołków reprezentujących operacje (dla uproszczenia wizualizacji ustawione w formie siatki). Operacja z zadania i-tego na maszynie a jest reprezentowana przez obciążony wierzchołek w i-tej kolumnie oraz a-tym wierszu oznaczony parą (a, i) z wagą p_i^a. Zbiory łuków:

$$E_1 = \bigcup_{\substack{a \in \mathcal{M} \\ i \in \mathcal{J} \setminus \{n\}}} \Big\{ \Big((a, i),\ (a, i+1) \Big) \Big\}, \tag{5.8}$$

$$E_2 = \bigcup_{\substack{a \in \mathcal{M} \setminus \{m\} \\ i \in \mathcal{J}}} \Big\{ \Big((a, i),\ (a+1, i) \Big) \Big\}. \tag{5.9}$$

Łuki grafu $G(\pi)$ reprezentują ograniczenia problemu: (1) łuki poziome E_1 – ograniczenia maszynowe oraz (2) łuki pionowe E_2 – ograniczenia technologiczne. Długość najdłużej ścieżki dochodzącej do danego wierzchołka (a, i) jest równa odpowiedniemu czasowi zakończenia wykonywania C_i^a. Przez długość ścieżki rozumie się sumę wag wierzchołków i krawędzi należących do ścieżki.

Kolejnym krokiem jest rozszerzenie grafu $\mathcal{G}(\pi)$. Każdemu łukowi „poziomemu” $((a, i),\ (a, i+1))$ w zbiorze E_1 przypisano wagę $s_{i,j}^a$. Po drugie,



Rysunek 5.2: Przykładowy graf $\mathcal{G}(\pi)$ dla problemu NPFSSP-S

aby wymusić ciągła prace maszyn należy wprowadzić nowy zbiór łuków:

$$E_3 = \bigcup_{\substack{a \in \mathcal{M} \\ i \in \mathcal{J} \setminus \{n\}}} \Big\{ \big((a, i+1),\ (a, i) \big) \Big\}. \tag{5.10}$$

Każdy łuk „powrotny" $((a, i+1),\ (a, i))$ w zbiorze E_3 ma wagę $\hat{S}_i^a$, zdefiniowaną jako:

$$\hat{S}_i^a = -p_i^a - p_{i+1}^a - s_{i,i+1}^a. \tag{5.11}$$

Strukturę grafu $\mathcal{G}(\pi)$ dla problemu o rozmiarze 5×3 oraz kolejności $\pi = (1, 2, 3, 4, 5)$ przedstawiono na Rysunku 5.2. Dla ułatwienia czytelności łuki E_1, E_2 oraz E_3 oznaczono kolejno zieloną linią ciągła, niebieską przerywaną oraz pomarańczowa kropkowaną.

5.1.2 Własności

Poniżej zostaną przedstawione oraz omówione nowe własności i twierdzenia dla problemu $FP|\texttt{no idle, setups}|C_{\max}$, w tym: własność o braku cykli dodatnich, dopuszczalności rozwiązań, tworzeniu harmonogramu oraz definicji dedykowanego sąsiedztwa. Według wiedzy autora własności te nie były publikowane w literaturze.

Własność 5.1 *Dla dowolnego rozwiązania $\pi \in \Pi$, graf $\mathcal{G}(\pi)$ dla problemu $FP|\texttt{no idle, setups}|C_{max}$ nie zawiera cykli o dodatniej długości.* □

Dowód. Dowód jest analogiczny jak dowód Własności 5.1 dla problemu $FP|\texttt{minimal idle, maximal idle}|C_{\max}$, ponieważ również w tym wypadku

graf nie zawiera krawędzi idących w górę. Podobnie w tym przypadku cykl może powstać tylko za pomocą łuków „poziomych" oraz „powrotnych" (w lewo), których struktura jest niezmienna. ■

W związku z tym, wszystkie cykle w grafie $\mathcal{G}(\pi)$ mają długość niedodatnia. Obliczając czas zakończenia wykonywania wszystkich zadań, będzie można zignorować cykle zgodnie z następującą obserwacją.

Wniosek 5.1 Dla każdego rozwiązania π, jeśli graf rozwiązania $\mathcal{G}(\pi)$ dla $FP|\texttt{no idle, setups}|C_{\max}$ zawiera ścieżkę p o długości $\text{len}(p)$ taka, że p zawiera cykl, to graf $\mathcal{G}(\pi)$ zawiera również ściekę p' bez cyklu, taką że:

$$\text{len}(p') \geqslant \text{len}(p). \tag{5.12}$$

□

Zatem, jeśli ścieżka p w grafie $\mathcal{G}(\pi)$ zawiera cykl, to: (1) nie jest to najdłuższa ścieżka w grafie $\mathcal{G}(\pi)$ (przynajmniej jeden cykl w ścieżce p ma ujemną długość) lub (2) możemy skonstruować inną ścieżkę zaczynającą się i kończącą dokładnie w tym samym wierzchołku, ale niezawierającej cyklu (wszystkie cykle w ścieżce p mają długość zero).

Twierdzenie 5.1 *Dla każdego rozwiązania $\pi \in \Pi$ permutacyjnego problemu $FP|\texttt{no idle, setups}|C_{max}$, istnieje harmonogram $(\boldsymbol{S}, \boldsymbol{C})$, który jest dopuszczalny oraz dosunięty w lewo. Ponadto, wartość kryterium $C_{max}(\pi)$ dla harmonogramu $(\boldsymbol{S}, \boldsymbol{C})$ można wyznaczyć w czasie $O(nm)$.* □

Dowód. Dowód jest analogiczny jak dowód Twierdzenia 4.1 dla problemu $FP|\texttt{minimal idle, maximal idle}|C_{\max}$ na stronie 61 . Należy uwzględnić czas odpowiedniego przezbrojenia zamiast minimalnego czasu przestoju na maszynie a. Ponadto w drugiej fazie (kiedy operacje są dosuwane w prawo) trzeba ustawić zadania bez przerw między przezbrojeniami i operacjami. Modyfikację algorytmu dla $FP|\texttt{no idle, setups}|C_{\max}$ pokazano w Algorytmie 12 ze zmianami w linii 6. oraz 8. Zmiana nie wpływa na złożoność obliczeniową algorytmu.

Wniosek 5.2 Dla dowolnej kolejności wykonywania $\pi \in \Pi$ dla problemu $FP|\texttt{no idle, setups}|C_{\max}$ istnieje co najmniej jeden dopuszczalny harmonogram $(\boldsymbol{S}, \boldsymbol{C})$ (zatem wszystkie kolejności są dopuszczalne). □

Dowód. Metodę wyznaczania harmonogramu przedstawioną w dowodzie Twierdzenia 5.1 można zastosować dla dowolnej kolejności π, ponieważ przy zmianie kolejności zmianie ulegają są jedynie aktualnie wartości p_i^a, które są używanie do ustalania czasów C_i^a bazujących na aktualnych wartościach S_i^a. W ten sposób przy użyciu algorytmu tworzony jest dopuszczalny harmonogram $(\boldsymbol{S}, \boldsymbol{C})$ dla dowolnego π. ■

Algorytm 12: Konstruowanie harmonogramu dosuniętego w lewo

Dane: $\boldsymbol{\pi}$: kolejność wykonywania zadań
Wynik: $\boldsymbol{C}$: macierz momentów zakończenia operacji

```
1  function Evaluate(π):
2      C_1^1 ← p^1_{π(1)};
3      for i = 2, 3, ..., n do
4          C_i^1 ← C_{i-1}^1 + r̂_0 + p^1_{π(i)};
5      for a = 2, 3, ..., m do
6          C_1^a ← C_i^{a-1} + p^a_{π(1)};
7          for i = 2, 3, ..., n do
8              C_i^a ← max{C_{i-1}^a + s^a_{π(i),π(i+1)}, C_i^{a-1}} + p^a_{π(i)};
9          for i = n − 1, n − 2, ..., 1 do
10             C_i^a ← C_{i+1}^a − p^a_{π(i+1)} − s^a_{π(i),π(i+1)};
11     return C;
```

Twierdzenie 5.2 *Szeregowanie operacji na każdej maszynie $a \in \mathcal{M}$ można przedstawić jako problem szukania najkrótszej ścieżki w grafie.* □

Dowód. Niech $\text{len}^a(p^a)$ oznacza odległość między wierzchołkami $(a, 1)$ oraz (a, n) na maszynie $a \in \mathcal{M}$, która jest sumą czasów wykonywania operacji na tej maszynie oraz czasów wykonywania przezbrojeń między nimi:

$$\text{len}^a(p^a) = p^a_{\pi(1)} + \sum_{j \in \mathcal{J} \setminus \{1\}} (s^a_{\pi(i-1),\pi(i)} + p^a_{\pi(i)}), \tag{5.13}$$

gdzie p^a to ścieżka od wierzchołka$(a, 1)$ do (a, n) na maszynie $a \in \mathcal{M}$. Czas wykonywania każdej operacji jest stały oraz niezależny od kolejności wykonywania zadań, więc

$$\text{len}(p^a) = \sum_{j \in \mathcal{J} \setminus \{1\}} (p^a_i) + \sum_{j \in \mathcal{J} \setminus \{1\}} (s^a_{\pi(i-1),\pi(i)}). \tag{5.14}$$

Dlatego podczas minimalizacji długości uszeregowania na maszynie a można pominąć czasy wykonywania operacji i skupić się tylko na minimalizowaniu sumy czasów wykonywania przezbrojeń co jest równoważne z szukaniem najkrótszej ścieżki Hamiltona, gdzie odległości między wierzchołkami będą dane przez macierz przezbrojeń dla maszyny a:

$$\min \text{len}^a(p^a) = \min_{\pi \in \Pi} \sum_{j \in \mathcal{J} \setminus \{1\}} (s^a_{\pi(i-1),\pi(i)}). \tag{5.15}$$

■

Wniosek 5.3 Podczas tworzenia sąsiedniego rozwiązania dla dowolnej oraz dopuszczalnej kolejności wykonywania $\pi \in \Pi$ można wybrać jeden z atrybutów ruchu tylko na podstawie czasów wykonywania przezbrojeń. Wszystkie rozwiązania sąsiednie są dopuszczalne oraz ich liczba wynosi $n-1$. □

Dowód. Dla problemu $FP|\texttt{no idle, setups}|C_{\max}$ kolejność wykonywania zadań na każdej maszynie jest taka sama. Należy potraktować problem szukania kolejności wykonywania zadań jako problem szukania najkrótszej ścieżki Hamiltona dla każdego poziomu w grafie. Trzeba uwzględnić sumę czasów wykonywania przezbrojeń łącznie na wszystkich maszynach, ale przed każdym zadaniem. Następnie należy wyznaczyć zadanie i' przed którym suma czasów wykonywania przezbrojeń jest jak największa:

$$i' = \max_{i \in \mathcal{J} \setminus \{1\}} \sum_{a \in \mathcal{M}} s^a_{\pi(i-1),\pi(i)}. \tag{5.16}$$

Następnie sąsiednie rozwiązania otrzymuje się przez zamianę wartości $\pi(i')$ oraz $\pi(j)$, gdzie $j \in \mathcal{J} \setminus \{i'\}$. Rozmiar sąsiedztwa wynosi dokładnie $n-1$, ponieważ tyle jest możliwych zamian. ■

5.1.3 Algorytmy optymalizacyjne

Poniżej zostaną opisane dwie metody rozwiązania problemu: (1) Algorytm Genetyczny (wchodzący w skład pakietu optymalizacyjnego `DEAP`) oraz (2) Przeszukiwanie z Zabronieniami.

Algorytm Genetyczny

Do implementacji GA użyto predefiniowanego modelu `DEAP` (*Distributed Evolutionary Algorithms in Python*) [41]. Wspomniany model jest używany do implementacji i uruchamiania metod ewolucyjnych dla szerokiej klasy problemów optymalizacyjnych [37, 107] w tym dla FSSP (*Flow Shop Scheduling Problem*) [110].

Pakiet optymalizacyjny `DEAP` pozwala na skorzystanie z wielu wbudowanych funkcji oraz możliwość odpowiedniego ustawienia parametrów dla operatorów genetycznych według potrzeb:

- *inicjalizacja* – sposób generowania osobników w populacji początkowej (losowa permutacja, ciąg liczb rzeczywistych lub lista dowolnych elementów),
- *selekcja* – proces doboru osobników do kolejnej populacji lub/oraz dobór rodziców z bieżącej populacji; dostępne są najbardziej popularne rodzaje selekcji: turniejowa, ruletka, losowa, najlepsi, najgorsi,

ponadto jest również możliwość wyboru selekcji *NSGA2* lub *NSGA3* dla problemów z wielokryterialną funkcją celu,
- *krzyżowanie* – wymiana informacji pomiędzy parą osobników; dostępnych jest ponad 10 rodzajów krzyżowania w tym najczęściej używane w literaturze: (1) z porządkowaniem (*Ordered Crossove*,r OX), (2) z częściowym odwzorowaniem (*Partially Matched Crossover*, PMX),
- *mutacja* – losowa zmiana w genotypie osobnika; dostępnych jest 6 różnych mutacji, w tym losowa zamiana genów znajdujących się na różnych pozycjach (*Shuffle Indexes*),
- *algorytm* – pakiet pozwala również na wybór stosowanej metody, w tym prostej metody ewolucyjnej (*Simple Evolutionary Algorithm*).

Każda z wymienionych metod udostępnia szereg możliwych do ustawienia parametrów, miedzy innymi: (1) rozmiar populacji, (2) liczba generacji, (3) prawdopodobieństwo krzyżowania, (4) prawdopodobieństwo mutacji, (5) rozmiar turnieju i wiele innych. Ponadto pakiet udostępnia wiele rożnych funkcji, między innymi statystyczne lub do wizualizacji przebiegu GA (np.: najlepszy osobnik lub średnia wartość przystosowania wszystkich osobników we wszystkich pokoleniach).

Przeszukiwanie z Zabronieniami

Przeszukiwanie z Zabronieniami (*Tabu Search*, TS) [56] jest zaawansowaną wersją metody poszukiwania lokalnego. W celu znalezienia najlepszego sąsiada jest przeglądane całe sąsiedztwo. Sąsiedztwo $\mathcal{N}(\pi)$ rozwiązania π jest zdefiniowane jako wszystkie rozwiązania π', które można otrzymać z π po przez predefiniowaną funkcją *ruchu*. Dla badanego problemu rozpatrzono trzy typu ruchów zamień

- *normalny* – $\mathtt{S}(\pi, i, j)$: sąsiednie rozwiązanie π' otrzymuje się przez zamianę wartości $\pi(i)$ oraz $\pi(j)$, gdzie $i \in \mathcal{J}$, $j \in \mathcal{J} \setminus \{i\}$; sąsiedztwo typu zamień ma dokładnie $\frac{n(n-1)}{2}$ rozwiązań.
- *sąsiedni* – $\mathtt{A}(\pi, i)$: sąsiednie rozwiązanie π' uzyskuje się poprzez jednoczesną zamianę sąsiednich wartości $\pi(i)$ oraz $\pi(i{+}1)$, gdzie $i \in \mathcal{J}\setminus\{n\}$; sąsiedztwo typu zamień sąsiedni ma dokładnie $(n-1)$ rozwiązań.
- *maksymalny* – $\mathtt{M}(\pi, j)$: ruch zbliżony do $\mathtt{S}(\pi, i, j)$, ale zadanie do zamiany jest wybieranie na bazie dowodów Twierdzenia 5.2 oraz Wniosku 5.3 ze strony 103. Rozmiar sąsiedztwa typu zamień maksymalny również wynosi $(n-1)$.

Najlepszy sąsiad może być gorszy niż aktualne rozwiązanie, dlatego w celu zapobiegnięcia wejścia w nieskończony cykl, w algorytmie implementuje

się pamięć krótkotrwałą, tak zwaną listę zakazów. Na liście przechowuje się zazwyczaj atrybuty ruchu, przy pomocy którego został wygenerowany najlepszy sąsiad. Rozwiązania uzyskane poprzez zabronione ruchy są ignorowane przy wyborze najlepszego sąsiada (z pewnymi wyjątkami). W celu szybkiego przeglądania listy zakazów należy zaimplementować ją w formie dwuwymiarowej macierzy o wymiarach $n \times n$, gdzie wartość elementu macierzy będzie zawierać informację od której iteracji algorytmu dany ruch będzie już dostępny.

5.1.4 Eksperymenty obliczeniowe

Celem eksperymentów było zbadanie wpływu danych wejściowych na działanie zaimplementowanych metod przybliżonych: (1) Algorytmu Genetycznego oraz (2) Przeszukiwania z Zabronieniami. Do testów użyto instancji zaproponowanych przez Ruiza [111]. Użyte instancje bazowały na instancjach Taillarda dla FSSP [129], ale zostały rozszerzone o czasy wykonywania przezbrojeń. Instancje są podzielone na cztery grupy (w zależności od rozrzutu)wylosowane z rozkładu równomiernego: (1) 1–9, (2) 1–49, (3) 1–99 oraz (4) 1-124. Grupy instancji będą w dalszej części dysertacji nazywane odpowiednio: (1) SDST10, (2) SDST50, (3) SDST100 oraz (4) SDST125. Instancje w ramach każdej grupy są podzielone według rozmiarów zaproponowanych przez Taillarda, jednak użyto 11 grup spośród 12 pomijając instancje o rozmiarze 500×20. W ramach każdej takiej grupy względem rozmiaru problemu jest 10 instancji. Zatem łącznie w testach użyto 440 różnych instancji. Jako miarę jakości zastosowano procentowe względne odchylenie (*Percentage Relative Deviation*, PRD).

Eksperymenty zostały przeprowadzone na komputerze z procesorem Intel Core i7-6700K 4,0 GHz oraz 16 GB pamięci RAM. Kod został napisany w języku programowania `Python` i został uruchomiony pod kontrolą systemu operacyjnego MS Windows 10.

W eksperymencie zostały porównane trzy wersje algorytmu TS oraz jedna wersja GA. Każda wersja algorytmu TS bazowała na przeglądzie innego typu sąsiedztwa (TSS dla ruchu zamień, TSA dla ruchu zamień sąsiedni oraz TSM dla ruchu zamień maksymalny, patrz str. 104). Warunkiem zatrzymania algorytmu we wszystkich trzech wersjach był limit czasowy ustalony przez czas wykonywania się GA z następującymi parametrami: liczba pokoleń – 25, wielkość populacji – 50, selekcja turniejowa o rozmiarze 3, krzyżowanie OX, prawdopodobieństwo krzyżowania – 80%, mutacja generująca sąsiada w otoczeniu typu zamień, prawdopodobieństwo mutacji 5%. Ponadto rozwiązanie $\pi_b^* est$ zostało wybrane jako najlepsze ze wszyst-

kich badanych algorytmów dla danej instancji. Wyniki przedstawiono w Tabeli 5.3.

Należy zwrócić uwagę, że przy użyciu algorytmu TSM najlepsze rozwiązanie udało się znaleźć zawsze aż dla 17 różnych grup instancji (na 44 grupy). Ponadto najlepsze rozwiązanie zostało znalezione dla $65,2\%$ wszystkich instancji. Dla żadnej grupy: (1) SDST10, (2) SDST50, (3) SDST100 oraz (4) SDST125, średnie PRD nie przekroczyło $1,17\%$. Wydłużenie czasów wykonywania przezbrojeń miało niewielki wpływ na efektywność tego otoczenia, można zaobserwować poprawę rzędu $0,1\%$ przy wydłużonych czasach przezbrojeń w grupach: SDST100 i SDST125.

Drugim najlepszym algorytmem okazał się GA dla którego średnie PRD dla żadnej z czterech grup nie przekroczyło 5%, a znaleziono najlepsze rozwiązanie dla $21,5\%$ instancji. Zauważyć można pogorszenie efektywności GA wraz z wydłużeniem czasów wykonywania przezbrojeń (w szczególności dla $s^a_{i,j} \geqslant p^a_i$). Jednak algorytm okazał się najlepszy dla największych instancji ($n = 200$) w szczególności dla grupy SDST10, gdzie zawsze znalazł najlepsze rozwiązanie.

Poprzez zastosowanie metoda TSS znaleziono dobre wyniki dla małych instancji ($n = 20$). Jednak dla pozostałych grup instancji średnie PRD wynosiło nawet około 20%, a najlepsze rozwiązanie udało się znaleźć dla $15,7\%$ instancji. „Najsłabszą" metodą okazało się TS z sąsiedztwem typy zamień sąsiedni, gdzie najlepsze rozwiązania udało się znaleźć jedynie dla $0,5\%$ instancji. Jednak średnie PRD okazało się lepsze niż dla pełnego sąsiedztwa typy zamień.

Wyniki wyraźnie wskazują, że standardowe metody przeglądania sąsiedztwa (zamień lub zamień sąsiedni) w metodach bazujących na TS nie radzą sobie z problemem $FP|\texttt{no idle, setups}|C_{\max}$. Implementacja dedykowanego sąsiedztwa wyraźnie poprawiła jakość dostarczanych rozwiązań nawet względem dedykowanego pakietu optymalizacyjnego `DEAP` dla GA.

5.1.5 Wnioski i uwagi

W tym rozdziale został przedstawiony permutacyjny problem przepływowy z ciągła pracą maszyn oraz przezbrojeniami zależnymi od kolejności wykonywania zadań NPFSSP-S. Dla badanego problemu sformułowano model matematyczny oraz grafowy. Następnie sformułowano własności w tym metodę wyznaczania wartości funkcji celu oraz dedykowany typ sąsiedztwa.

Przedstawiono dwie metody: (1) Algorytm Genetyczny z pakietu `DEAP` oraz (2) metaheurystykę Przeszukiwania z Zabronieniami przeglądającą trzy typy otoczeń. Eksperymenty komputerowe przeprowadzono na instan-

Tablica 5.3: Średnie PRD oraz czas działania T dla algorytmów TS oraz metody GA

$n \times m$	*PRD* [%]																T[s]
	SDST10				*SDST50*				*SDST100*				*SDST125*				
	GA	TSS	TSA	TSM	GA	TSS	TSA	TSM	GA	TSS	TSA	TSM	GA	TSS	TSA	TSM	
20 × 5	5,35	0,65	11,43	1,68	6,24	0,73	13,59	1,31	9,55	0,49	13,57	1,50	8,17	0,96	14,24	1,51	0,45
20 × 10	5,75	1,20	12,13	1,60	6,22	0,38	11,84	1,77	6,50	1,34	10,61	0,72	5,31	2,44	9,76	0,31	0,73
20 × 20	3,87	0,91	10,20	1,65	4,66	0,49	10,20	2,22	5,78	1,24	9,33	0,85	5,26	0,88	10,30	1,39	1,29
50 × 5	4,44	6,53	8,69	0,00	6,99	12,92	9,49	0,00	10,26	16,54	11,67	0,00	10,78	17,13	11,60	0,00	1,02
50 × 10	5,95	12,03	13,56	0,00	6,63	16,40	9,97	0,00	7,58	17,51	7,49	0,00	7,21	19,17	8,41	0,00	1,77
50 × 20	5,44	13,10	13,40	0,00	5,79	19,26	10,84	0,00	3,69	17,87	7,37	0,01	6,35	20,05	6,95	0,00	3,24
100 × 5	2,30	9,45	8,30	0,05	3,40	13,05	4,88	0,00	4,52	14,99	4,35	0,00	5,01	16,32	5,01	0,00	2,01
100 × 10	1,52	11,43	9,80	0,26	1,66	15,09	5,02	0,09	1,98	16,21	2,51	0,00	2,41	16,63	2,08	0,21	3,51
100 × 20	2,48	10,61	9,55	0,11	3,62	14,65	6,00	0,00	3,66	16,07	3,42	0,00	3,53	16,38	1,96	0,42	6,75
200 × 10	0,00	7,84	8,24	3,35	0,04	10,11	6,96	4,29	0,79	9,80	4,82	2,53	0,32	10,78	5,43	3,28	7,22
200 × 20	0,00	8.57	9.18	3.13	0,15	9,75	6,21	3,20	0,00	9,78	5,47	4,23	0,26	8,52	3,90	2,92	16,06
średnia	3,37	7,49	10,41	1,08	4,13	10,26	8,64	1,17	4,94	11,08	7,33	0,89	4,96	11,75	7,24	0,91	-

GA – Algorytm Genetyczny TSS – Przeszukiwanie z Zabronieniami z otoczeniem typu zamień normalny
TSA – Przeszukiwanie z Zabronieniami z otoczeniem typu zamień sąsiedni
TSM – Przeszukiwanie z Zabronieniami z otoczeniem typu zamień maksymalny

cjach zaproponowanych przez Ruiza. Wyniki wykazały przewagę TS z dedykowanym otoczeniem nad innymi metodami, przy średnim PRD rzędu 1% oraz znalezieniem najlepszego rozwiązania dla $65,2\%$ instancji.

5.2 Szeregowania rozłącznych przezbrojeń

W niniejszym rozdziale przedstawiony zostanie problem szeregowania rozłącznych przezbrojeń dla problemu przepływowego szeregowania zadań dla ustalanej kolejności wykonywania zadań FSSP-DS. Sformułowano model matematyczny, a następnie zaproponowano własności eliminacyjne i blokowe. Pokazano powiązanie rozmiaru przestrzeni rozwiązań z liczbami Catalana. Przeprowadzono badania eksperymentalne z wykorzystaniem modelu MILP oraz dedykowanej heurystyki bazującej na własności eliminacyjnej dla stałej liczby maszyn, oraz badania z wykorzystaniem metaheurystyki bazującej na Przeszukiwaniu z Zabronieniami wykorzystującej własności eliminacyjne.

5.2.1 Definicja problemu

Niech $\mathcal{J} = \{1, 2, \ldots, n\}$ oraz $\mathcal{M} = \{1, 2, \ldots, m\}$ będą odpowiednio zbiorami n zadań oraz m maszyn. Zadanie i składa się ze zbioru m operacji $\mathcal{O}_i = (o_i^1, o_i^2, \ldots, o_i^m)$. Przez p_i^a oznaczono czas wykonywania operacji o_i^a. Przez $\mathcal{O}$ oznaczono zbiór wszystkich operacji, w którego skład wchodzą wszystkie operacje ze wszystkich zadań:

$$\mathcal{O} = \bigcup_{i \in \mathcal{J}} \mathcal{O}_i. \tag{5.17}$$

Pomiędzy każdymi dwoma następującymi po sobie zadaniami $i, i+1,\ i \in \mathcal{J} \setminus \{n\}$, wykonywanymi na tej samej maszynie $a \in \mathcal{M}$ należy wykonać przezbrojenie. Przezbrojenie nie może być przerwane oraz musi być wykonywanie bez przerywania przez $s_i^a > 0$ czasu. Ograniczenia tak skonstruowanego systemu wytwarzania można podsumować następująco:

1. Każda operacja w zadaniu i jest wykonywania na innej maszynie, a operacje są przetwarzane w kolejności występowania w $\mathcal{O}_i$,
2. Kolejność wykonywania zadań jest taka sama na każdej maszynie,
3. Każda operacja o_i^a jest wykonywania przez p_i^a czasu bez przerywania,
4. Każde zadanie może być przetwarzania w danym czasie tylko przez jedną maszynę,
5. Pomiędzy każdymi dwoma następującymi po sobie zadaniami jest wykonywane przezbrojenie,

6. Jednocześnie może być wykonywane co najwyżej jedno przezbrojenie (model *single server* lub *disjoint setups*).

Zatem maszyna może w danym momencie: (1) wykonać pojedynczą operację, (2) wykonać pojedyncze przezbrojenie, albo (3) być bezczynna. Całkowita liczba przezbrojeń do wykonania w opisanym systemie wynosi $m(n-1)$ (jest do wykonania $n-1$ przezbrojeń na każdej z m maszyn). Przez $\mathcal{S} = \{1, 2, \ldots, (n-1)m\}$ zdefiniowano zbiór wszystkich przezbrojeń.

Rozwiązanie problemu można przedstawić za pomocą harmonogramu. Harmonogram to zestaw momentów rozpoczęcia oraz zakończenia wykonywania wszystkich operacji oraz przezbrojeń. Przez S_i^a oraz C_i^a, $a \in \mathcal{M},\ i \in \mathcal{J}$ zostały oznaczone odpowiednio momenty rozpoczęcia oraz zakończenia wykonywania operacji z i-tego zadania wykonywanego na maszynie a. Analogicznie przez ${}^{\sigma}S_i^a$ oraz ${}^{\sigma}C_i^a$, $a \in \mathcal{M},\ i \in \mathcal{J}\setminus\{n\}$ zostały oznaczone momenty rozpoczęcia oraz zakończenia wykonywania przezbrojeń na maszynie a po zadaniu i. Przy użyciu wprowadzonej notacji, ograniczenia problemu można sformalizować następująco:

$$\forall a \in \mathcal{M}\ \forall i \in \mathcal{J} \qquad 0 \leqslant C_i^a, S_i^a, {}^{\sigma}C_i^a, {}^{\sigma}S_i^a, \tag{5.18}$$

$$\forall a \in \mathcal{M}\ \forall i \in \mathcal{J} \qquad C_i^a = S_i^a + p_i^a, \tag{5.19}$$

$$\forall a \in \mathcal{M}\ \forall i \in \mathcal{J} \setminus \{n\} \qquad {}^{\sigma}C_i^a = {}^{\sigma}S_i^a + s_i^a, \tag{5.20}$$

$$\forall a \in \mathcal{M} \setminus \{m\}\ \forall i \in \mathcal{J} \qquad C_i^a \leqslant S_i^{a+1}, \tag{5.21}$$

$$\forall a \in \mathcal{M}\ \forall i \in \mathcal{J} \setminus \{n\} \qquad {}^{\sigma}C_i^a \leqslant S_{i+1}^a, \tag{5.22}$$

$$\forall a \in \mathcal{M}\ \forall i \in \mathcal{J} \setminus \{n\} \qquad C_i^a \leqslant {}^{\sigma}S_i^a, \tag{5.23}$$

$$\begin{array}{l}\forall a, b \in \mathcal{M},\ a \neq b \\ \forall i, j \in \mathcal{J} \setminus \{n\},\ i \neq j\end{array} \qquad {}^{\sigma}C_i^a \leqslant {}^{\sigma}S_j^b \ \veebar\ {}^{\sigma}C_j^b \leqslant {}^{\sigma}S_i^a, \tag{5.24}$$

gdzie "$\veebar$" jest alternatywą rozłączną (*exclusive or*, XOR). Powyższe ograniczenia (5.19) oraz (5.20) gwarantują, że operacje oraz przezbrojenia nie będą przerywane. Ograniczenie (5.21) reprezentuje sekwencyjne wykonywanie zadania (ograniczenia technologiczne). Ograniczenia (5.22) oraz (5.23) gwarantują, że przezbrojenia i operacje wykonywane na tej samej maszynie nie nakładają się oraz są wykonywane w predefiniowanej kolejności (ograniczenia maszynynowe). Ograniczenia (5.24) zapewnia, że w dowolnym momencie można wykonywać w systemie co najwyżej jedne przezbrojenie. Harmonogram będzie nazywany *dopuszczalnym* jeśli spełnia wszystkie ograniczenia (5.19)–(5.24). Problem polega na znalezieniu takiego dopuszczalnego harmonogramu by, *długość uszeregowania* $C_{\max} = C_n^m - S_1^1$ była jak najmniejsza. Bez utraty ogólności można ustalić czas rozpoczęcia przetwarzania pierwszego zadania $S_1^1 = 0$. Z powyższych ograniczeń oraz przepływowego charakteru problemu wynika, że $C_{\max} = C_n^m$.

Reprezentacja rozwiązania jako harmonogramu prowadzi do nieskończonego zbioru możliwych rozwiązań. Dlatego reprezentacji przy użyciu ograniczeń używa się jedynie do analizy teoretycznej lub w solverach obliczeniowych takich, jak `CPLEX` czy `Gurobi`. Mimo stałej kolejności wykonywania zadań to kolejność wykonywania przezbrojeń nie jest stała. Niech $\boldsymbol{\tau} \in \mathcal{T}$ opisuje kolejność w jakiej są wykonywane przezbrojenia, gdzie τ_k, $k \in \mathcal{S}$, oznacza k-te wykonywane przezbrojenie w systemie. Przez $\mathcal{T}$ oznaczono zbiór wszystkich możliwych kolejności wykonywania przezbrojeń. Przezbrojenia są identyfikowane przez parę liczb (a, i), gdzie $a \in \mathcal{M}$ jest maszyną na której jest wykonywane przezbrojenie oraz $i \in \mathcal{J} \setminus \{n\}$ jest zadaniem po którym jest wykonywane przezbrojenie. Liczba wszystkich możliwych kombinacji wykonywania przezbrojeń $|\mathcal{T}|$ jest równa $((n-1)m)!$. Kolejność wykonywania przezbrojeń będzie nazywana dopuszczalną, jeśli opisuję co najmniej jeden dopuszczalny harmonogram. Kolejność wykonywania przezbrojeń na każdej maszynie jest stała, więc każdą dopuszczalną kolejność wykonywania przezbrojeń można opisać jednoznacznie przez kolejność maszyn, na których są wykonywane: $\boldsymbol{\sigma} \in \Sigma$, gdzie $\sigma(k)$, $k \in \mathcal{S}$ jest maszyną, na której jest wykonywane k-te przezbrojenie. W dalszej części pracy opisując kolejność wykonywania przezbrojeń będą wykorzystywane obie notacje: $\boldsymbol{\sigma}$ lub $\boldsymbol{\tau}$, w zależności od kontekstu. W celu transformacji dowolnego dopuszczalnego $\boldsymbol{\tau} \in \mathcal{T}$ do unikatowej $\boldsymbol{\sigma} \in \Sigma$, można zignorować numer zadania z każdej pary opisującej przezbrojenie. Możliwa jest również procedura odwrotna poprzez przypisanie kolejnych zadań do ustawień wykonanych na tej samej maszynie (Przykład 5.2). Co ciekawe, czasami niedopuszczalne $\boldsymbol{\tau}$ można przekształcić w dopuszczalną $\boldsymbol{\sigma}$ (co również przedstawiono w przykładzie). Liczba różnych $\boldsymbol{\sigma}$, oznaczonych przez $|\Sigma|$, jest mniejsza niż $|\mathcal{T}|$ i wynosi dokładnie:

$$|\Sigma| = \frac{|\mathcal{T}|}{(n-1)!^m} = \frac{((n-1)m)!}{(n-1)!^m}. \tag{5.25}$$

Zbiór wszystkich dopuszczalnych $\boldsymbol{\sigma}$ został oznaczy przez Σ_{feas}.

Przykład 5.2 Można rozważyć kolejność wykonywania przezbrojeń dla instancji o rozmiarze $n = 3$ oraz $m = 2$:

$$\boldsymbol{\tau}^1 = \big((1,3),\ (1,2),\ (1,1),\ (2,3),\ (2,2),\ (2,1)\big). \tag{5.26}$$

Kolejność wykonywania przezbrojeń $\boldsymbol{\tau}^1$ jest niedopuszczalna, ponieważ $\tau_1^1 = (1,3)$ jest zaplanowana jako pierwsza, podczas gdy operacja 1 z zadania 3 nie może zacząć się wykonywać, ponieważ przezbrojenie je poprzedzające $(1,2)$ nie zostało wykonane. Następnie rozważmy Kolejność wykonywania

przezbrojeń $\boldsymbol{\sigma}^1 = (1, 1, 1, 2, 2, 2)$, która została utworzona na podstawie $\boldsymbol{\tau}^1$. Ta kolejność wykonywania przezbrojeń jest dopuszczalna oraz może zostać przekształcona w dopuszczalną kolejność wykonywania przezbrojeń:

$$\boldsymbol{\tau}^2 = \big((1,1),\ (1,2),\ (1,3),\ (2,1),\ (2,2),\ (2,3)\big). \tag{5.27}$$

Następnie rozważono niedopuszczalną kolejność wykonywania przezbrojeń:

$$\boldsymbol{\tau}^3 = \big((2,1),\ (2,2),\ (2,3),\ (1,1),\ (1,2),\ (1,3)\big). \tag{5.28}$$

Odpowiadająca jej kolejność wykonywania przezbrojeń $\boldsymbol{\sigma}^2 = (2, 2, 2, 1, 1, 1)$ również jest niedopuszczalna. □

Aby kolejność wykonywania przezbrojeń $\boldsymbol{\sigma}$ mogła być efektywną reprezentacją rozwiązania, wymagana jest szybka metoda przekształcenia dopuszczalnego $\boldsymbol{\sigma} \in \Sigma_{\text{feas}}$ w unikalny, dopuszczalny harmonogram. Dyskusją ograniczono do harmonogram dosuniętego w lewo, to znaczy harmonogram, w których żadne przezbrojenie ani operacja nie może być wykonana wcześniej, bez zmiany kolejności przezbrojeń. Można łatwo wykazać, że każde dopuszczalne rozwiązanie $\boldsymbol{\sigma}$ opisuje dokładnie jeden dopuszczalny harmonogram dosunięty w lewo, a każdy harmonogram dosunięty w lewo jest opisany przez dokładnie jedną $\boldsymbol{\sigma}$. Taki harmonogram można zbudować w analogiczny sposób, jak jest budowany harmonogram w oparciu o kolejność wykonywania operacji w problemie gniazdowym [95], ponieważ ograniczenia (5.19)-(5.24) można również przedstawić jako rzadki, acykliczny oraz ważony graf. Zatem dla dowolnego $\boldsymbol{\sigma} \in \Sigma_{\text{feas}}$, odpowiedni harmonogram można zbudować w czasie $O(nm)$. Przez $C_{\max}(\boldsymbol{\sigma})$ oznaczono długość uszeregowania dla kolejności $\boldsymbol{\sigma}$. Ponieważ każdy optymalny harmonogram można przekształcić w harmonogram dosunięty w lewo bez wpływu na jego długość, rozważany problem można przekształcić do problemu znalezienia optymalnej kolejności wykonywania przezbrojeń $\boldsymbol{\sigma}^*$, takiej, że

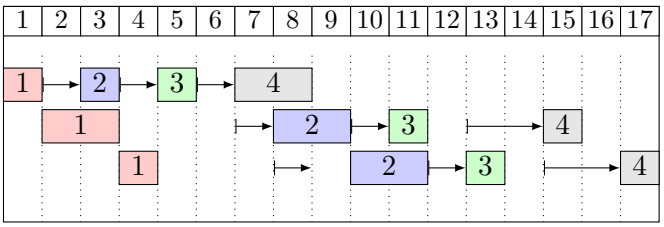
$$C_{\max}(\boldsymbol{\sigma}^*) = \min_{\boldsymbol{\sigma} \in \Sigma_{\text{feas}}} C_{\max}(\boldsymbol{\sigma}). \tag{5.29}$$

W dalszej części, o ile nie napisano inaczej, będzie używana reprezentacja kolejności wykonywania przezbrojeń przy użyciu $\boldsymbol{\sigma} \in \Sigma$.

Przykład 5.3 Rozważono instancję o rozmiarze $n = 4$, $m = 3$ i czasami wykonywania przezbrojeń oraz operacji z Tabeli 5.4. Kolejność wykonywania przezbrojeń $\boldsymbol{\sigma} = (1, 1, 1, 2, 3, 2, 3, 2, 3)$ jest dopuszczalna dla tej instancji. Dosunięty w lewo harmonogram dla rozwiązania $\boldsymbol{\sigma}$ z wartością $C_{\max} = 17$ pokazano na schemacie Gantta na Rysunku 5.3. Przedstawione

Tablica 5.4: Instancja problemu z Przykładu 5.3 dla $n = 4$ oraz $m = 3$

i	p_i^1	p_i^2	p_i^3	s_i^1	s_i^2	s_i^3
1	1	2	1	1	1	1
2	1	2	2	1	1	1
3	1	1	1	1	2	2
4	2	1	1	-	-	-



Rysunek 5.3: Dosunięty w lewo harmonogram dla instancji z Przykładu 5.3 oraz $\boldsymbol{\sigma} = (1, 1, 1, 2, 3, 2, 3, 2, 3)$

$\boldsymbol{\sigma}$ nie jest rozwiązaniem optymalnym. Można na przykład pierwsze przezbrojenie na maszynie 2 wykonać wcześniej (podczas wykonywania trzeciej operacji na maszynie pierwszej).

W rezultacie otrzymano problem dwupoziomowy. Na wyższym poziomie należy ustalić kolejność π wykonywania zadań, a na dolnym kolejność σ. Zatem badany problem jest znacznie trudniejszy niż standardowy wariant FSSP z przezbrojeniami zależnymi od kolejności. Oddzielne zajmowanie się poziomami problemów okazało się skutecznym podejściem dla różnych wariantów FSSP, takich jak np. CFSSP z dwoma maszynami [14,15]. W dalszej części tego rozdziału skupiono się na niższym poziomie: próbując zminimalizować $C_{\max}(\pi, \sigma)$ dla ustalonego π.

5.2.2 Własności problemu

Poniżej zostaną przedstawione oraz omówione własności i twierdzenia dla problemu szeregowania rozłącznych przezbrojeń. Dla dwóch maszyn zidentyfikowano rozmiar przestrzeni wszystkich rozwiązań oraz rozwiązań dopuszczalnych, a następnie pokazano ich zależność z liczbami Catalana. Dla dowolnej liczby maszyn zidentyfikowano własność eliminacyjną bazującą na blokach oraz zaproponowano procedurę udoskonalenia pojedynczego dopuszczalnego rozwiązania.

Można łatwo zaobserwować, że dla dowolnego $\pi \in \Pi$ zawsze można zbudować co najmniej jeden dopuszczalny harmonogram, jednak w przypadku kolejności przezbrojeń nie ma takiej gwarancji, gdzie dla pewnych kombinacji kolejności wykonywania przezbrojeń mogą one być niedopuszczalne. Przez σ_k zdefiniowano częściową kolejność wykonywania przezbrojeń, która jest sekwencją utworzoną z k pierwszych elementów pewniej kolejności przezbrojeń σ:

$$\forall i \in \{1, 2, \ldots, k\} \quad \sigma(i) = \sigma_k(i). \tag{5.30}$$

Ponadto częściowa kolejność wykonywania przezbrojeń σ_k będzie nazywana dopuszczalną jeśli istnieje dopuszczalny harmonogram dla pierwszych k przezbrojeń w ${}^{\sigma}S$ opisanych przez σ_k.

Następnie niech $\mathcal{D}(\sigma_k, a)$ dla dowolnego $k \in \{1, 2, \ldots, (n-1)m\}$ oraz $a \in \mathcal{M}$ będzie zdefiniowane jako:

$$\mathcal{D}(\sigma_k, a) = \big|\{i \in \{1, 2, \ldots, k\} : \sigma(i) = a\}\big|, \tag{5.31}$$

gdzie $\mathcal{D}(\sigma_k, i)$ oznacza liczbę przezbrojeń wykonanych na maszynie a w σ_k. Korzystając z powyższej notacji oraz ograniczeń (5.18)–(5.24) można zdefiniować następującą własność o dopuszczalności σ.

Własność 5.2 *Dowolna kolejność wykonywania przezbrojeń σ jest dopuszczalna wtedy i tylko wtedy, gdy dla wszystkich $k \in \{1, 2, \ldots, (n-1)m - 1\}$ oraz dla wszystkich $a \in \{1, 2, \ldots, \sigma(k+1) - 1\}$,*

$$\mathcal{D}(\sigma_k, a) \geqslant \mathcal{D}\big(\sigma_k, \sigma(k+1)\big). \tag{5.32}$$

□

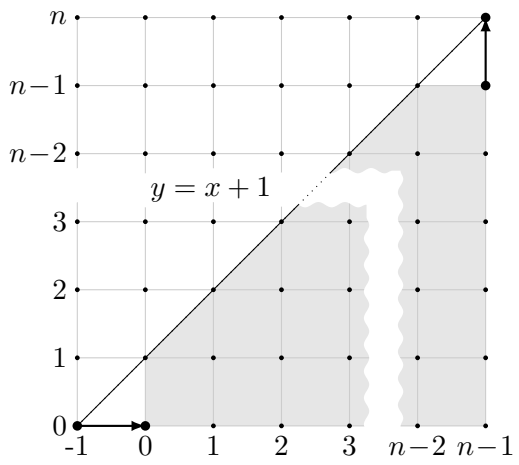
Wiedząc dokładnie jakie kolejności wykonywania przezbrojeń są dopuszczalne, można wyznaczyć ich liczbę.

Lemat 5.1 *Dla danej instancji problemu z $m = 2$ maszynami oraz $n > 1$ zadaniami, liczba dopuszczalnych kolejności wykonywania przezbrojeń $|\Sigma_{\text{feas}}|$ jest dana przez n-tą liczbę Catalana:*

$$|\Sigma_{\text{feas}}| = \mathcal{C}_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}. \tag{5.33}$$

□

Dowód. Należy rozpocząć od przekształcenia problemu znalezienia liczby dopuszczalnych kolejności wykonywania przezbrojeń do problemu zliczenia unikatowych ścieżek kratowych (*lattice paths*) spełniających specyficzne ograniczenia. Formalnie ścieżka kratowa $L = (L_0, L_1, \ldots, L_l)$ jest sekwencją punktów $\mathbb{Z}^d$, gdzie każdy krok $\overrightarrow{L_i L_{i+1}},\ i \in \{0, \ldots, l-1\}$ jest pobierany



Rysunek 5.4: Zależność między liczbą dopuszczalnych σ oraz liczbami Catalana

z predefiniowanego zbioru. W tym przypadku dla $d = 2$ zbiór kroków jest zdefiniowany jako $\{(1,0),(0,1)\}$. Jeśli powiążemy wykonanie przezbrojenia na pierwszej maszynie z krokiem $(1,0)$ oraz na drugiej maszynie z krokiem $(0,1)$, wtedy współrzędne punktu w $\mathbb{Z}^2$ mogą reprezentować liczbę wykonany przezbrojeń. Ostatecznie na każdej maszynie musi być wykonanych $n-1$ przezbrojeń, więc aby znaleźć rozmiar przestrzeni rozwiązań, należy obliczyć liczbę ścieżek z $(0,0)$ do $(n-1,n-1)$. Jednak ze względu na ograniczenia technologiczne, w dowolnym momencie liczba przezbrojeń na drugiej maszynie nie może być większa niż liczba przezbrojeń na pierwszej maszynie plus jeden. Fakt ten można opisać w dziedzinie ścieżek kratowych jako ograniczenie punktów leżących słabo (tj. spełniający nierówność $y \leqslant x+1$) pod linią $y = x + 1$.

Patrząc na Rysunek 5.4 można łatwo zaobserwować, że liczba ścieżek z punktu $(0,0)$ do $(n-1,n-1)$ pozostających w szarym obszarze jest równa liczbie ścieżek od $(-1,0)$ do $(n,n-1)$ pozostających słabo pod $y = x + 1$. Jest to spowodowane faktem, że jedynym możliwym krokiem dla punktu $(-1,0)$ jest ruch „w prawo" $(1,0)$ oraz dla punktu $(n-1,n-1)$ jest ruch „w górę" $(0,1)$. Przesuwając układ współrzędnych o wektor $(1,0)$, otrzymamy równoważny problem znalezienia liczby ścieżek z punktu $(0,0)$ do (n,n), pozostając słabo pod linią $y = x$. Ten problem ma znane rozwiązanie jako n-ta liczba Catalana C_n [125, 126]. ■

Twierdzenie 5.3 *Niech* (π, σ) *będzie dopuszczalnym rozwiązaniem takim, że dla pewnych* $1 \leqslant k < (n-1)2$, $1 < a \leqslant 2$, $1 \leqslant b < a$ *and* $i \in \mathcal{J}$

$$\tau_\sigma(k) = \tau_i^a, \tag{5.34}$$

$$\tau_\sigma(k+1) = \tau_i^b. \tag{5.35}$$

Następnie rozwiązanie (π, σ'),

$$\tau_{\sigma'}(j) = \begin{cases} \tau_i^b & \text{for } j = k, \\ \tau_i^a & \text{for } j = k+1, \\ \tau_\sigma(j) & \text{w przeciwnym wypadku,} \end{cases} \tag{5.36}$$

jest dopuszczalne oraz

$$C_{\max}(\pi, \sigma') \leqslant C_{\max}(\pi, \sigma). \tag{5.37}$$

□

Dowód. Twierdzenie można udowodnić, analizując czasy zakończenia wykonywania operacji po przezbrojeniach τ_{i+1}^a oraz τ_{i+1}^b w lewo dosuniętym harmonogramie. Przekształcając σ w σ', czasy zakończenia wykonywania zarówno przezbrojeń jak i operacji nigdy w takim przypadku nie zostaną zwiększone, zatem długość uszeregowania również nie zostanie wydłużona.■

Twierdzenie 5.3 można zawsze zastosować bez pogorszenia wartości $C_{\max}$, ponieważ istnieje co najmniej jedno rozwiązanie optymalne dla którego równania (5.34)–(5.35) nie są prawdziwe, tzn. przed dowolnym i-tym zadaniem najpierw jest wykonywane przezbrojenie na pierwszej maszynie. Zatem wszystkie inne rozwiązania można „wyeliminować". W celu oszacowania części przestrzeni rozwiązań jaką można wyeliminować dzięki Twierdzeniu 5.3, należy najpierw policzyć liczbę wyeliminowanych kolejności wykonywania przezbrojeń.

Lemat 5.2 *Dla ustalanego* π *liczba dopuszczalnych kolejności wykonywania przezbrojeń takich, że* (5.34)–(5.35) *nie jest prawdą, podana jest* $n-1$ *liczbą Catalana*

$$C_{n-1} = \frac{1}{n}\binom{2n-2}{n-1} = \frac{(2n-2)!}{n!(n-1)!}. \tag{5.38}$$

□

Dowód Lematu 5.2 jest analogiczny do dowodu Lematu 5.1, w którym były brane pod uwagę jedynie ograniczenia technologicznie. Z dowodu Twierdzenia 5.3 wynika, że niekorzystne jest, aby w dowolnym momencie

liczba przezbrojeń na drugiej maszynie była większa o jeden niż na maszynie pierwszej. Fakt ten w dziedzinie ścieżek kratowych można opisać jako problem znalezienia ścieżek leżących słabo pod linią $y = x$ z punktu $(-1, 0)$ do $(n, n-1)$. Ponownie przesuwając układ współrzędnych o wektor $(1, 0)$ otrzymany równoważny problem znalezienia liczby ścieżek z punktu $(0, 0)$ do (n, n), pozostających słabo pod linią $y = x - 1$, którego rozwiązaniem jest $n-1$ liczba Catalana. Używając Lematu 5.1 oraz 5.2 można oszacować wpływ Twierdzenia 5.3.

Twierdzenie 5.4 *Dla ustalonego π oraz $m = 2$, Twierdzenie 5.3 eliminuje od 50% do 75% dopuszczanych rozwiązań z przestrzeni rozwiązań.* □

Dowód. Z Lematu 5.1 oraz 5.2, Stosunek $\mathcal{F}(n)$ wyeliminowanych rozwiązań do całej przestrzeni rozwiązań dopuszczalnych jest równy:

$$\begin{aligned}\mathcal{F}(n) = \frac{C_{n-1}}{C_n} &= \frac{(2n-2)!}{n!(n-1)!} \cdot \frac{(n+1)!n!}{(2n)!} = \\ &= \frac{(2n)!}{2n(2n-1)} \cdot \frac{n}{n!} \cdot \frac{n!(n+1)}{(2n)!} = \\ &= \frac{n+1}{4n-2}. \end{aligned} \tag{5.39}$$

Wiedząc, że $\mathcal{F}(n)$ jest silnie monotoniczne dla $n > 1$, otrzymujemy:

$$\max_{n>1} \mathcal{F}(n) = \lim_{n\to\infty} \mathcal{F}(n) = \lim_{n\to\infty} \frac{n+1}{4n-2} = 0.25, \tag{5.40}$$

$$\min_{n>1} \mathcal{F}(n) = \mathcal{F}(2) = 3/6 = 0.5. \tag{5.41}$$

■

Blok przezbrojeń w $\boldsymbol{\sigma}$ (w skrócie *blok*) nazywana będzie sekwencja przezbrojeń z $\boldsymbol{\sigma}$ wykonywanych jedno po drugim tak, że wszystkie przezbrojenia są wykonywane po operacjach tego samego zadania. Długość bloku definiowana jest jako liczba przezbrojeń zawartych w bloku.

Przykład 5.4 Rozważmy następujące rozwiązanie:

$$\boldsymbol{\sigma} = (\underbrace{3,2,1,}\ \underbrace{1,}\ \underbrace{4,}\ \underbrace{2,4,}\ \underbrace{1,}\ \underbrace{3,}\ \underbrace{3,2,4}).$$

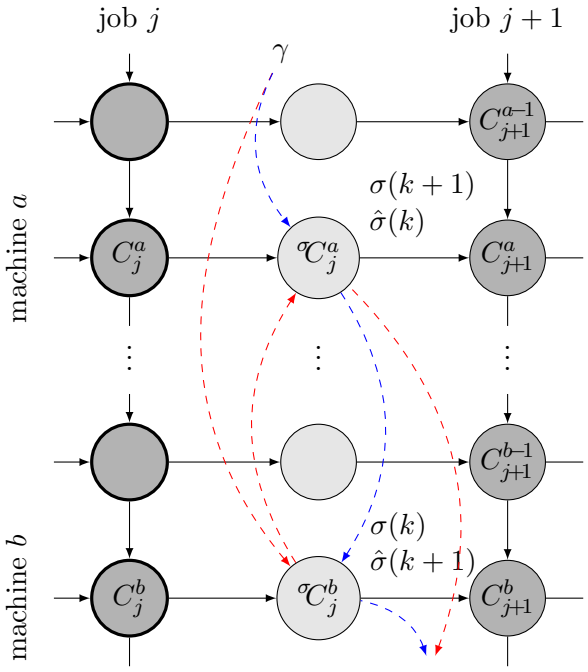
Powyższe rozwiązanie zawiera 7 bloków. Bloki oznaczono klamrami. □

Twierdzenie 5.5 *Niech $\boldsymbol{\sigma} \in \Sigma_{feas}$ będzie rozwiązaniem dopuszczalnym z dwoma następującymi po sobie przezbrojeniami $\sigma(k) = b$, $\sigma(k+1) = a$,*

w bloku po zadaniu $j \in \mathcal{J} \setminus \{n\}$, takimi, że $a < b$. Niech $\hat{\boldsymbol{\sigma}}$ oznacza rozwiązanie, gdzie jest odwrócona kolejność tych dwóch przezbrojeń:

$$\forall i \in \mathcal{S} \quad \hat{\sigma}_i = \begin{cases} a & dla\ i = k, \\ b & dla\ i = k+1, \\ \sigma_i & w.p.p, \end{cases} \tag{5.42}$$

$\hat{\boldsymbol{\sigma}}$ jest rozwiązaniem dopuszczalnym oraz $C_{\max}(\hat{\boldsymbol{\sigma}}) \leqslant C_{\max}(\boldsymbol{\sigma})$. □



Rysunek 5.5: Ilustracja do dowodu Twierdzenia 5.5. Pełne łuki przedstawiają ograniczenia technologiczne i maszynowe. Łuki przerywaną linią reprezentują kolejność wykonywania przezbrojeń (czerwone dla $\boldsymbol{\sigma}$ oraz niebieskie dla $\hat{\boldsymbol{\sigma}}$). Ciemnoszare wierzchołki reprezentują operacje, a jasnoszare wierzchołki przedstawiają przezbrojenia

Dowód. Dowód opiera się na analizie dosuniętych w lewo harmonogramów zbudowanych zarówno dla $\boldsymbol{\sigma}$ oraz $\hat{\boldsymbol{\sigma}}$. Elementy harmonogramu zbudowanego dla $\hat{\boldsymbol{\sigma}}$ są oznaczone daszkami ($^{\sigma}\hat{C}$, $^{\pi}\hat{C}$, itd.). Ponadto γ będzie oznaczać czas zakończenia wykonywania przezbrojenia $\sigma(k-1)$, na który zamiana nie ma wpływu. Notację przedstawiono na Rysunku 5.5.

Najpierw należy zidentyfikować elementy harmonogramu, na które nie ma wpływu zmiana kolejności wykonywania k-tego oraz $k+1$-go przezbrojeń. W oryginalnym rozwiązaniu k-te przezbrojenie jest wykonywane na maszynie b, to zanim przezbrojenie może zacząć być wykonywane, to operacja b zadania j musiała być już wykonana wcześniej (oznaczono poprzez pogrubienie na Rysunku 5.5). Dlatego wartości C_j^a oraz C_j^b pozostają stałe dla obu kolejności wykonywania przezbrojeń. W rezultacie czas zakończenia wykonywania przezbrojeń w $\hat{\boldsymbol{\sigma}}$ wykonywanych po zadaniu j na maszynach $\{1, 2, \ldots, b\} \setminus \{a, b\}$, może być zmieniony jedynie przez czas zakończenia wykonywania $k+1$-tego przezbrojenia. Zmianę można wyrazić jako ${}^{\sigma}C_j^a - {}^{\sigma}\hat{C}_j^b$, gdzie

$$\begin{aligned} {}^{\sigma}C_j^a &= s_j^a + \max\{s_j^b + \max\{\gamma,\ C_j^b\},\ C_j^a\} \\ &= s_j^a + s_j^b + \max\{\gamma,\ C_j^b\}, \qquad (5.43) \\ {}^{\sigma}\hat{C}_j^b &= s_j^b + \max\{s_j^a + \gamma,\ s_j^a + C_j^a,\ C_j^b\} \\ &= s_j^b + s_j^a + \max\{\gamma,\ C_j^a,\ C_j^b - s_j^a\}. \qquad (5.44) \end{aligned}$$

Zatem,

$${}^{\sigma}C_j^a - {}^{\sigma}\hat{C}_j^b = \max\{\gamma,\ C_j^b\} - \max\{\gamma,\ C_j^a,\ C_j^b - s_j^a\}. \qquad (5.45)$$

Dlatego, że $C_j^a < C_j^b$, mamy ${}^{\sigma}C_j^a - {}^{\sigma}\hat{C}_j^b \geqslant 0$ oraz czas zakończenia wykonywania $k+1$-tego przezbrojenia nie mógł się zwiększyć w $\hat{\boldsymbol{\sigma}}$. W rezultacie, operacje z zadania $j+1$ wykonywane na maszynie a oraz b nie zostaną opóźnione, więc ${}^{\pi}\hat{C}_{j+1}^{a-1} \leqslant C_{j+1}^{a-1}$ oraz ${}^{\pi}\hat{C}_{j+1}^{b-1} \leqslant {}^{\sigma}C_{j+1}^{b-1}$. W związku z tym, że operacje z zadania $j+1$ mogą zostać wykonane tylko wcześniej na maszynach a oraz b, zatem długość uszeregowania również może tylko zmaleć.

Przejdźmy teraz do omówienia czasu zakończenia wykonywania zadania $j+1$ na maszynie a,

$$\begin{aligned} C_{j+1}^a - {}^{\pi}\hat{C}_{j+1}^a = \max\{&s_j^a + s_j^b + \max\{\gamma, C_j^b\}, C_{j+1}^{a-1}\} \\ &- \max\{s_j^a + \max\{\gamma, C_j^a\}, {}^{\pi}\hat{C}_{j+1}^{a-1}\}. \qquad (5.46) \end{aligned}$$

Wiedząc, że $C_{j+1}^{a-1} \geqslant {}^{\pi}\hat{C}_{j+1}^{a-1}$, wystarczy obliczyć tyko

$$\begin{aligned} s_j^a + s_j^b + \max\{\gamma,\ C_j^b\} - (s_j^a + \max\{\gamma,\ C_j^a\}) \\ = s_j^b + \max\{\gamma,\ C_j^b\} - \max\{\gamma,\ C_j^a\}. \qquad (5.47) \end{aligned}$$

Dlatego, że $C_j^b > C_j^a$ oraz wartość wyrażenia (5.47) jest nieujemna, to $C_{j+1}^a - {}^{\pi}\hat{C}_{j+1}^a \geqslant 0$ oraz $C_{j+1}^a \geqslant {}^{\pi}\hat{C}_{j+1}^a$.

Teraz omówmy zmianę czasu zakończenia wykonywania zadania $j+1$ na maszynie b. W związku z tym, że ${}^{\sigma}C_j^b < {}^{\sigma}C_j^a < C_{j+1}^a < C_{j+1}^{b-1}$, otrzymujemy

$$C_{j+1}^b = p_{j+1}^b + \max\{\, {}^{\sigma}C_j^b,\ C_{j+1}^{b-1}\} = p_{j+1}^b + C_{j+1}^{b-1}, \tag{5.48}$$

$${}^{\pi}\hat{C}_{j+1}^b = p_{j+1}^b + \max\{\, {}^{\hat{\sigma}}\hat{C}_j^b,\ {}^{\pi}\hat{C}_{j+1}^{b-1}\}. \tag{5.49}$$

Dlatego, że $C_{j+1}^{b-1} \geqslant {}^{\pi}\hat{C}_{j+1}^{b-1}$ oraz $C_{j+1}^{b-1} > {}^{\sigma}C_j^a \geqslant {}^{\hat{\sigma}}\hat{C}_j^b$, to również $C_{j+1}^b \geqslant {}^{\pi}\hat{C}_{j+1}^b$.

Podsumowując, udowodniono, że w dosuniętym w lewo harmonogramie zbudowanym dla $\hat{\boldsymbol{\sigma}}$, poprzez zamianę kolejności wykonywania przezbrojeń na pozycjach k oraz $k+1$ w rozwiązaniu $\boldsymbol{\sigma}$, czas zakończenia wykonywania zadań $1, 2, \ldots, j+1$ na maszynach $1, 2, \ldots, b$ może zostać wyłącznie zmniejszony. To samo dotyczy czasu zakończenia wykonywania $k+1$-tego przezbrojenia. Zatem wszystkie późniejsze zadania oraz przezbrojenia można wyłącznie zacząć wykonywać wcześniej, więc w rezultacie $C_{\max}(\hat{\boldsymbol{\sigma}}) \leqslant C_{\max}(\boldsymbol{\sigma})$. ■

Twierdzenie 5.5 pozwala wykryć potencjalnie nieoptymalne rozwiązania. Dlatego, że opisane przekształcenie $\boldsymbol{\sigma} \to \boldsymbol{\sigma}'$ nie może prowadzić do zwiększania długości uszeregowania, więc dowolna $\boldsymbol{\sigma} \in \Sigma_{\text{feas}}$, która spełnia warunki Twierdzenia 5.5 może zostać bezpiecznie *wyeliminowana* z przestrzeni rozwiązań, bez ryzyka usunięcia optymalnych rozwiązań.

Procedura udoskonalenia

Na podstawie wniosków wynikających z Twierdzenia 5.5 można zmniejszyć przestrzeń rozwiązań, którą należy przejrzeć w celu znalezienia najlepszego rozwiązania. Jednak na tej samej podstawie można również skonstruować metodę potencjalnej poprawy rozwiązania przy użyciu prostych zamian w kolejności wykonywania przezbrojeń tak, aby ta kolejność spełniała własności eliminacyjne. W jednym rozwiązaniu może istnieć więcej niż jedna para, którą można w ten sposób zamienić. Ponadto, po wykonaniu zamiany, może powstać nowa para wymagająca zamiany, jak pokazano w Przykładzie 5.5.

Przykład 5.5 Rozważono problem o rozmiarze $m = 3$ oraz $n = 2$. Rozwiązanie $\boldsymbol{\sigma} = (3, 2, 1)$ jest dopuszczalne, jednak dwie pary przezbrojeń: $(3, 2)$ oraz $(2, 1)$, spełniają założenia własności eliminacyjnej. Stosując ruch zamiany dla pary $(3, 2)$, otrzymujemy rozwiązanie $\boldsymbol{\sigma}' = (2, 3, 1)$ – zawierające teraz jedną, nową parę $(3, 1)$, która również spełnia własność eliminacyjną pokazaną w Twierdzeniu 5.5. Stosując kolejny ruch zamiany dla pary $(2, 1)$, otrzymujemy $\boldsymbol{\sigma}'' = (2, 1, 3)$. Rozwiązanie ponownie zawiera parę $\boldsymbol{\sigma}$,

$(2,1)$. Ostatecznie, ruch zamiany można zastosować po raz trzeci, aby uzyskać $\boldsymbol{\sigma}''' = (1,2,3)$. Kolejność wykonywania przezbrojeń $\boldsymbol{\sigma}'''$ nie spełnia już założenia z Twierdzenia 5.5 i nie może zostać wyeliminowana. Procedurę można podsumować w następujący sposób:

$$\underbrace{(3,\underbrace{2,1}) \xrightarrow[(3,2)]{\text{swap}} (2,\underbrace{3,1}) \xrightarrow[(3,1)]{\text{swap}} (\underbrace{2,1},3) \xrightarrow[(2,1)]{\text{swap}} (1,2,3)}_{\boldsymbol{\sigma}\quad\boldsymbol{\sigma}'\quad\boldsymbol{\sigma}''\quad\boldsymbol{\sigma}'''}. \qquad \square$$

Bazując na powyższej obserwacji oraz powyższym przykładzie, można zdefiniować `procedurę udoskonalenia`. Mając daną kolejność wykonywania przezbrojeń $\boldsymbol{\sigma}$, należy wykonywań w niej zamiany do moment, gdy Twierdzenie 5.5 nie będzie mogło być już zastosowane. W rezultacie rozwiązanie $\boldsymbol{\sigma'}$ będzie nazywane rozwiązaniem udoskonalanym oraz długość jego uszeregowania będzie mniejsza lub równa względem rozwiązania $\boldsymbol{\sigma}$:

$$C_{\max}(\boldsymbol{\sigma'}) \leqslant C_{\max}(\boldsymbol{\sigma}). \tag{5.50}$$

Oczywiście niektóre rozwiązania nie zawierają par przezbrojeń spełniających własność eliminacyjną, w takim przypadku nie zostanie wykonana ani jedna zamiana. W Przykładzie 5.5 rozwiązanie $(1,2,3)$ jest wynikiem udoskonalenia rozwiązania $(3,2,1)$.

Poniżej zostanie przedstawiony kompletna procedura udoskonalenia rozwiązania, w tym wykrycie par przezbrojeń do zamiany. Zostanie również wykazana złożoność obliczeniowa metody bazującej na idei Sortowania Bąbelkowego (*Bubble Sort*).

Twierdzenie 5.6 *Dla dowolnego dopuszczalnego rozwiązania σ, procedurę udoskonalenia można przeprowadzić w czasie $O(nm)$.* $\square$

Dowód. Należy zaobserwować, że zmiana kolejności wykonywania przezbrojeń wewnątrz jednego bloku nie zmienia całkowitej liczby bloków ani zawartości innych bloków. Dlatego procedurę udoskonalenia można zastosować do każdego bloku osobno. Rozważmy blok o długości l. Istnieje tylko jedna kolejność wykonywania przezbrojeń, dla której żadna para przezbrojeń wewnątrz bloku nie spełnia twierdzenia o eliminacji – przezbrojenia uporządkowane są w kolejności rosnącej względem maszyny na której są wykonywane. Blok może być posortowany przez co najwyżej $O(l^2)$ zamian-ruchów (Sortowanie Bąbelkowe [29, str. 40])), wybranych na podstawie twierdzenia o własności eliminacyjnej. W rezultacie pesymistyczna złożoność czasowa procedury wyniosłaby $O(nm^2)$. Jednak sortowanie można wykonać szybciej, w czasie $O(nm)$, stosując Algorytm 13, który sortuje wszystkie bloki jednocześnie. ■

Algorytm 13: Procedura udoskonalenia

Dane: $\boldsymbol{\sigma}$: rozwiązanie do udoskonalenia.
Wynik: $\boldsymbol{\sigma}'$: rozwiązanie udoskonalone.

```
1  function RefinementProcedure(σ):
       // Znalezienie zadania poprzedzającego dla każdego
          przezbrojenia.
2      lastJob ← 1;
3      for k = 1, 2, ..., (n − 1)m do
4          job[k] ← lastJob[σ(k)];
5          lastJob[σ_k] ← lastJob[σ(k)] + 1;
       // Znalezienie identyfikatora bloku dla każdego
          przezbrojenia.
6      currBlockId ← 1, c ← job[1];
7      for k = 1, 2, ..., m(n − 1) do
8          if job[k] ≠ currJob then
9              currBlockId ← currBlockId + 1;
10             currJob ← job[k];
11         blockId[currJob, σ(k)] ← currBlockId;
       // Konstruowanie bloków
12     for i = 1, 2, ..., n − 1 do
13         for a = 1, 2, ..., m do
14             push back a to block[blockId[i, a]];
       // Zbudowanie udoskonalonej kolejności wykonywania
          przezbrojeń.
15     σ' ← ();
16     for q = 1, 2, ..., |block| do
17         push back block[q] to σ';
18     return σ'
```

Procedura udoskonalania może być wykorzystana do potencjalnej poprawy jakości rozwiązań w ramach większego algorytmu rozwiązywania (podobnie do procedury uczenia się w algorytmach memetycznych [91]). Złożoność obliczeniowa $O(nm)$ jest równa złożoności wyznaczania wartości funkcji celu, stąd procedura udoskonalania może być często stosowana.

5.2.3 Algorytmy optymalizacyjne

W dalszej części pracy zostaną opisane cztery metody: (1) sformułowanie MILP rozważanego problemu, (2) zachłanna heurystyka dla dwóch maszyn, (3) zachłanna heurystyka dla wielu maszyn oraz (4) Przeszukiwanie z Zabronieniami dla dowolnej liczby maszyn. Pierwsza metoda, jako dokładna, zostanie użyta jako odniesienie dla pozostałych algorytmów. Wszystkie algorytmy mogą być również częścią dwupoziomowej metody, rozwiązując problem ze zmienną kolejnością wykonywania zadań. W tym wypadku przyjęto stałą kolejność wykonywania zdań $\pi = (1, 2, \ldots, n)$, gdyż znalezienie optymalnej kolejności wykonywania przezbrojeń dla pewnego π należy do klasy problemów NP-trudnych.

MILP

Proponowane sformułowanie MILP wykorzystuje porządek względny (podobnie jak w [4,8]) do zakodowania kolejności wykonywania przezbrojeń σ:

$$x_{i,j}^{a,b} = \begin{cases} 1 & \text{kiedy przezbrojenie } s_{i,i+1}^{a} \text{ jest przed } s_{j,j+1}^{b} \text{ w } \sigma, \\ 0 & \text{w przeciwnym wypadku,} \end{cases} \tag{5.51}$$

gdzie $a, b \in \mathcal{M}$ oraz $i, j \in \mathcal{J} \setminus \{n\}$. Model definiuje się następująco:

$$\min_{C_{\max}, x, S, {}^{\sigma}S} C_{\max}, \tag{5.52}$$

s.t.

$$S_i^a + p_i^a \leqslant S_i^{a+1} \quad \forall a \in \mathcal{M} \setminus \{m\},\ i \in \mathcal{J}, \tag{5.53}$$

$${}^{\sigma}S_{i-1}^a + s_{i-1,i}^a \leqslant S_i^a \quad \forall a \in \mathcal{M},\ i \in \mathcal{J} \setminus \{1\}, \tag{5.54}$$

$$S_i^a + p_i^a \leqslant {}^{\sigma}S_i^a \quad \forall a \in \mathcal{M},\ i \in \mathcal{J} \setminus \{n\}, \tag{5.55}$$

$${}^{\sigma}S_i^a + S_{i,i+1}^a \leqslant {}^{\sigma}S_j^b + \mathtt{N} \cdot (1 - x_{i,j}^{a,b}) \quad \forall a, b \in \mathcal{M}, a \neq b;\ i, j \in \mathcal{J} \setminus \{n\}, \tag{5.56}$$

$$x_{i,j}^{a,b} + x_{j,i}^{b,a} = 1 \quad \forall a, b \in \mathcal{M}, a \neq b;\ i, j \in \mathcal{J} \setminus \{n\}, \tag{5.57}$$

$$S_i^m + p_i^m \leqslant C_{\max} \quad \forall i \in \mathcal{J}, \tag{5.58}$$

gdzie

$$C_{\max} \in \mathbb{R}^+ \cup \{0\}, \tag{5.59}$$

$$S_i^a \in \mathbb{R}^+ \cup \{0\} \quad \forall a \in \mathcal{M},\ i \in \mathcal{J}, \tag{5.60}$$

$${}^{\sigma}S_i^a \in \mathbb{R}^+ \cup \{0\} \quad \forall a \in \mathcal{M},\ i \in \mathcal{J} \setminus \{n\}, \tag{5.61}$$

$$x_{i,j}^{a,b} \in \{0,1\} \qquad \forall a,b \in \mathcal{M},\ i,j \in \mathcal{J} \setminus \{n\}, \tag{5.62}$$

to zmienne modelu (${}^{\sigma}S_i^a$ jest momentem rozpoczęcia wykonywania przezbrojenia $s_{i,i+1}^a$) oraz $\mathtt{N}$ to duża liczba ($\mathtt{N} \gg C_{\max}$). Rozdział jest poświęcony problemowi o stałej licznie maszyn $m = 2$, ale powyższe sformułowanie MILP może być użyte dla dowolnej liczby maszyn $m > 1$.

Biorąc pod uwagę stałą kolejność wykonywania zadań, można dodatkowo przyjąć pewne wartości początkowe $\boldsymbol{x}$ dla $a \in \mathcal{M}$ oraz $i,j \in \mathcal{J} \setminus \{n\},\ i \neq j$:

$$x_{i,j}^{a,a} = \begin{cases} 0 & \text{for } i \in \{1,\ldots,j-1\}, \\ 1 & \text{for } i \in \{j,\ldots,n-1\}. \end{cases} \tag{5.63}$$

Algorytm zachłanny dla dwóch maszyn

Algorytm zachłanny pokazany w Algorytmie 14 opiera się na idei z Twierdzenia 5.3, tzn. w każdej chwili może być więcej wykonanych przezbrojeń na pierwszej maszynie niż na drugiej, ale nigdy odwrotnie. Dlatego pierwsze przezbrojenie zawsze jest wykonywane na maszynie pierwszej. Następnie dla każdego $k = 1,2,\ldots,2(n-1)$ przezbrojenia w kolejności σ testowana jest każda z dwóch wartość dla $\sigma(k)$, analogicznie jak w algorytmie NEH [40]. Jeśli na obu maszynach jest w danym momencie przydzielone tyle samo przezbrojeń lub oszacowanie ma taką samą wartość dla obu maszyn, to wtedy zawsze jest wybierana maszyna pierwsza. Złożoność obliczeniowa algorytmu zależy liniowo od liczby przezbrojeń $O(|\mathcal{S}|)$.

Algorytm zachłanny dla wielu maszyn

Algorytm zachłanny dla wielu maszyn, podobnie jak algorytm dla $m = 2$, bazują ta tej samej idei, analogicznie do algorytmu NEH [94] dla PFSSP. Jednak w tym przypadku wynikiem działania algorytmu jest kolejność wykonywania przezbrojeń $\boldsymbol{\sigma}$ dla ustalonej kolejności wykonywania zadań. Metoda polega na stopniowym budowaniu rozwiązania cząstkowego. Najpierw należy ustalić czasy zakończenia wykonywania wszystkich operacji pierwszego zadania, ponieważ nie wymagają one przezbrojeń. Następnie trzeba wykonać pierwsze przezbrojenie na pierwszej maszynie ($\sigma(1) = 1$) oraz operację po niej, jednocześnie obliczając czasy ich ukończenia. Następnie

Algorytm 14: Algorytm zachłanny dla dwóch maszyn

Dane: π: kolejność wykonywania zadań
Wynik: σ: kolejność wykonywania przezbrojeń

function `Greedy(`π`):`
- $\sigma(1) \leftarrow 1$;
- $C^1_1 \leftarrow p^1_1$;
- ${}^{\sigma}C^1_1 \leftarrow C^1_1 + s^1_{1,2}$;
- $C^1_2 \leftarrow {}^{\sigma}C^1_1 + p^1_1$;
- **for** $k = 2, 3, \ldots, 2(n-1)$ **do**
  - $\mathsf{st}(1) \leftarrow \mathcal{D}(\sigma_{k-1}, 1)$;
  - $\mathsf{st}(2) \leftarrow \mathcal{D}(\sigma_{k-1}, 2)$;
  - **if** $\mathsf{st}(1) < n-1 \wedge (\mathsf{st}(1) = \mathsf{st}(2) \vee C^1_{\mathsf{st}(1)} + s^1_{\mathsf{st}(1)} \leqslant C^2_{\mathsf{st}(2)} + s^2_{\mathsf{st}(2)})$ **then**
    - $\sigma(k) \leftarrow 1$;
    - $\mathsf{st}(1) \leftarrow \mathcal{D}(\sigma_k, 1)$;
    - ${}^{\sigma}C^1_{\mathsf{st}(1)} \leftarrow \max\{C^1_{\mathsf{st}(1)-1}, {}^{\sigma}C^2_{\mathsf{st}(2)}\} + s^1_{\mathsf{st}(1)-1}$;
    - $C^1_{\mathsf{st}(1)} \leftarrow {}^{\sigma}C^1_{\mathsf{st}(1)} + p^1_{\mathsf{st}(1)}$;
  - **else**
    - $\sigma(k) \leftarrow 2$;
    - $\mathsf{st}(2) \leftarrow \mathcal{D}(\sigma_k, 2)$;
    - ${}^{\sigma}C^2_{\mathsf{st}(2)} \leftarrow \max\{C^2_{\mathsf{st}(2)-1}, {}^{\sigma}C^1_{\mathsf{st}(1)}\} + s^2_{\mathsf{st}(2)-1}$;
    - $C^2_{\mathsf{st}(2)} \leftarrow {}^{\sigma}C^2_{\mathsf{st}(2)} + p^2_{\mathsf{st}(2)}$;
- **return** σ;

przystępujemy do wstawiania pozostałych elementów do σ w następujący sposób.

Za każdym razem, gdy należy ustalić wartość $\sigma(k)$, trzeba wziąć pod uwagę wszystkie wartości $a \in \mathcal{M}$, których wstawienie jest dopuszczalne (biorąc pod uwagę częściową kolejność wykonywania przezbrojeń σ_{k-1}). Dla każdej takiej wartości a kandydata obliczany jest czas ukończenia wstawianego przezbrojenia. Aby to zrobić, przechowywany jest czas zakończenia ostatniego wykonanego przezbrojenia w σ_{k-1}. Dla każdego obliczonego czasu zakończenia wykonania przezbrojenia należy również obliczyć powiązany z tym przezbrojeniem czas zakończenia wykonywania operacji po wstawionym przezbrojeniu. Następnie jako $\sigma(k)$ wybierana jest taka wartość a, dla której obliczony czas zakończenia operacji był najmniejszy. Złożoność

obliczeniowa algorytmu zależy od liczby przezbrojeń oraz maszyn, wynosi dokładnie $O(m|\mathcal{S}|) = O(nm^2)$.

Algorytm 15: Algorytm zachłanny dla wielu maszyn

Wynik: $\boldsymbol{\sigma}$: kolejność wykonywania przezbrojeń.

function `Greedy()`**:**
- Initialize arrays op and st of size m with 0;
- $C_1^1 \leftarrow p_1^1$;
- $\mathsf{op}_1 \leftarrow 1$;
- **for** $i = 2, 3, \ldots, m$ **do**
  - $C_1^i \leftarrow C_1^{i-1} + p_1^i$;
  - $\mathsf{op}_i \leftarrow 1$;
- $\sigma(1) \leftarrow 1$;
- ${}^{\sigma}C_1^1 \leftarrow C_1^1 + s_1^1$;
- $\mathsf{last} \leftarrow {}^{\sigma}C_1^1$;
- $\mathsf{st}_1 \leftarrow 1$;
- **for** $k = 2, 3, \ldots, |\mathcal{S}|$ **do**
  - **for** $i = 1, 2, \ldots, m$ **do**
    - **if** $\mathsf{op}_i < n \wedge \mathsf{op}_i < \mathsf{op}_{i-1}$ **then**
      - $\mathsf{l}_i \leftarrow \max\{C^i_{\mathsf{op}_i}, last\} + s^i_{\mathsf{st}_i} + p^i_{\mathsf{op}_i}$;
  - $i^* \leftarrow \arg\min_{i \in \mathcal{M}} \mathsf{l}_i$;
  - $C^{i^*}_{\mathsf{op}_{i^*}} \leftarrow p^{i^*}_{\mathsf{op}_{i^*}}$;
  - $\mathsf{op}_{i^*} \leftarrow \mathsf{op}_{i^*} + 1$;
  - $\sigma(k) \leftarrow i^*$;
  - ${}^{\sigma}C^{i^*}_{\mathsf{op}_{i^*}} \leftarrow C^{i^*}_{\mathsf{op}_{i^*}} + s^{i^*}_{\mathsf{st}_{i^*}}$;
  - $\mathsf{last} \leftarrow {}^{\sigma}C^{i^*}_{\mathsf{st}_{i^*}}$;
  - $\mathsf{st}_{i^*} \leftarrow \mathsf{st}_{i^*} + 1$;
- **return** σ;

Przeszukiwanie z Zabronieniami

Metoda TS jest jest jedną z najbardziej znanych metod przeszukiwania lokalnego [46, 47], która w celu znalezienia lokalnego optimum zazwyczaj używa pamięci krótkoterminowej oraz czasami długoterminowej. Algorytm ze względu na deterministyczny charakter oraz wysoką efektywność w literaturze ma szerokie zastosowanie (szeregowanie zadań [14], problem pleca-

kowy [79], wybór modelu [87] czy na nawet replikacja danych w środowiskach chmurowych [36].

Algorytm 16: Przeszukiwanie z Zabronieniami

Dane: $\boldsymbol{\sigma}$: rozwiązanie początkowe.
Wynik: $\boldsymbol{\sigma}^*$: najlepsze znalezione rozwiązanie.

```
1  σ ← CreateInitialSolution();
2  function TabuSearch(σ):
3      σ* ← σ;
4      it ← 0, noImprove ← 0;
5      while TimeElapsed() < maxTime do
6          σ ← SearchNeihgborhood(σ);
7          UpdateTabuList();
8          if C_max(σ) < C_max(σ*) then
9              σ* ← σ;
10             noImprove ← 0;
11             AddJump();
12         else
13             noImprove ← noImprove + 1;
14         if noImprove > maxNoImprove then
15             BackJump();
16             noImprove ← 0;
17         it ← it + 1;
18     return σ*;
```

Ogólny schemat metody został przedstawiony w Algorytmie 16. Przyjęto otoczenie typu wstaw generowane przy pomocy ruchu $\mathtt{I}(\sigma, k, l)$, w którym wartość $\sigma(k)$ jest wycinana z rozwiązania i wstawiana na pozycji l, gdzie $k, l \in \mathcal{S}$. Wartości na pozycjach od $k+1$ do l zostaną zsunięte w lewo. Standardowo rozmiar sąsiedztwa zawiera $O(n^2m^2)$ rozwiązań. Jednak poprzez następujące reguły można ograniczyć liczbę sprawdzanych rozwiązań:

1. Zignorowanie wartości $l = k$, ponieważ nie wpływa na zmianą w $\boldsymbol{\sigma}$.
2. Wprowadzenie pojęcia szerokości sąsiedztwa $\mathtt{W}$, gdzie jest wykonywana tylko zadana liczba wstawień $|l - k| < \mathtt{W}$.
3. Odrzucone zostaną wstawienia, które skutkują rozwiązaniami niedopuszczalnymi lub mogą być wyeliminowane na podstawie wniosków Twierdzenia 5.5. Ta operacja jest szybsza niż wyznaczanie wartości funkcji celu i może być potencjalnie wykonana w czasie stałym $O(1)$

przy założeniu, że znana jest ścieżka kratowa odpowiadająca rozwiązaniu początkowemu. Metodę pokazano w Algorytmie 17.

Algorytm 17: Ocena sąsiedztwa

Dane: $\boldsymbol{\sigma}$: rozwiązanie początkowe.
Wynik: Moves: Lista ruchów.

```
1  L ← CalculateLatticePath(σ);
2  obliczyć skumulowane maksima L;
3  for k = 2, 3, ..., |S|−1 do
4      prMach ← σ(k − 1);
5      prJob ← L[k−1][prMach];
6      for l = k+1, k+2, ..., min{k+W, |S|−1} do
           // Sprawdzenie dopuszczalności;
7          x ← L[b][σ(k)] − 1;
8          if x < 0 then
9              break
10         if max_{σ(k)⩽a⩽m} L[l][a] − x > 1 then
11             break
           // Sprawdzanie własności eliminacyjnej;
12         crJob ← { L[l][σ(l)] − 1   if σ(k) = σ(l)
                   { L[l][σ(l)]       otherwise ;
13         if crJob = prJob ∧ σ(l) < prMach then
14             break
15         if crJob = L[l][σ(k)] ∧ σ(l) > σ(k) then
16             break
17         if crJob = L[l+1][σ(b + 1)] ∧ σ(l) > σ(l + 1) then
18             break
19         prMach ← σ(l);
20         prJob ← crJob;
21         Moves.PushBack(I(σ, k, l));
22 return Moves;
```

Operację z punktu 3. można wykonać w czasie stałym dla każdego potencjalnego ruchu, zakładając, że ścieżka kratowa odpowiadająca $\boldsymbol{\sigma}$ została już obliczona ($O(nm^2)$). Zamiast rozważać każdy ruch osobno należy przetestować ruchu kolejno, jeden po drugim. Oznacza to, że dla ustalonej war-

tości $k \in \{1, 2, \ldots, (n-1)m-1\}$, testujemy $\mathtt{I}(\sigma, k, l)$ dla $l = k+1,\ k+2,\ \ldots$. Wiedząc, że jeśli jakikolwiek ruch $\mathtt{I}(\sigma, k, l)$ prowadzi do rozwiązania niedopuszczalnego (lub rozwiązania spełniającego własność eliminacyjną), to jest to również prawdziwe dla ruchów $\mathtt{I}(\sigma, k, c)$, $k < c \leqslant l$ (takie są niedopuszczalne lub wyeliminowane). Dlatego wystarczy przetestować tylko ostatnią zmianę w $\boldsymbol{\sigma}$, czyli zmianę wartości $\sigma(b)$, jeśli poprzednie zmiany były wcześniej testowane. Operacja w linii 10. może być wykonana w czasie stałym zakładając, że skumulowane maksimum zostało wstępnie obliczone dla każdego elementu ścieżki kratowej. Ponadto każde rozwiązanie może zostać polepszone poprzez procedurę udoskonalenia.

Algorytmy bazujące na przeszukiwaniu lokalnym mają tendencję do wchodzenia w cykl. W celu uniknięcia wejścia w cykl metoda TS korzysta z pamięci krótkotrwałej nazywanej listą tabu. Lista tabu określa które ruchy są zabronione. Rozwiązania utworzone przez zabronione ruchy są ignorowane, chyba że są prowadzą do rozwiązań lepszych niż $\boldsymbol{\sigma}^*$. Lista tabu została zaimplementowana jako macierz `tabuList` o rozmiarze $(nm)^2$. Jeśli podczas iteracji `it` wykonywany jest ruch $\mathtt{I}(\sigma, k, l)$, zapisujemy wartość `it + C` do $\mathtt{tabuList}[k][l]$. Ruch $\mathtt{I}(\sigma, k, l)$ jest uważany za zabroniony w iteracji `it` jeśli $\mathtt{it} < \mathtt{tabuList}[k][l]$, gdzie parametr `C` nazywany kadencją, określa dla ilu iteracji ruchy pozostają zabronione. Na podstawie wstępnych badań ustalano parametr $\mathtt{C} = \lfloor\sqrt{nm}\rfloor$.

Dodatkowo oprócz pamięci krótkoterminowej została zaimplementowana pamięć długoterminowa w formie skoków powrotnych (*Back Jumps*). Za każdym razem, kiedy jest poprawione $\boldsymbol{\sigma}^*$ należy do listy możliwych skoków dodać trójkę złożoną z: (1) kopii bieżącego rozwiązania, (2) kopii bieżącej listu tabu oraz (3) ruchu, który doprowadził do utworzenia rozwiązania $\boldsymbol{\sigma}^*$. W przypadku, gdy algorytm nie może polepszyć najlepszego znalezionego rozwiania przez określoną liczbę iteracji `MaxNoImprove`, wtedy następuje skok powrotny do ostatniego rozwiązania z listy skoków. Odtwarzane jest bieżące rozwiązanie oraz lista Tabu. W celu zmiany trajektorii przeszukiwania sąsiedztwa do listu Tabu dodawany jest ruch, który również został zapamiętany. Po wykonaniu skoku należy usunąć go z listy skoków. Jeśli zajdzie warunek wykonania skoku, a lista skoków jest pusta, to zostanie wylosowane nowe rozwiązanie.

5.2.4 Eksperymenty obliczeniowe

Badania zostały podzielone na dwa etapy: (1) dla stałej liczby maszyn $m = 2$ oraz (2) dla dowolnej liczby maszyn. W obu testach były badane różne metody rozwiązania dla innych instancji testowych.

Badania dla stałej liczby maszyn $m = 2$

Celem eksperymentów było zbadanie wpływu danych wejściowych na działanie zaimplementowanego algorytmu heurystycznego. Dlatego zdefiniowano 5 grup instancji, podzielonych w zależności od rozrzutu wartości czasu wykonywania operacji oraz przezbrojeń: (1) 1–99, (2) 25–75, (3) 35–65, (4) 40–60, (5) 45–55. Na przykład w grupie (3) czasy wykonywania operacji i przezbrojeń zostały wylosowane z rozkładu jednostajnego $\mathcal{U}(35, 65)$. Dla każdej grupy wygenerowano 100 instancji (łącznie 4500). W każdym teście przyjęto naturalną permutację $\pi = (1, 2, \ldots, n)$ wykonywania zadań. Instancje testowe zostały wygenerowanie przy użyciu generatora liczb Mersenne Twister [88]. Użyto wersji MT19937 z domyślnymi parametrami oraz kolejnymi liczbami naturalnymi jako seed losowania. Rozważono dziewięć różnych rozmiarów problemu $n \in \{10, 15, 20, \ldots, 50\}$ dla $m = 2$.

Do rozwiązania sformułowania problemu jako MILP został użyty solver Gurobi w wersji 8.1.1. Algorytm heurystyczny podobnie jak model MILP został napisany w języku programowania Julia. Eksperymenty przeprowadzono na stacji roboczej PC z procesorem Intel Core i7-8550U @ 2.0GHz, 16 GB RAM z systemem Microsoft Windows 10.

Zaproponowano dwie miary jakości: procent uzyskanych rozwiązań optymalnych (#*opt*) oraz względne odchylenie od optymalnej długości uszeregowania (PRD). Ponadto mierzono czas T działania algorytmu heurystycznego. Otrzymane wyniki uśrednione dla każdej wielkości i grupy instancji pokazane w Tabeli 5.5. Czasy działania metody opartej na MILP przedstawiono na Rysunku 5.6 oraz dla heurystyki na Rysunku 5.7.

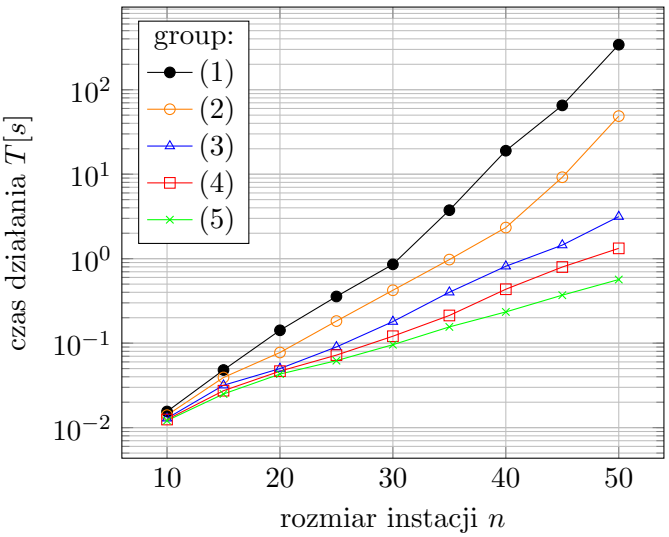
Na podstawie wyników zamieszczonych w Tabeli 5.5 można zauważyć, że zgodnie z oczekiwaniami liczba znalezionych rozwiązań optymalnych #*opt* maleję wraz ze wzrostem liczby zadań. Co ciekawsze, rozrzut czasów operacji oraz przezbrojeń ma znaczący wpływ na #*opt*. Dla pierwszej grupy (rozrzut wynosi 99, podczas gdy średni czas operacji/przezbrojenia wynosi 50, co daje około 200% względnego rozrzutu), praktycznie nie zostały znalezione optymalne rozwiązania dla $n > 20$. Natomiast w trzeciej grupie (rozrzut względny ok. 60%) liczba znalezionych rozwiązań optymalnych nie spada poniżej 60% dla $n < 50$, a heurystyka prawie dla wszystkich instancji znajduje optymalne rozwiązania dla $n < 20$. Dla grup 4 i 5 (względny rozrzut odpowiednio 40% i 20%) heurystyka znajduje prawie wyłącznie optymalne rozwiązania, nawet dla $n = 50$.

Procentowe odchylenie PRD względem referencyjnego rozwiązania uzyskanego przy pomocy MILP generalnie rośnie wraz z liczbą zadań, ale średnia w grupach instancji prawie nigdy nie przekracza 3%. Dla grupy 3. oraz późniejszych PRD zwykle pozostaje poniżej $0,3\%$. Naturalnym wnio-

Tablica 5.5: Wydajność algorytmu heurystycznego w różnych grupach instancji

liczba zadań	grupa 1 (1–99)			grupa 2 (25–75)			grupa 3 (35–65)			grupa 4 (40–60)			grupa 5 (45–55)		
	#opt [%]	PRD [%]	T [μs]	*#opt* [%]	PRD [%]	T [μs]	*#opt* [%]	PRD [%]	T [μs]	*#opt* [%]	PRD [%]	T [μs]	*#opt* [%]	PRD [%]	T [μs]
10	14	4.37	1.38	59	1.47	1.82	96	0.20	1.74	100	0	1.56	100	0	1.84
15	3	4.75	1.67	41	1.18	2.02	96	0.12	1.67	100	0	1.88	100	0	1.81
20	1	4.52	1.79	18	1.99	2.17	86	0.28	1.77	100	0	2.15	100	0	1.86
25	1	4.96	2.03	17	1.62	2.54	90	0.13	2.28	100	0	1.84	100	0	2.18
30	0	5.03	2.29	10	1.79	2.83	77	0.27	2.14	100	0	2.04	100	0	2.09
35	0	5.36	2.58	8	1.72	2.87	72	0.47	2.36	99	0	2.14	100	0	2.21
40	0	4.94	2.74	2	1.95	3.17	69	0.20	2.53	96	0.01	2.41	100	0	2.48
45	0	5.24	3.05	1	1.98	3.54	60	0.26	2.68	97	0.01	2.44	100	0	2.38
50	0	5.41	4.07	2	2.06	3.57	58	0.23	3.01	97	0.01	2.61	100	0	2.72
średnio	2.11	4.96	–	17.6	1.75	–	78.3	0.21	–	98.8	0.01	–	100	0	–

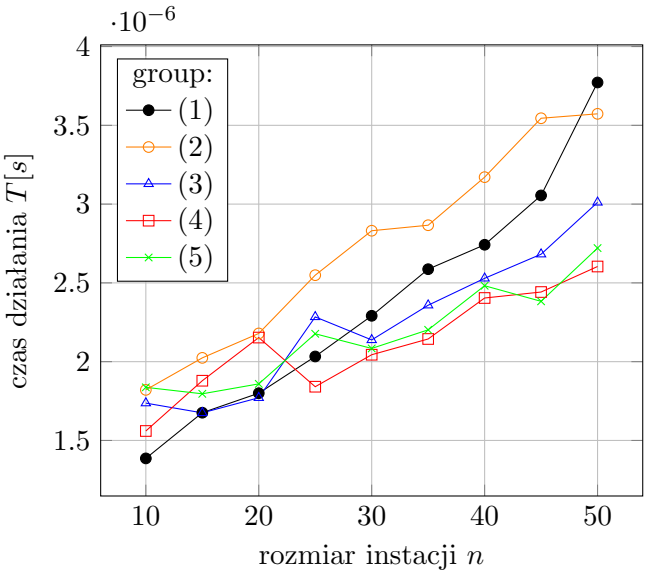
#opt – procent optymalnych rozwiązań
PRD – procentowe względne odchylenie od optymalnej długości uszeregowania T – czas działania



Rysunek 5.6: Czas działania metody bazującej na MILP dla różnych rozmiarów i grup instancji

skiem jest, że proponowana zachłanna heurystyka daje bardzo dobre wyniki, szczególnie w przypadku problemów z niskim względnym rozrzutem czasów wykonywania operacji oraz przezbrojeń. Heurystyka jest również bardzo szybka, a jej czas działania prawie nigdy nie przekracza $4\mu s$, jak widać na Rysunku 5.7. Na podstawie przeprowadzanych badań można zaobserwować, że złożoność czasowa heurystyki wykonywania wynosi $O(n)$ (dla ustalonej wartości $m = 2$), a czas wykonywania jest zasadniczo niezależny od wyboru rozrzutu czasów wykonywania operacji oraz przezbrojeń. Dzięki temu proponowaną metodę można wykorzystać jako część dwupoziomowego algorytmu, podobnego do tego opisanego w [14], aby szybko ocenić sąsiedztwo.

Algorytm bazujący na postaci MILP był metodą referencyjną dla metody zachłannej, ale na Rysunku 5.6 można zauważyć, że algorytm nadal ma zastosowanie dla $n = 50$ z czasem działania około 6 minut. Co więcej, czas działania metody maleje wraz ze zmniejszaniem rozrzutu czasów wykonywania operacji oraz przezbrojeń. Dla grupy piątej (o najmniejszym rozrzucie, 45–55) czas działania wynosi poniżej 10 sekund, co dodatkowo rozszerza zastosowania tej optymalnej metody rozwiązywania.



Rysunek 5.7: Czas działania alg. heurystycznego dla różnych rozmiarów i grup instancji

Badania dla różnej liczby maszyn $m > 1$

Celem eksperymentów było sprawdzenie skuteczności zaproponowanej własności eliminacyjnej. Wybrano 12 grup instancji o różnym rozmiarze problemu. Dla każdej grupy wygenerowano 10 instancji (łącznie 120) używając generatora zaproponowanego przez Taillarda [129]. W pierwszej kolejności wygenerowano czasy wykonywania operacji, a następnie wygenerowano czasy wykonywania przezbrojeń. Zarówno czasy wykonywania operacji jaki i przezbrojeń zostały wylosowane z rozkładu jednostajnego $\mathcal{U}(1, 99)$.

Do rozwiązania sformułowania problemu jako MILP ponownie został użyty solver Gurobi w wersji 8.1.1, włączany z poziomu języka Julia. Algorytm TS został napisany w języku programowania `C++`. Eksperymenty przeprowadzono na stacji roboczej PC z procesorem AMD Ryzen Threadripper 3990X 2.9 GHz oraz 64 GB RAM pracującej pod kontrolą systemu Microsoft Windows 10.

Ponownie jako miary jakości użyto PRD względem najlepszego rozwiązania znalezionego przez wszystkie trzy metody: (1) model MILP z ograniczeniem czasowym (2) TS z procedurą udoskonalenia najlepszego sąsiada oraz (3) TS wykorzystującego własność eliminacyjną. Jako rozwiązanie początkowe dla wszystkich metod przyjęto najlepsze z rozwiązań: (1) natu-

Tablica 5.6: Średnie PRD oraz czas działania T dla algorytmu TS w dwóch wersjach oraz sformułowania MILP

$n \times m$	PRD[%]			T[s]
	MILP	TS1	TS2	
5 × 2	0,00	0,00	0,00	100
10 × 2	0,00	0,20	0,45	100
10 × 3	0,00	1,40	1,43	100
15 × 2	0,00	0,33	0,48	100
15 × 3	0,00	0,93	1,44	100
20 × 3	0,00	1,68	1,77	100
20 × 5	0,01	0,06	0,06	100
20 × 10	0,17	0,08	0,00	100
30 × 5	0,22	0,07	0,08	100
30 × 10	0,90	0,09	0,05	100
40 × 5	0,40	0,00	0,00	100
40 × 10	0,45	0,06	0,00	100
średnia	0,18	0,41	0,49	-

ralne lub (2) zachłanne. Przyjęto limit czasowy $100s$ jako warunek stopu dla wszystkich algorytmów, w tym dla MILP. Dla algorytmów TS ustalono szerokość przeglądu sąsiedztwa $\mathtt{W} = 10\%$. Wyniki przedstawiono w Tabeli 5.6.

Przy użyciu obu wersji TS znaleziono zbliżone rozwiązania. Dla żadnej z grup instancji średnie PRD nie przekroczyło 2% względem najlepszego znalezionego wyniku. Dla liczby zadań $n \geqslant 30$ metody TS zwracają lepsze wyniki niż MILP z ograniczeniem czasowym, z lekką przewagą na korzyść metody TS wykorzystującej własność eliminacyjną. Ponadto warto zaznaczyć, że przy użyciu sformułowania MILP udało się znaleźć rozwiązania optymalne dla wszystkich instancji w grupach dla $n \leqslant 15$.

5.2.5 Wnioski i uwagi

W rozdziale zaproponowano problem szeregowania przezbrojeń w permutacyjnym przepływowym problemie szeregowania zadań z rozłącznymi przezbrojeniami niezależnymi od kolejności. Opisany problem pozwala na modelowanie rzeczywistych systemów produkcyjnych, w których tylko jedna maszyna może być w danej chwili przezbrajania ze względu na ograniczoną liczbę ekip wykonujących przezbrojenia lub konieczności użycia specjalistycznego sprzętu.

Następnie sformułowano własności dotyczące zarówno dopuszczalności, jak i liczby rozwiązań problemu. Zaproponowano również własność eliminacyjną, która pozwala wyeliminować nawet do 75% przestrzeni rozwiązania dla dwóch maszyn. Pokazano również związek między liczbą rozwiązań a liczbami Catalana. Następnie dla dowolnej liczny maszyn zaproponowano własność eliminacyjną opartą na blokach oraz procedure udoskonalania rozwiązania.

Zaproponowano kilka metod rozwiązywania problemu w celu znalezienia najlepszej wykonywania kolejności przezbrojeń: model Programowania Liniowego Całkowitoliczbowego, dwa algorytmy zachłanne oraz Przeszukiwanie z Zabronieniami. Dedykowana heurystyka dla dwóch maszyn jest bardzo szybka (czas działania w mikrosekundach) oraz zapewnia uzyskiwanie wysokiej jakości rozwiązan, szczególnie przy małych rozrzutach czasów wykonywania operacji oraz przezbrojeń. Algorytm rozwiązywania zagadnienia sformułowanego w formie MILP zapewnia optymalną kolejność σ dla instancji z 50 zadaniami w mniej niż 15 minut. W drugiej fazie badań dla dowolnej liczby maszyn algorytmy bazujące na schemacie Przeszukiwania z Zabronieniami wykorzystujące procedurę ulepszania lub własności blokowe dla małych instancji dawały wartości zbliżone do optymalnych. Proponowane metody mogą w przyszłości znaleźć zastosowanie jako szybkie rozwiązywania dolnego poziomu w dwupoziomowym problemie, gdzie priorytetem będzie znalezienie najlepszej kolejności wykonywania nie tylko przezbrojeń, ale i zadań.

Część III

Zastosowania praktyczne

Rozdział 6

Aplikacje

W niniejszym rozdziale zostaną przedstawione aplikacje opracowanej metodologii rozwiązywania problemów ze sprzężeniami czasowymi (opisanych w poprzednich rozdziałach) dla rzeczywistych problemów spotykanych w praktyce. Przedstawione zagadnienia zostały opracowane jako wynik współpracy z zewnętrznymi firmami. Zaproponowane podejście okazało się na tyle efektywne, że przy jego zastosowaniu można nie tylko analizować procesy przemysłowe, ale także modelować problemy z dziedziny logistyki i transportu. W Podrozdziale 6.1 zaprezentowano studium przypadku dla problemu szeregowania rozłącznych przezbrojeń na przykładzie firmy zajmującej się produkcją urządzań AGD. W Podrozdziale 6.2 przedstawiono przykład przedsiębiorstwa logistycznego, które zajmuje się krajowym transportem towarów z centralnego magazynu [62]. Z kolei w Podrozdziale 6.3 poddano analizie działalność przedsiębiorstwa, które zajmuje się odbiorem towarów od producentów wyrobów mlecznych w ramach stałej struktury komunikacyjnej.

6.1 Produkcja urządzeń AGD

W poniższym rozdziale zostanie omówione studium przypadku dla problemu przepływowego szeregowania zadań z rozłącznymi przezbrojeniami, w oparciu o bazę danych o czasach trwania pozyskaną w ramach współpracy Politechniki Wrocławskiej z firmą Electrolux. Przedsiębiorstwo zajmuje się produkcją m. in. urządzeń AGD. Proces produkcyjny obejmuje kształtowanie elementów metalowych przy użyciu pras hydraulicznych. Ze względu na duży rozmiar pras, w celu zmiany konfiguracji prasy (wymiany sztancy o wadze koło 1 tony) niezbędne jest użycie specjalistycznego wózka. Obsługą wózka widłowego i przezbrajaniem zajmuje się (w ramach zmiany) jedna

ekipa, więc jednocześnie może być wykonywane tylko jedno przezbrojenie. Analizowany proces jest zbliżony do dwupoziomowego problemu opisanego w Podrozdziale 5.2.

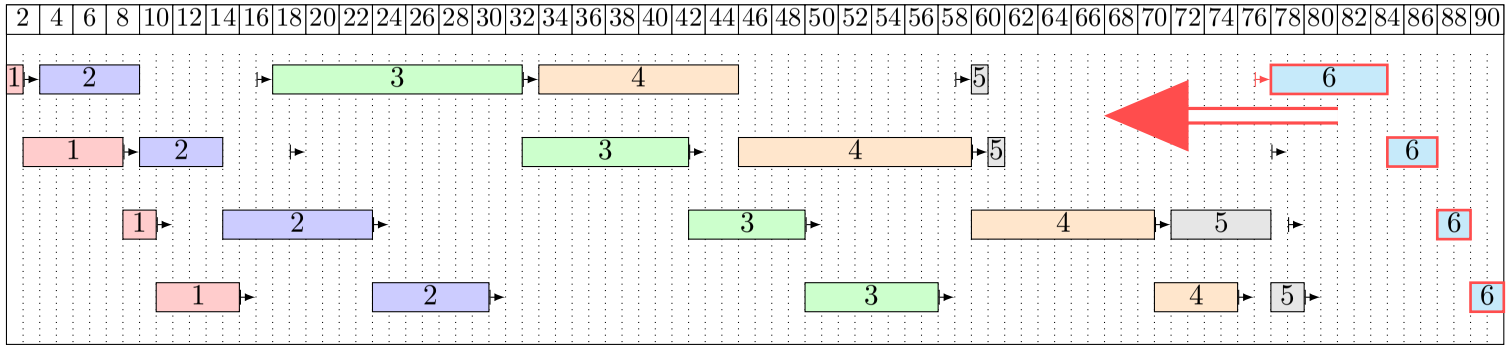
6.1.1 Definicja problemu

Każde zdanie ze zbioru zadań $\mathcal{J} = \{1, 2, \ldots, n\}$ opisuje proces wytwarzania elementów serii urządzeń danego typu. Wszystkie elementy muszą przejść przez serię czterech pras hydraulicznych, które oznaczono poprzez zbiór maszyn $\mathcal{M} = \{1, 2, 3, 4\}$. Czas wykonywania operacji z i-tego zadania na a-tej maszynie zależy od liczby urządzeń oraz ich typu i wynosi $p_i^a > 0$. Prasy należy przezbroić między wykonywaniem elementów urządzeń różnego typu. Przezbrojenie polega na zmianie odpowiedniej tzw. sztancy w prasie, bądź jej przezbrojenia. Czas wykonywania przezbrojenia jest niezależny od wytarzanego urządzania, więc jest stały $s_{i,j}^a = 1$ dla każdego $i, j \in \mathcal{J}$ oraz $a \in \mathcal{M}$. Przezbrojenie wymaga użycia specjalistycznego sprzętu, dlatego w danym momencie czasowym może być wykonywano jednocześnie wyłącznie jedno przezbrojenie.

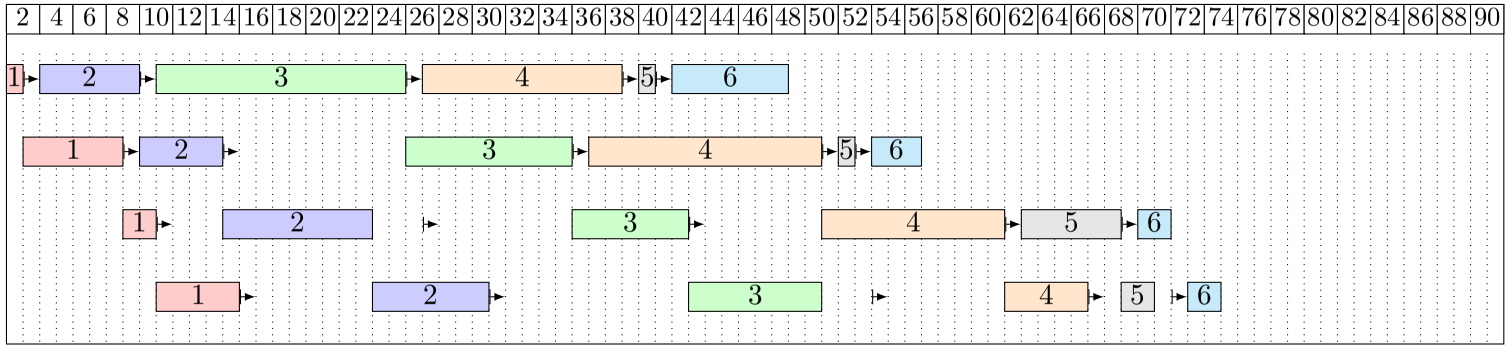
W Tabeli 6.1 przedstawiono instancje problemu stworzoną na bazie rzeczywistych danych uzyskanych od przedsiębiorstwa. Dane przedstawiają produkcję 6 typów urządzeń z wykorzystaniem 4 pras hydraulicznych. Na Rysunku 6.1a pokazano harmonogram wyznaczony dla domyślniej kolejności $\sigma = (1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4)$, gdzie przezbrojenia są wykonywane kolejno na następujących po sobie maszynach. W łatwy sposób można zaobserwować (zobacz czerwona strzałka na Rysunku 6.1a), że w tym wypadku nie jest to najlepsze rozwiązanie. Poprzez samo wcześniejsze wykonanie przezbrojenia na pierwszej maszynie przed zadaniem 6-tym można skrócić długość uszeregowania $C_{\max}$. Dzięki zastosowaniu metod optymalizacyjnych opisanych we wcześniejszych rozdziałach, w szczególności bazujących na Przeszukiwaniu z Zabronieniami, dla tego konkretnego przykładu udało się skrócić długość uszeregowania o blisko $17,8\%$. Skrócony harmonogram po optymalizacji pokazano na Rysunku 6.1b. Dalsza poprawa jakości rozwiązania wymagałaby zmiany kolejności wykonywania zadań.

6.1.2 Wnioski i uwagi

W rozdziale pokazano studium przypadku dla problemu szeregowania rozłącznych przezbrojeń w problemie przepływowym szeregowania zadań. Wykorzystano rzeczywiste dane pozyskane w ramach współpracy z zewnętrzna firmą. W celu poprawy rozwiązania użyto metod opisanych w Podroz-



(a) Harmonogram dla $\sigma = (1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4)$ przed optymalizacją.



(b) Harmonogram dla $\sigma = (1, 2, 1, 3, 2, 4, 1, 3, 4, 2, 1, 1, 3, 2, 2, 4, 3, 4, 3, 4)$ po optymalizacji.

Rysunek 6.1

a	p_1^a	p_2^a	p_3^a	p_4^a	p_5^a	p_6^a
1	1	6	15	12	1	7
2	6	5	10	14	1	3
3	2	9	7	11	6	2
4	5	7	8	5	2	2

Tablica 6.1: Instancja stworzona na podstawie rzeczywistych danych.

dziale 5.2. Kolejnym etapem jest zidentyfikowanie własności dla problemu dwupoziomowego, gdzie będzie należało znaleźć kolejność wykonywania zadań i dla niej ustalić kolejność wykonywania przezbrojeń.

6.2 Transport spedycyjny

W tym rozdziale zostanie przedstawiony przykład aplikacji dotyczącej zarządzania flotą pojazdów ciężarowych w firmie logistycznej zajmującej się spedycją towarów z centralnego magazynu. Firma zajmuje się wysyłaniem transportów na terenie całych Stanów Zjednoczonych Ameryki, jednak nie zajmuje się „ściąganiem" pustych pojazdów z powrotem do magazynu centralnego (to zadanie innej firmy). Pojemność pojazdów jest z góry ograniczona, jednak efektywne zapakowanie pojazdów oraz ułożenie towarów nie jest przedmiotem optymalizacji. Dodatkowo należy uwzględnić czas pracy kierowców wynikający z przepisów federalnych.

Proces automatyzacji zarządzania firmą spedycyjną sprowadza się do problemu wyznaczenia tras dla floty pojazdów tak, aby dostarczyć towary dla wszystkich klientów z centralnego magazynu. W literaturze problem ten jest określany jako problem marszrutyzacji pojazdów (*Vehicle Routing Problem*) [33]. VRP można traktować jako uogólnienie klasycznego problemu komiwojażera [31], zatem również, tak jak TSP, należy do klasy problemów NP-trudnych. Ponadto wariant VRP bez powrotów pojazdów do magazynu centralnego rozpatrywany jest w literaturze jako otwarty problem marszrutyzacji pojazdów (*Open Vehicle Routing Problem* OVRP) [116].

VRP to dobrze znany problem z wieloma potencjonalnymi ograniczeniami praktycznymi, jak VRP z ograniczoną pojemnością pojazdów (*Capacitated Vehicle Routing Problem*, CVRP) [135] lub z oknami czasowymi (*Vehicle Routing Problem with Time Windows*,VRP-TW) [34], co wymusza dostawę ładunku w określonych ramach czasu. Jednak takie rozszerzenia nie zawsze są wystarczające do modelowania rzeczywistych przypadków. Przykładem takiej sytuacji jest uwzględnienie przepisów ruchu drogowego, które

poza ograniczeniem prędkości również zabraniają kierowcom ciągłej jazdy bez przerw.

6.2.1 Definicja problemu

W literaturze można znaleźć przykłady rozwiązywania problemów optymalizacyjnych przy użyciu metod dedykowanych innym problemom [7]. Problem wyznaczania tras dla floty pojazdów ciężarowych może być również zamodelowany jako problem szeregowania zadań na identycznych oraz równoległych maszynach (*Identical Parallel Machines Scheduling* IPMS) [11]. Każdego klienta można opisać jako zadanie; przez $\mathcal{J} = \{1, 2, \ldots, n\}$ oznaczono zbiór n zadań (klientów). Każdy pojazd może być opisany jako maszyna, przez $\mathcal{M} = \{1, 2, \ldots, m\}$ oznaczono zbiór m maszyn (pojazdów). Każde i-te zadanie składa się dokładnie z jednej operacji. W systemie będzie dokładnie tyle samo operacji co zadań, $o = n$. Założono, że pojazdy wyjeżdżają z magazynu centralnego (*hub*) załadowane i czas załadunku nie jest wliczany do ich czasu pracy. Jednak należy wyładować ładunek u klienta, co będzie rozumienie przez obsługę/wykonanie i-tego zadania i oznaczone przez $p_i > 0$ (czasu na maszynie). Czas wykonywania zadania jest niezależny od maszyny, ponieważ wszystkie pojazdy są takie same. Odległości w milach między klientami oraz magazynem centralnym zostaną opisane jako macierz kosztów przezbrojeń $\hat{\mathcal{S}}$:

$$\hat{\mathcal{S}} = [\hat{s}_{i,j}]^{(n+1)\times(n+1)} \tag{6.1}$$

gdzie $\hat{s}_{i,j}$ oznacza koszt wykonania przezbrojenia z zadania i-tego na zadanie j-te – odległość do przejechania od klienta i-tego do j-tego. Ponadto przez $\hat{s}_{0,j}$ oznaczono koszt przejechania z magazynu centralnego do pozostałych $j \in \mathcal{J}$ klientów.

Koszty wykonania przezbrojeń $\hat{s}_{i,j}$ należy przeliczyć na czas, więc ze względu na różnice w lokalnych oraz stanowych przepisach drogowych przyjęto średnią prędkość 45 mil na godzinę dla odległości poniżej 100 mil oraz 60 mil na godzinę na dłuższych dystansach, przy założeniu, że w tym drugim przypadku używane są autostrady międzystanowe i główne drogi państwowe. Czas wykonania przezbrojenia oznaczono przez $s_{i,j}$ oraz przeliczono następująco:

$$s_{i,j} = \begin{cases} \hat{s}_{i,j}/45 & \text{dla } d_{i,j} \leqslant 100, \\ \hat{s}_{i,j}/60 & \text{w p. p.} \end{cases} \tag{6.2}$$

Jest to czas wykonania przezbrojenia po zadaniu i-tym, a przed zadaniem j-tym – czas przejechania od klienta i-tego do j-tego.

Należy pamiętać, że każdy towar dostarczany do klienta ma określone parametry: wagę i rozmiar. W literaturze można znaleźć problemy szeregowania na identycznych maszynach z ograniczoną pojemnością [143]. Każde zadanie zostanie więc dodatkowo opisane poprzez wagę $\omega_i > 0$ w funtach oraz długość $\lambda_i > 0$ w stopach. Długość ładunku (towaru zapakowanego na palecie) jest w pełni wystarczającym opisem rozmiaru ładunku przy założeniu, że nie można układać ładunków na sobie oraz ładunek ma szerokość zbliżoną do szerokości przestrzeni transportowej.

Przez π_a oznaczono n-elementową kolejność zadań wykonywanych na maszynie a, czyli trasę odwiedzanych klientów przez pojazd a. Zatem $\pi_a(i)$ oznacza i-te zadanie wykonywane w kolejności π_a na maszynie a oraz i-tego odwiedzonego klienta przez pojazd a . Rozwiązaniem będzie nazywany zestaw kolejności wykonywania zadań na każdej maszynach $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_m)$ dla wszystkich $a \in \mathcal{M}$ pojazdów.

Na podstawie rozwiązania $\boldsymbol{\pi}$ można zbudować odpowiadający mu *harmonogram* $(\boldsymbol{S}, \boldsymbol{C})$. Harmonogram składa się z dwóch wektorów: (1) wektora momentów rozpoczęcia wykonywania operacji $\boldsymbol{S} = [S_i]^n$, gdzie S_i jest czasem rozpoczęcia wykonywania i-tej operacji, czyli czasem rozpoczęcia obsługi i-tego ładunku oraz (2) wektor momentów zakończenia wykonywania operacji $\boldsymbol{C} = [C_i]^n$, gdzie C_i jest czasem zakończenia wykonywania i-tej operacji, czyli czasem zakończenia rozładowania ładunku u i-tego klienta. Aby harmonogram $(\boldsymbol{S}, \boldsymbol{C})$ był dopuszczalny, musi spełniać wszystkie następujące warunki:

$$\forall a \in \mathcal{M} \quad S_i, C_i \geqslant 0, \qquad i = 1, 2, \ldots, n, \tag{6.3}$$

$$C_i = S_i + p_i, \qquad i = 1, 2, \ldots, n, \tag{6.4}$$

$$\forall a \in \mathcal{M} \quad s_{0,\pi_a(1)}, \leqslant S_{\pi_a(1)}, \tag{6.5}$$

$$\forall a \in \mathcal{M} \quad C_{i-1} + s_{i-1,i} \leqslant S_i, \qquad i = \pi_a(2), \pi_a(3), \ldots, \pi_a(|\pi_a|), \tag{6.6}$$

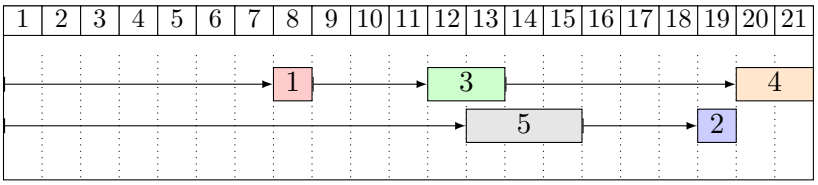
$$\forall a \in \mathcal{M} \quad \sum_i \lambda_i \leqslant \Lambda, \qquad i = \pi_a(1), \pi_a(2), \ldots, \pi_a(|\pi_a|), \tag{6.7}$$

$$\forall a \in \mathcal{M} \quad \sum_i \omega_i \leqslant \Omega, \qquad i = \pi_a(1), \pi_a(2), \ldots, \pi_a(|\pi_a|), \tag{6.8}$$

gdzie przez Λ oraz Ω oznaczono odpowiednio maksymalną sumaryczną długość ładunków oraz maksymalną sumaryczną wagę ładunków dla pojedynczego pojazdu. Ograniczenie (6.4) zapewnia ciągłość obsługi ładunku u klienta. Ograniczenie (6.5) wymusza dojazd do pierwszego klienta z magazynu centralnego. Ograniczenie (6.6) zapewnia wykonanie przejazdu między kolejnymi klientami, ale również gwarantuje, aby czas rozpoczęcia wykonywania obsługi kolejnego klienta był nie mniejszy od czasu zakończenia poprzednika. Ograniczenia (6.7) oraz (6.8) zapewniają odpowiednio, aby nie

Tablica 6.2: Instancja problemu COVRP z Przykładu 6.1.

i	p_i	$s_{i,0}$	$s_{i,1}$	$s_{i,2}$	$s_{i,3}$	$s_{i,4}$	$s_{i,5}$
0	-	0	**12**	8	10	15	**7**
1	1	12	0	21	**3**	7	16
2	1	8	21	0	19	16	3
3	2	10	3	19	0	**6**	16
4	2	15	7	16	6	0	12
5	3	7	16	3	16	12	0



Rysunek 6.2: Harmonogram dla rozwiązania π z Przekładu 6.1.

została przekroczona dozwolona długość oraz masa wszystkich ładunków dla a-tego pojazdu.

Na podstawie ograniczenia (6.4) można zaobserwować, że dopuszczalny harmonogram można jednoznacznie opisać za pomocą $\boldsymbol{S}$ lub $\boldsymbol{C}$. Harmonogram będzie nazywany w lewo dosuniętym (*left-shifted*), jeśli ani jedna operacja transportu nie może rozpocząć wykonywać się wcześniej bez naruszenia ograniczeń lub zmiany kolejności wykonywania transportów.

Przykład 6.1 W Tabeli 6.2 pokazano instancje problemu COVRP. Na Rysunku 6.2 przedstawiono w lewo dosunięty harmonogram dla tej instancji oraz permutacji $\boldsymbol{\pi} = ((1, 3, 4)(5, 2))$. □

Badanym kryterium optymalizacyjnym będzie suma długości uszeregowań na każdej maszynie odpowiadającą dla rozważanego problemu sumarycznego czasowi transportu wszystkich ładunków. Przez $\sum C^a(\boldsymbol{\pi})$ oznaczono sumę czasów zakończenia wykonywania operacji ostaniach operacji na wszystkich maszynach, które zakończyły się wykonywać w lewo dosuniętym harmonogramie dla kolejności $\boldsymbol{\pi}$. Z ograniczeń (6.4)–(6.8) wynika, że będzie to:

$$\sum C^a(\boldsymbol{\pi}) = \sum_{a \in \mathcal{M}} C^a_{\pi_a(|\pi_a|)} \tag{6.9}$$

gdzie $|\pi_a|$ to liczba elementów w π_a. Celem będzie znalezienie takiego rozwiązania π, że:

$$\sum C^a(\boldsymbol{\pi}^{\text{opt}}) = \min_{\boldsymbol{\pi}\in\Pi} \sum C^a(\boldsymbol{\pi}), \tag{6.10}$$

gdzie Π jest zbiorem wszystkich dopuszczalnych rozwiązań. Rozwiązanie $\boldsymbol{\pi}^{\text{opt}}$ będzie nazywane rozwiązaniem optymalnym.

Przepisy drogowe zabraniają kierowcom zbyt długiej jazdy, wymuszając przerwy na odpoczynek. Ograniczenia te można podsumować w 4 następujących punktach:

1. 8-godzinny limit: kierowca może jechać przez maksymalnie 8 godzin bez przerwy, jeśli zrobił 30-minutową przerwę przed jazdą,
2. Limit 11-godzinny: kierowca może jechać maksymalnie 11 godzin bez przerwy, jeśli zrobił 10-godzinną przerwę przed jazdą,
3. 14-godzinny limit: kierowca może pracować (łącznie z prowadzeniem pojazdu, załadunkiem i rozładunkiem itp.) przez maksymalnie 14 godzin bez przerwy, jeśli zrobił 10-godzinną przerwę przed pracą,
4. 70-godzinny limit: kierowca może pracować przez 70 godzin przez 8 kolejnych dni. Mogą zostać wznowione po 34-godzinnej nieprzerwanej przerwie.

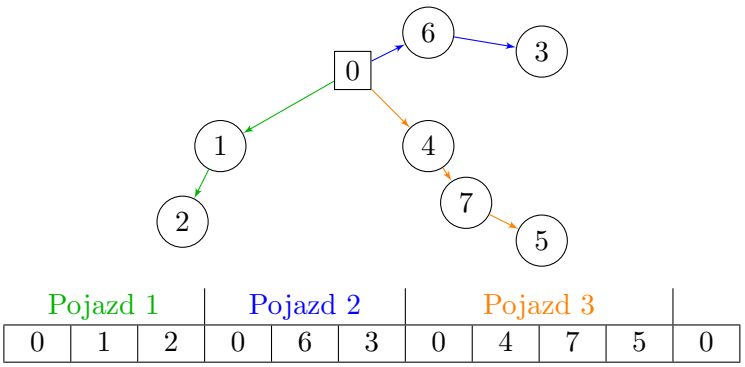
Dla pierwszego ograniczenia należy zawsze doliczyć 0.5 godziny dla każdego przezbrojenia $s_{i,j} > 8h$. Dla ograniczenia drugiego i trzeciego należy doliczyć 10 godzin do czasu zakończenia ostatniego zadania na maszynie za każdym razem, gdy któryś z limitów został przekroczony oraz analogicznie dla ostatniego ograniczenia doliczając 34 godziny.

6.2.2 Algorytmy optymalizacyjne

Poniżej zostanie zdefiniowana reprezentacja wielkiej trasy (*Giant Tour Representation*, GTR). Następnie będą zaproponowane dwie metody rozwiązania problemu: (1) metoda zachłanna bazująca na rozwiązaniu stosowanym w przedsiębiorstwie oraz (2) Przeszukiwanie z Zabronieniami.

Reprezentacja wielkiej trasy

W celu szybszego przeglądania sąsiedztwo zastosowano tzw. wielką trasę, GTR [44]. Zabieg ten pozwala na przedstawienie kompletnego rozwiązania za pomocą pojedynczej permutacji o długości $n + m + 1$. W rozwiązaniu jako GTR wszystkie trasy są umieszczane sekwencyjnie, oddzielone zerami. Wartość zero jest również umieszczana na początku i na końcu rpzwiązania. Pozwala to na zmianę liczby klientów na trasie poprzez zmianę pozycji zer. Jeśli obok siebie pojawiają się dwa zera, to trasa jest pusta (nie ma



Rysunek 6.3: Reprezentacja wielkiej trasy

klientów), co jest dopuszczalne (pojazd nie wyjeżdża z magazynu centralnego). Na Rysunku 6.3 rozważono przykładowe rozwiązanie w formie GTR dla $n = 7$, $m = 3$ oraz $\boldsymbol{\pi} = ((1, 2), (6, 3), (4, 7, 5))$.

Algorytm zachłanny

Metoda zachłanna bazuje na algorytmie stosowanym obecnie w rozważanym studium przypadku firmy spedycyjnej. Algorytm działa podobnie do metody szukania k-najbliższych sąsiadów (*K-Nearest Neighbor Search*) [122]. W każdej iteracji algorytmu należy najpierw znaleźć najdalej położonego klienta od centralnego magazynu (o największym czasie wykonywania przezbrojenia $s_{0,i}$ dla $i \in \mathcal{J}$). Następnie należy dobierać najbliższych sąsiadów (zadania, które wymagają najkrótszego czasy przezbrojenia) dopóki nie zostanie przekroczona pojemność pojazdu/maszyny. Na koniec należy odwrócić pełną kolejność wykonywania zadań. W Algorytmie 18 przedstawiono schemat metody.

Przeszukiwanie z Zabronieniami

Algorytm TS [48] jest lokalną metodą przeszukiwania sąsiedztwa. Algorytm wymaga rozwiązania początkowego, które jest zazwyczaj losowe (ale musi być dopuszczalne) lub uzyskane za pomocą algorytmu, np. zachłannego. Następnie przy wykorzystaniu ruchów, np. typy zamień, jest wyszukiwane najlepsze rozwiązanie w sąsiedztwie. Rozwiązania będące na liście zakazów są ignorowane. Lista zakazów przechowuje atrybuty ruchu, który doprowadził do utworzenia najlepszego sąsiada, w celu uniknięcia w lokalnym optimum. Lista zakazów została zaimplementowana w formie macierzy, która

Algorytm 18: Algorytm zachłanny

Dane: N: zbiór zadań nieuszeregowanych
Wynik: $\boldsymbol{\pi}$: kolejność wykonywania zadań.

```
1  function Greedy(N):
2      π ← ∅;
3      while N ≠ ∅ do
4          π.PushBack(0);
5          i* ← arg max_{i∈N} s_{0,j};
6          π.PushBack(i*);
7          weight ← ω_{i*};
8          size ← λ_{i*};
9          N ← N \ {i*} while N ≠ ∅ do
10             i* ← arg max_{i∈N} s_{i*,i};
11             if weight + ω_{i*} ⩽ weightLimit ∧ size + λ_{i*} ⩽ size then
12                 π.PushBack(i*);
13                 weight ← weight + ω_{i*};
14                 size ← size + λ_{i*};
15                 N ← N \ {i*}
16             else
17                 break;
18     π.PushBack(0);
19     π ← π.Reverse();
20     return π;
```

zawiera informację od której iteracji dany ruch będzie dostępny, co pozwala na dodawanie lub sprawdzanie dostępności danego ruchu w czasie stałym.

Dla problemów z ograniczoną pojemnością maszyn/pojazdów bardzo łatwo wygenerować rozwiązanie niedopuszczalne poprzez stosowanie standardowych typów ruchów. W przypadku dwóch zapełnionych maszyn a i b, zamiana dowolnego zadania na maszynie a z dowolnym „większym" zadaniem na maszynie b doprowadzi do złamania ograniczeń dotyczących ładowności – wygenerowania rozwiązania niedopuszczalnego. W celu ominięcia tego efektu można stosować ruchy złożone [53]:

1. podwójny zamień (*double swap*); poza bazową parą (i, j), należy jeszcze zamienić parę $(i + 1, j + 1)$,

2. zamień i wstaw (*swap and insert*); poza bazową parą (i, j), należy jeszcze wyciąć zadanie $i + 1$ i następnie wstawić na pozycji $j + 1$.

Przeglądanie ruchów złożonych może być zaimplementowane bezpośrednio po standardowym przeglądzie sąsiedztwa, ale nie zamiast niego.

6.2.3 Aplikacja użytkownika

W porozumieniu z zewnętrzna firmą powstał prototyp aplikacji. Program został wykonany w całości w języku programowania C#. Aplikacje wyposażono w graficzny interfejs użytkownika (*Graphical User Interface*, GUI) zgodny z `.NET Framework 5.0`. Program stanowi kompleksowe rozwiązanie dla firmy spedycyjnej, ponieważ zawiera:

- dodawanie nowych lokalizacji – z wykorzystaniem `Google API Gecode` (do pobierania współrzędnych) oraz `Google API Distance Matrix` (do pobierania odległości między miastami), podczas dodawania elementów wymagane jest połączenie z internetem,
- dodawanie nowych klientów – nazwa, lokalizacja, długość ładunku, masa ładunku oraz data utworzenia zlecania (wybrana z kalendarza),
- wyznaczanie tras – z wykorzystaniem zaproponowanej Metody TSC, dodatkowo uproszczona wizualizacja na mapie oraz parametry trasy (klienci, wypełnienie pojazdu, przejechany dystans).

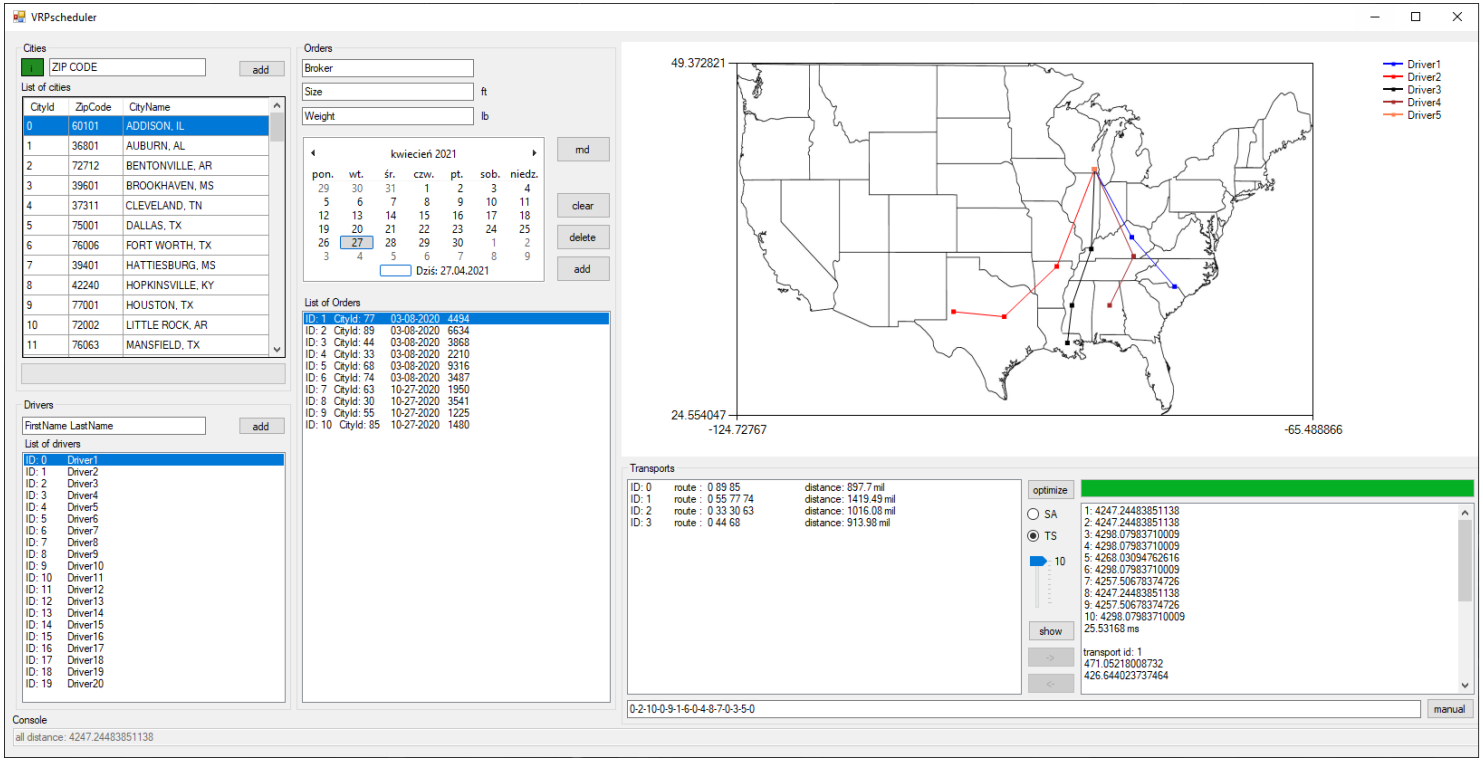
Na Rysunku 6.4 przedstawiono GUI programu z rozwiązaniem dla 10 losowych klientów. W planach jest dalsze rozwijanie opracowanego systemu.

6.2.4 Eksperymenty obliczeniowe

Badania przeprowadzono na komputerze PC wyposażonym w procesor Intel i7-6700K taktowany zegarem 4,00 GHz, 16 GB pamięci RAM oraz dysk SSD pracujący pod kontrolą systemu operacyjnego Microsoft Windows 10.

W celu zweryfikowania wyników zostały zaimplementowane oba zaproponowane w Podrozdziale 6.2.2 algorytmy: (1) algorytm zachłanny oraz (2) Przeszukiwanie z Zabronieniami. Wykorzystano rzeczywistą sieć drogową w Stanów Zjednoczonych Ameryki. Została zaprojektowana automatyczna funkcja, która przy użyciu biblioteki Distance Matrix z Google API, pozwala na pobranie wszystkich rzeczywistych odległości między miastami z bazy klientów.

Do testów wybrano 96 różnych miast na terenie USA oraz magazyn centralny umieszczony w okolicach Chicago. Bazując na rzeczywistych danych wygenerowano instancje testowe o zbliżonych parametrach. Dla każdego klienta losowano miasto, wagę ładunku oraz długość ładunku. Dla



Rysunek 6.4: Widok na interfejs użytkownika systemu marszrutyzacji

Tablica 6.3: Średnie PRD oraz czasy działania metod TS oraz algorytmu zachłannego

L. klientów	Czacy wykonywania			PRD	
n	G[ms]	TS[s]	TSC[s]	TS[%]	TSC[%]
5	17,21	0,01	0,01	0	0
10	29,01	0,02	0,03	$-0,21$	$-2,31$
15	48,54	0,06	0,10	$-1,51$	$-3,17$
20	89,34	0,13	0,22	$-1,79$	$-3,05$
25	109,61	0,24	0,42	$-1,49$	$-3,53$
50	196,49	1,75	3,13	$-1,89$	$-4,42$
75	344,51	5,91	10,80	$-2,32$	$-3,92$
100	553,21	13,40	25,20	$-2,11$	$-5,26$
średnia	–	–	–	$-1,41$	$-3,08$

uproszczenia jako adres przyjęto centrum docelowego miasta oraz dopuszczono klientów o tym samym adresie. Limit długości oraz dopuszczalna maksymalna waga całego ładunku dla każdego pojazdu zostały ustawione odpowiednio na $\Lambda = 53$ *stóp* oraz $\Omega = 45000$ *funtów*.

W celu pomiaru jakości rozwiązań dostarczanych przez oba algorytmy zastosowano procentowe odchylenie względne (PRD), które definiuje się w zależności od rozwiązania jako:

$$\mathrm{PRD}(\pi) = 100\% \frac{\sum C^a(\pi) - \sum C^a(\pi^{\mathrm{ref}})}{\sum C^a(\pi^{\mathrm{ref}})}, \tag{6.11}$$

gdzie π i π^{ref} są rozwiązaniami uzyskanymi odpowiednio przez TS oraz metodę zachłanna (jako odniesienie), a $\sum C^a(\pi)$ jest wartością funkcji celu dla rozwiązania π. Wyniki, w tym PRD i czas działania algorytmów, przedstawiono w Tabeli 6.3. Przez G, TS, oraz TSC oznaczono obwiednio metody: zachłanną, Przeszukiwanie z Zabronieniami oraz Przeszukiwanie z Zabronieni wykorzystujące ruchy złożone.

Wyniki wskazują, że metody bazujące na schemacie TS oraz TSC pozwalają na uzyskanie mniejszych całkowitych długości tras niż te uzyskane metodą zachłanną, odpowiednio średnio o $1,41\%$ oraz $3,08\%$. Warto zauważyć również, że przewaga metod TS i TSC rośnie wraz z rozmiarem problemu, czyli dla trudniejszych przypadków.

Ponadto wszystkie rozważane metody działają w ciągu kilku sekund (poniżej 0,5 sekundy w przypadku G oraz poniżej pół minuty dla obu wariantów TS), dzięki czemu można je łatwo wykorzystać w rzeczywistych

sytuacjach. Zauważamy również, że wariant TSC wymaga jedynie dwukrotnie dłuższego czasu obliczeń niż wariant TS bez mechanizmu ruchów złożonych, ale pozwala uzyskać nawet dwukrotnie większą poprawę względem rozwiązania referencyjnego, pozostając jednocześnie bardzo szybki.

6.2.5 Wnioski i uwagi

W rozdziale zaprezentowano przykład zastosowania metod szeregowania zadań do rozwiązywania rzeczywistego problemu optymalizacyjnego. W procesie optymalizacji użyto metody zachłannej bazującej na rzeczywistym procesie oraz dwóch wersji algorytmu Przeszukiwania z Zabronieniami. Zaproponowane sformułowanie problemu oraz algorytm optymalizacyjny pozwoliły na uzyskanie wyników lepszych niż zachłanna heurystyka stosowana obecnie w rozważanej firmie. W przyszłości można rozbudować algorytmy TS użyte w prototypie o własności blokowe zaproponowane w Podrozdziałach 4.2 oraz 5.2, lub dedykowane sąsiedztwo (Podrozdział 5.1).

6.3 Obiór od stałych dostawców

W tym rozdziale analizowany jest problem zarządzania flotą pojazdów ciężarowych wyposażonych w cysterny wykonujące zlecone transporty mleka dla lokalnego przedsiębiorstwa zajmującego się odbiorem towarów od lokalnych dostawców. Opisywany problem ma dwa poziomy: (1) należy dobrać odpowiednią flotę pojazdów z dostępniej puli o określonych pojemnościach, (2) należy wyznaczyć trasy dla każdego pojazdu. Kierowców ograniczają przepisy polskiego kodeksu ruchu drogowego oraz 8-godziny dzień pracy. Ponadto pojawiają się dodatkowe ograniczenia: (1) należy zadbać o odpowiedni poziom wypełnienia cysterny, (2) pojazdy powinny wracać do bazy w różnych odstępach czasowych (z uwagi na czasochłonne zlewanie zawartości do głównego zbiornika w bazie).

6.3.1 Definicja problemu

Analogicznie jak w Rozdziale 6.2 analizowany problem można rozpatrzyć jako problem szeregowania zadań, ale w tym przypadku na niejednorodnych, równoległych maszynach (*Unrelated Parallel Machines Scheduling*, UPMS) [39]. Ponownie klienci zostali opisani jako zbiór $\mathcal{J} = \{1, 2, \ldots, n\}$ zawierający n zadań, a pojazdy jako zbiór $\mathcal{M} = \{1, 2, \ldots, m\}$ składający się z m maszyn. Ponadto każde zadanie składa się z dokładnie jednej operacji, a sumaryczna liczba operacji jest większa niż liczba zadań $o > n$, ponieważ na każdej maszynie jako ostatnią dodano jedną operację specjalną. Ostatnia

operacja będzie opisywała rozładowanie pojazdu oraz jego wyczyszczenie w magazynie centralnym (bazie).

Ponownie odległości oraz czasy przejazdów opisane są jako odpowiednio macierz kosztów $\hat{\mathcal{S}}$ oraz macierz czasów $\mathcal{S}$ wykonywania przezbrojeń. W przypadku odległości podano w kilometrach oraz czasy w godzinach. Macierz $\mathcal{S}$ przeliczono na podstawie macierzy $\hat{\mathcal{S}}$ przyjmując średnią prędkość pojazdów równą 40 $\frac{km}{h}$.

Ładunek odbierany od klienta wystarczy opisać jednym parametrem: przez ω_i dla $i \in \mathcal{J}$ oznaczono masę ładunku w litrach. Jako Ω^a oznaczono pojemność a-tej maszyny (pojazdu), ponieważ cysterny mają różną pojemność. Na czas wykonywania zadania p_i składa się czas potrzebny na złożenie oraz rozłożenie niezbędnego sprzętu do pompowania (około $2 \cdot 3\ min$) oraz czas potrzebny na przepompowanie mleka pompą o wydajności 900 $\frac{l}{min}$.

Przez π_a oznaczono n-elementową kolejność wykonywania zadań na maszynie a. Zatem $\pi_a(i)$ oznacza i-te zadanie wykonywane w kolejności π_a na maszynie a. Rozwiązaniem będzie nazywana sekwencja kolejności wykonywania zadań na wszystkich maszynach $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_m)$ dla $a \in \mathcal{M}$. Ponadto na każdej maszynie, która ma przydzielone chociaż jedno zdanie, będzie dodane na koniec jedno zadanie, które jest odpowiedzialne za rozładowanie oraz umycie pojazdu w magazynie centralnym. W danym momencie może być rozładowywany oraz czyszczony tylko jeden pojazd.

Na podstawie rozwiązania $\boldsymbol{\pi}$ można zbudować odpowiadający mu *harmonogram* $(\boldsymbol{S}, \boldsymbol{C})$. Harmonogram składa się z dwóch wektorów: (1) wektora momentów rozpoczęcia wykonywania operacji $\boldsymbol{S} = [S_i]^n$, gdzie S_i jest czasem rozpoczęcia wykonywania i-tej operacji oraz (2) wektora momentów zakończenia wykonywania zadań $\boldsymbol{C} = [C_i]^n$, gdzie C_i jest czasem zakończenia wykonywania i-tej operacji. Harmonogramu $(\boldsymbol{S}, \boldsymbol{C})$, aby był dopuszczalny, musi spełniać następujące ograniczenia:

$$C_i = S_i + p_i, \qquad \text{dla} \quad i = 1, 2, \ldots, n, \tag{6.12}$$

$$\forall a \in \mathcal{M} \quad s_{0,\pi_a(1)} \leqslant S_{\pi_a(1)}, \tag{6.13}$$

$$\forall a \in \mathcal{M} \quad C_{i-1} + s_{i-1,i} \leqslant S_i, \qquad i = \pi_a(2), \pi_a(3), \ldots, \pi_a(|\pi_a|), \tag{6.14}$$

$$\forall a \in \mathcal{M} \quad \sum_i \omega_i \leqslant \Omega^a, \qquad i = \pi_a(1), \pi_a(2), \ldots, \pi_a(|\pi_a|). \tag{6.15}$$

Ograniczenie (6.12) zapewnia ciągłość wykonywania zadań. Następnie ograniczenie (6.5) wymusza przezbrojenie przed pierwszym zadaniem. Ograniczenie (6.14) zapewnia wykonanie przezbrojenia między zadaniami. Ograniczenie (6.15) zapewnia odpowiednio, aby nie została przekroczona dozwolona ładowność każdej maszyny. Ponadto przez S^a oraz C^a oznaczono odpowiednio moment rozpoczęcia oraz zakończenia wykonywania operacji

specjalnej (zlanie mleka na bazie) na każdej maszynie a:

$$\forall a \in \mathcal{M} \quad S^a = C_{\pi_a(|\pi_a|)} + s_{\pi_a(|\pi_a|),0}, \tag{6.16}$$

$$\forall a \in \mathcal{M} \quad C^a = S^a + \sum_i p_i \quad i = \pi_a(1), \pi_a(2), \ldots, \pi_a(|\pi_a|), \tag{6.17}$$

$$\forall a, b \in \mathcal{M} \quad C^a \leqslant S^b \ \veebar \ C^b \leqslant S^a, \tag{6.18}$$

$$\forall a \in \mathcal{M} \quad C^a \leqslant 8h, \tag{6.19}$$

gdzie "$\veebar$" jest alternatywą rozłączną (*exclusive or*, XOR). Ograniczenie (6.16) gwarantuje przezbrojenie maszyny do stanu początkowego (powrót do magazynu). Ograniczenie (6.17) zapewnia, że rozładowanie wraz z wyczyszczeniem maszyny nie będzie przerywane. Ograniczenie (6.18) gwarantuje, że w dowolnym momencie może być wykonywania tylko jedna operacja specjalna, analogicznie do rozłącznych przezbrojeń w Rozdziale 5.2. Ponadto ograniczenie (6.19) wymusza 8-godzinny cykl pracy.

Podstawowym badanym kryterium optymalizacyjnym jest minimalizowanie czasu pracy dla wszystkich maszyn. Zatem celem będzie znalezienie takiego rozwiązania π, by:

$$\sum C^a(\boldsymbol{\pi}^{\text{opt}}) = \min_{\boldsymbol{\pi} \in \Pi} \sum C^a(\boldsymbol{\pi}), \tag{6.20}$$

gdzie Π jest zbiorem wszystkich dopuszczalnych permutacji zbioru. Rozwiązanie $\boldsymbol{\pi}^{\text{opt}}$ będzie nazywane rozwiązaniem optymalnym. Drugim kryterium optymalizacyjnym będzie minimalizowanie niewykorzystanej przestrzeni ładunkowej. Jeśli przez zbiór $\boldsymbol{\Omega} = \{\Omega^1, \Omega^2, \ldots, \Omega^m\}$ zostanie znaczony zbiór pojemności wszystkich maszyn, to dla każdej maszyny należy znaleźć minimalną możliwą pojemność:

$$\sum \Delta^a(\boldsymbol{\Omega}) = \min_{\boldsymbol{\Omega}} \sum_{a \in \mathcal{M}} |(\sum_i \omega_i) - \Omega^a| \tag{6.21}$$

gdzie przez $\sum \Delta^a(\boldsymbol{\Omega})$ oznaczono sumą niewykorzystanej pojemności wszystkich maszyn. Ponadto zbiór dopuszczalnych wartości Ω^a będzie określony docelowo przez użytkownika końcowego. W procesie optymalizacyjnym problem był rozpatrywany dwuetapowo. W pierwszej kolejności wybierano wstępnie odpowiedni (dopuszczalny) zestaw pojemności, a następnie minimalizowano czas pracy.

6.3.2 Aplikacja użytkownika

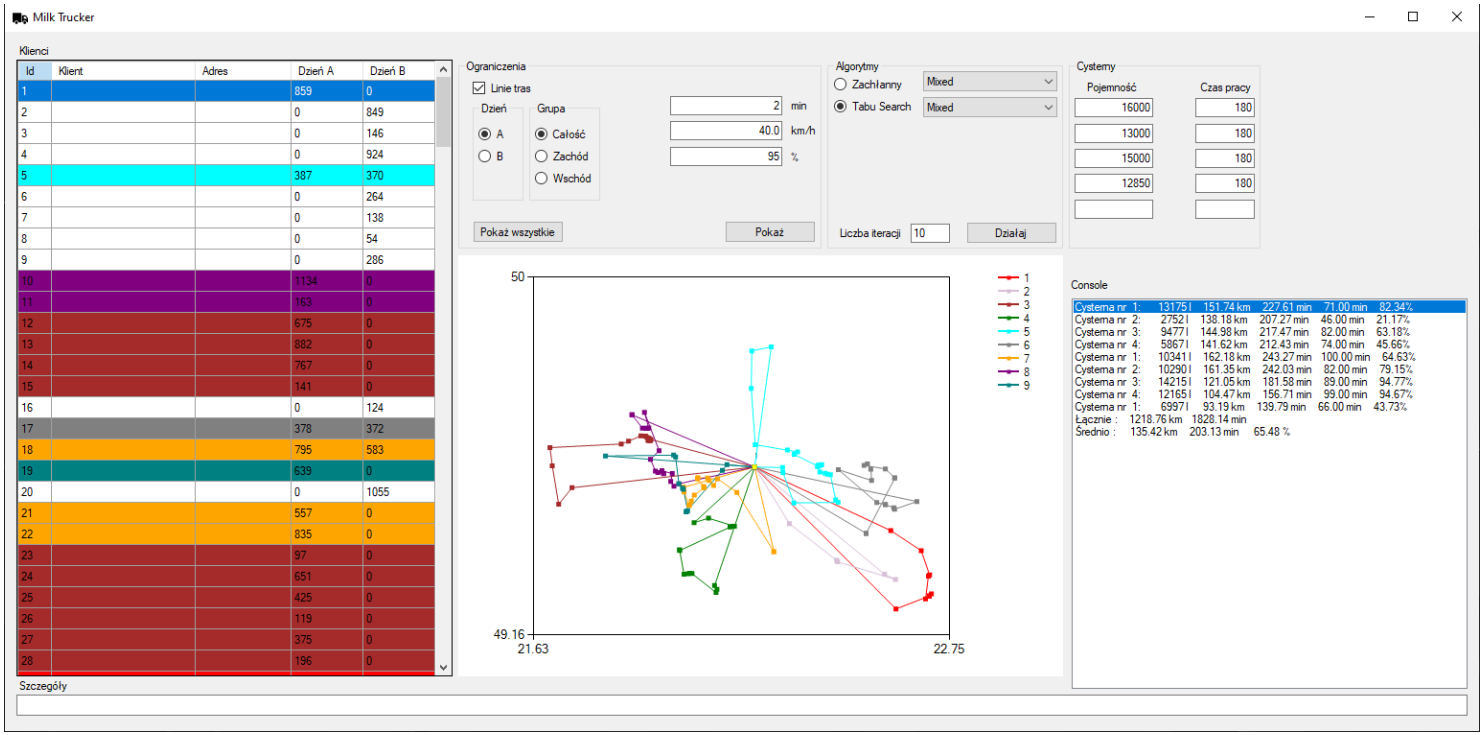
W porozumieniu z zewnętrzna firmą powstał prototyp aplikacji. Główny program został podzielony na dwa podprogramy. Pierwszy napisany w ję-

zyku programowania `PYTHON`, jest odpowiedzialny za stworzenie pełnej macierzy sąsiedztwa oraz bazy danych klientów na podstawie danych historycznych dostarczonych przez firmę. Funkcjonalność można było wydzielić, ponieważ baza klientów się nie zmienia. Wartości odbieranych ilości towaru od każdego klienta zostały uśrednione na podstawie danych z kilku miesięcy.

Drugi podprogram został zaimplementowany przy użyciu języka programowania C# oraz został wyposażony w graficzny interfejs użytkownika (GUI) zgodny z `.NET Framework 5.0`. Program pozwala na wprowadzenie puli dostępnych pojemności cystern. Dla wybranego losowo zestawu tworzy rozwiązanie dopuszczalne za pomocą algorytmu zachłannego (podobnego jak w Podrozdziale 6.2), które jest poprawiane przez algorytm Przeszukiwania z Zabronieniami. Następnie jest generowany nowy zestaw pojemności i cała procedura zostaje powtórzona. Program również został wyposażony w mapkę poglądową (patrz Rysunek 6.5).

6.3.3 Wnioski i uwagi

W rozdziale zaprezentowano przykład aplikacji metod szeregowania zadań do rozwiązywania rzeczywistego problemu optymalizacyjnego. Zaproponowano dwie metody rozwiązania z prostym wykorzystaniem dwóch funkcji celu. W dalszych etapach planowane jest uzupełnienie prototypu o sterowanie doborem zestawu pojemności z wykorzystaniem metod uczenia maszynowego (*Machine Learing*), w tym sztucznych sieci neuronowych (*Neural Networks*).



Rysunek 6.5: Widok na interfejs użytkownika systemu marszrutyzacji

Rozdział 7

Wnioski końcowe

Odnosząc się do postawionych tez pracy, na podstawie przeprowadzonych badań, można sformułować następujące wnioski. W rozprawie zaproponowano modele matematyczne oraz grafowe dla problemów przepływowych szeregowania zadań z uwzględnieniem sprzężeń czasowych, gdzie pojawiają się dodatkowe relacje między operacjami. Może zostać wymuszony minimalny lub maksymalny przestój między kolejnymi operacjami na tej samej maszynie. Dobierając odpowiednie wartości parametrów (minimalnego i maksymalnego przestoju) w proponowanych modelach problemów ze sprzężeniami czasowymi można rozważać klasyczne wersje problemów przepływowych z literatury, w szczególności standardowy wariant problemu bez dodatkowych ograniczeń lub z ciągła pracą maszyn. Uwzględniono zarówno wersję permutacyjną jak i niepermutacyjną problemu. W ramach sprzężeń czasowych można rozpatrywać również znane z literatury przezbrojenia, ponieważ dotyczą one operacji między różnymi zadaniami na tej samej maszynie. W pracy zaproponowano modele z przezbrojeniami na przykładzie problemu przepływowego szeregowania zadań. W pierwszej kolejności skupiono się na połączeniu przezbrojeń, zależnych od kolejności wykonywania zadań, z ograniczeniem bez przestrojów. Następnie zaproponowano modele uwzględniające rozłączne przezbrojenia, gdzie na raz może być wykonywane tylko jedno przezbrojenie ze względu na ograniczenia technologiczne. We wszystkich zaproponowanych modelach uwzględniono szereg dodatkowych ograniczeń, dzięki którym te modele lepiej opisują pracę rzeczywistych systemów produkcyjnych.

W części poświęconej analizie zaproponowanych problemów ze sprzężeniami czasowymi zidentyfikowano szereg nowych własności. Dla niepermutacyjnego problemu przepływowego szeregowania zadań zdefiniowano własność eliminacyjną bazującą na blokach zadań ze ścieżki krytycznej. Dla

problemu przepływowego szeregowania zadań z przezbrojeniami zależnymi od kolejności oraz ciągła pracą maszyn zdefiniowano nowy typ sąsiedztwa inspirowany problem znajdowania najkrótszej ścieżki Hamiltona w grafie. Następnie dla problemu szeregowania rozłącznych przezbrojeń w permutacyjnym problemie przepływowym szeregowania zadań również zidentyfikowano własność eliminacyjną bazującą na blokach. W rezultacie zidentyfikowanie powyższych własności wiąże się ze zmniejszeniem rozmiaru przestrzeni rozwiązań. Ponadto dla wszystkich opisanych problemów zaproponowano szybkie metody wyznaczania wartości funkcji celu, w czasie $O(nm)$ mimo dodania łuków powrotnych w modelach grafowych.

Po zaproponowaniu modelów oraz zidentyfikowaniu własności dla każdego omawianego problemu zaproponowano szereg metod optymalizacyjnych. Wśród nich można wymienić metody dokładne: model Całkowitoliczbowego Programowania Liniowego i Metodę Podziału i Ograniczeń. Poza metodami dokładnymi zaproponowano również wiele algorytmów przybliżonych: Przeszukiwanie Snopowe, Przeszukiwanie z Zabronieniami, Sztuczną Kolonię Pszczół, Algorytm Genetyczny oraz dedykowane algorytmy zachłanne. Najwięcej uwagi poświęcono algorytmowi Przeszukiwania z Zabronieniami, który to został wykorzystany do przeglądania dedykowanych otoczeń uwzględniających własności eliminacyjne. Zaprojektowane algorytmy zostały poddane badaniom eksperymentalnym na instancjach testowych, znanych z literatury lub wygenerowanych zgodnie z technikami opisanymi w literaturze. Wyniki badań eksperymentalnych pozwoliły określić relacje czasowe oraz jakościowe dla badanych algorytmów optymalizacyjnych.

Dla niepermutacyjnego problemu przepływowego szeregowania zadań ze sprzężeniami czasowymi zaproponowano równoległy algorytm liczenia wartości funkcji celu bazujący na schemacie algorytmu Floyda-Warshalla. Następnie przeprowadzono dyskusję teoretyczną na temat możliwego przyśpieszenia wykorzystując odpowiednie środowisko równoległe. Część prezentowanych w rozprawie rezultatów badań została opublikowana w czasopismach oraz materiałach konferencyjnych o zasięgu krajowym: Bożejko, Idzikowski, Wodecki [18], Bożejko, Idzikowski, Wodecki [19] i międzynarodowym: Idzikowski, Gnatowski, Rudy [63], Bożejko, Idzikowski, Rudy [22], Idzikowski [62], Bożejko, Idzikowski, Wodecki [21], Gnatowski, Rudy, Idzikowski [50]. Charakter oraz tematyka czasopism i konferencji lokuje przeprowadzone badania w dyscyplinie Automatyka, Elektronika i Elektrotechnika, ze względu na modelowanie typowych przypadków automatyki przemysłowej oraz sterowania procesami dyskretnymi. Do sterowania wykorzystano wiele metod najczęściej stosowanych w dyscyplinie Informatyka Techniczna i Telekomunikacja. Ponadto wśród zainteresowań autora są również

problemy szeregowania zadań z dwumaszynowymi gniazdami [13, 14] oraz tematyka wirtualnej rzeczywistości (*Virtual Reality*) [73].

Kierunki dalszych badań

Niniejsza praca, zadaniem autora, może stanowić inspiracje do dalszych badań dotyczących teoretycznych i praktycznych aspektów szeregowania zadań produkcyjnych oraz transportowych, w szczególności ze sprzężeniami czasowymi. Jednym z kierunków badań jest identyfikacja własności dla sprzężeń czasowych między operacjami tego samo zadania np.: bez czekania (*no wait*), z ograniczonym czekaniem (*limited wait*) itp. Rozważenie szczególnych przypadków dla sprzężeń czasowych dla operacji w ramach jednego zadania oraz operacjami z różnych zadań jednocześnie. Kolejnym kierunkiem badań jest przeniesienie sprzężeń czasowych oraz ich własności na inne problemy niż permutacyjny lub niepermutacyjny problem przepływowy szeregowania zadań, w szczególności na problem gniazdowy szeregowania zadań (JSSP), gdzie każde zadanie może mieć własną marszrutę technologiczną.

Kolejnym kierunkiem badań wynikającym wprost z rozważanych w rozprawie problemów jest kontynuacja badań nad zagadnieniem szeregowania rozłącznych przezbrojeń. Istotnym aspektem jest tu konstrukcja dedykowanych algorytmów optymalizacyjnych oraz identyfikacja własności dla podejścia dwupoziomowego, gdzie należy znaleźć optymalną kolejność wykonywania zadań dla górnego poziomu oraz korespondującą z nią optymalną kolejność wykonywania rozłącznych przezbrojeń. Istotnie jest również przeprowadzenie badań dla innych problemów szeregowania zadań oraz zagadnień w których może być wykonywanie maksymalnie k przezbrojeń w tym samym czasie.

Wszystkie badane problemy należały do klasy NP-trudnych, więc nie są znane do ich rozwiązywania algorytmy dokładne działające w czasie wielomianowym. Dlatego ostatnim, ale równie ważnym kierunkiem dalszych badań jest implementacja dedykowanych algorytmów w środowiskach równoległych. Istotne jest zaproponowanie algorytmów optymalizacyjnych dla całych problemów, ale równie istotna jest implementacja szybkich metod wyznaczania wartości funkcji celu, w szczególności dla algorytmów bazujących na przeglądaniu części lub całego sąsiedztwa. Co pozwoli na szybsze przeszukiwanie przestrzeni rozwiązań oraz w konsekwencji znajdowanie lepszych pod względem jakości rozwiązań.

Bibliografia

[1] Abadi, I. N. K., N. G. Hall, i C. Sriskandarajah: *Minimizing cycle time in a blocking flowshop.* Operations Research, 48(1):177–180, 2000.

[2] Aldowaisan, T. i A. Allahverdi: *New heuristics for m-machine no-wait flowshop to minimize total completion time.* Omega, 32(5):345–352, 2004.

[3] Allahverdi, A., J. N. Gupta, i T. Aldowaisan: *A review of scheduling research involving setup considerations.* Omega, 27(2):219–239, Kwiecień 1999, ISSN 0305-0483.

[4] Applegate, D. i W. Cook: *A Computational Study of the Job-Shop Scheduling Problem.* ORSA Journal on Computing, 3(2):149–156, Maj 1991, ISSN 0899-1499.

[5] Applegate, D. L., R. E. Bixby, V. Chvátal, W. Cook, D. G. Espinoza, M. Goycoolea, i K. Helsgaun: *Certification of an optimal TSP tour through 85,900 cities.* Operations Research Letters, 37(1):11–15, 2009.

[6] Back, T.: *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford university press, 1996.

[7] Bagchi, T. P., J. N. Gupta, i C. Sriskandarajah: *A review of TSP based approaches for flowshop scheduling.* European Journal of Operational Research, 169(3):816–854, 2006, ISSN 03772217.

[8] Balas, E.: *Project scheduling with resource constraints.* Rap., Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1968.

[9] Balas, E. i P. Toth: *Branch and bound methods for the traveling salesman problem.* 1983.

[10] Biruk, S. i P. Jaskowski: *SCHEDULING REPETITIVE CONSTRUCTION PROJECTS WITH WORK CONTINUITY CONSTRAINTS.* International Journal of Arts & Sciences, 5(4):107, 2012.

[11] Biskup, D., J. Herrmann, i J. N. D. Gupta: *Scheduling identical parallel machines to minimize total tardiness.* International Journal of Production Economics, 115(1):134–142, 2008, ISSN 0925-5273. https://www.sciencedirect.com/science/article/pii/S0925527308001515.

[12] Boyd, S. i J. Mattingley: *Branch and bound methods.* Notes for EE364b, Stanford University, str. 2006–2007, 2007.

[13] Bożejko, W., A. Gnatowski, R. Idzikowski, i M. Wodecki: *Algorytm wielomianowy dla cyklicznego problemu przydziału operacji w dwumaszynowym gnieździe.* W: *AUTOMATYZACJA PROCESÓW DYSKRETNYCH*, 2016.

[14] Bożejko, W., A. Gnatowski, R. Idzikowski, i M. Wodecki: *Cyclic flow shop scheduling problem with two-machine cells.* Archives of Control Sciences, 27(2):151–167, 2017, ISSN 23002611.

[15] Bożejko, W., A. Gnatowski, R. Klempous, M. Affenzeller, i A. Beham: *Cyclic scheduling of a robotic cell.* W: *2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*, str. 000379–000384. IEEE, Październik 2016, ISBN 978-1-5090-2645-6.

[16] Bożejko, W., A. Gnatowski, J. Pempera, i M. Wodecki: *Parallel tabu search for the cyclic job shop scheduling problem.* Computers & Industrial Engineering, 113:512–524, 2017.

[17] Bożejko, W., L. Gniewkowski, J. Pempera, i M. Wodecki: *Cyclic hybrid flow-shop scheduling problem with machine setups.* Procedia Computer Science, 29:2127–2136, 2014.

[18] Bożejko, W., R. Idzikowski, i M. Wodecki: *Problem przepływowy z przezbrojeniami oraz ciągłą pracą maszyn.* W: *Innowacje w Zarządzaniu i Inżynierii Produkcji*, 2016.

[19] Bożejko, W., R. Idzikowski, i M. Wodecki: *Problem przepływowy ze sprzężeniami czasowymi maszyn.* W: *Inżynieria Zarządzania Cyfryzacja Produkcji, Aktualności badawcze 1*, 2019.

[20] Bożejko, W., R. Idzikowski, i M. Wodecki: *Flow Shop Problem with Machine Time Couplings*, t. 987. 2020, ISBN 9783030195007.

[21] Bożejko, W., R. Idzikowski, i M. Wodecki: *Flow Shop Problem with Machine Time Couplings.* W: *Advances in Intelligent Systems and Computing*, t. 987, str. 80–89, 2020, ISBN 9783030195007.

[22] Bozejko, W., J. Rudy, i R. Idzikowski: *Parallel Computing for the Non-permutation Flow Shop Scheduling Problem with Time Couplings Using Floyd-Warshall Algorithm.* Performance Evaluation Models for Distributed Service Networks, str. 1–19, 2021.

[23] Brent, R. P.: *The parallel evaluation of general arithmetic expressions.* Journal of the ACM (JACM), 21(2):201–206, 1974.

[24] Brucker, P., B. Jurisch, i B. Sievers: *A branch and bound algorithm for the job-shop scheduling problem.* Discrete Applied Mathematics, 49(1-3):107–127, 1994, ISSN 0166218X.

[25] Brucker, P., S. Knust, i G. Wang: *Complexity results for flow-shop problems with a single server.* European Journal of Operational Research, 165(2):398–407, 2005, ISSN 03772217.

[26] Chakravarthy, K. i C. Rajendran: *A heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization.* Production Planning & Control, 10(7):707–714, 1999.

[27] Chen, B., C. N. Potts, i G. J. Woeginger: *A Review of Machine Scheduling: Complexity, Algorithms and Approximability.* W: Du, D. Z. i P. M. Pardalos (ed.): *Handbook of Combinatorial Optimization: Volume1–3*, str. 1493–1641. Springer US, Boston, MA, 1998, ISBN 978-1-4613-0303-9. `https://doi.org/10.1007/978-1-4613-0303-9_25`.

[28] Colorni, A., M. Dorigo, V. Maniezzo, i others: *An Investigation of some Properties of anAnt Algorithm".* W: *Ppsn*, t. 92, 1992.

[29] Cormen, T. H., C. E. Leiserson, R. L. Rivest, i S. Clifford: *Introduction to Algorithms.* The MIT Press, Cambridge, London, 3rd wyd., 2009, ISBN 978-0-262-03384-8.

[30] Cormen, T. H., C. E. Leiserson, R. L. Rivest, i C. Stein: *Introduction to algorithms*. MIT press, 2009.

[31] Dantzig, G. B.: *Application of the simplex method to a transportation problem*. Activity Analysis of Production and Allocation, 1951.

[32] Dantzig, G. B.: *Linear programming and extensions*, t. 48. Princeton university press, 1965.

[33] Dantzig, G. B. i J. H. Ramser: *The Truck Dispatching Problem*. Management Science, 6(1):80–91, 1959.

[34] Desrochers, M., J. Desrosiers, i M. Solomon: *A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows*. Operations Research, 40(2):342–354, 1992.

[35] Dorigo, M., M. Birattari, i T. Stutzle: *Ant colony optimization*. IEEE computational intelligence magazine, 1(4):28–39, 2006.

[36] Ebadi, Y. i N. Jafari Navimipour: *An energy-aware method for data replication in the cloud environments using a Tabu search and particle swarm optimization algorithm*. Concurrency and Computation: Practice and Experience, 31(1):e4757, Styczeń 2019, ISSN 1532-0626.

[37] Ellefsen, K. O., H. A. Lepikson, i J. C. Albiez: *Multiobjective Coverage Path Planning: Enabling Automated Inspection of Complex, Real-World Structures*. Applied Soft Computing Journal, 61:264–282, Styczeń 2019. http://arxiv.org/abs/1901.07272http://dx.doi.org/10.1016/j.asoc.2017.07.051.

[38] Engin, O. i A. Güçlü: *A new hybrid ant colony optimization algorithm for solving the no-wait flow shop scheduling problems*. Applied Soft Computing, 72:166–176, 2018.

[39] Fanjul-Peyro, L. i R. Ruiz: *Size-reduction heuristics for the unrelated parallel machines scheduling problem*. Computers & Operations Research, 38(1):301–309, 2011, ISSN 0305-0548. https://www.sciencedirect.com/science/article/pii/S0305054810001188.

[40] Fernandez-Viagas, V. i J. M. Framinan: *NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness*. Computers \& Operations Research, 60:27–36, 2015.

[41] Fortin, F. A., F. M. De Rainville, M. A. Gardner, M. Parizeau, i C. Gagné: *DEAP: Evolutionary Algorithms Made Easy.* Journal of Machine Learning Research, 13:2171–2175, 2012.

[42] Fortune, S. i J. Wyllie: *Parallelism in random access machines.* W: *Proceedings of the tenth annual ACM symposium on Theory of computing*, str. 114–118, 1978.

[43] Framinan, J. M., J. N. D. Gupta, i R. Leisten: *A review and classification of heuristics for permutation flow-shop scheduling with makespan objective.* Journal of the Operational Research Society, 55(12):1243–1255, 2004. https://doi.org/10.1057/palgrave.jors.2601784.

[44] Funke, B., T. Grünert, i S. Irnich: *Local Search for Vehicle Routing and Scheduling Problems: Review and Conceptual Integration.* J. Heuristics, 11:267–306, 2005.

[45] Garey, M. R., D. S. Johnson, i R. Sethi: *The Complexity of Flowshop and Jobshop Scheduling.* Mathematics of Operations Research, 1(2):117–129, 1976. https://doi.org/10.1287/moor.1.2.117.

[46] Glover, F.: *Tabu Search—Part I.* ORSA Journal on Computing, 1(3):190–206, Sierpień 1989, ISSN 0899-1499.

[47] Glover, F.: *Tabu Search—Part II.* ORSA Journal on Computing, 2(1):4–32, 1990, ISSN 0899-1499.

[48] Glover, F. i C. McMillan: *The general employee scheduling problem. An integration of MS and AI.* Computers and Operations Research, 13(5):563–573, 1986, ISSN 0305-0548.

[49] Gmys, J., M. Mezmaz, N. Melab, i D. Tuyttens: *A computationally efficient Branch-and-Bound algorithm for the permutation flowshop scheduling problem.* European Journal of Operational Research, 284(3):814–833, 2020, ISSN 03772217.

[50] Gnatowski, A., J. Rudy, i R. Idzikowski: *On two-machine Flow Shop Scheduling Problem with disjoint setups.* W: *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*, str. 277–282, 2020, ISBN 0000000310956.

[51] Grabowski, J.: *A New Algorithm of Solving the Flow—Shop Problem.* W: *Operations research in progress*, str. 57–75. Springer, 1982.

[52] Grabowski, J., E. Nowicki, i C. Smutnicki: *Block algorithm for scheduling of operations in job-shop system.* Przeglad Statystyczny, 35(1):67–80, 1988.

[53] Grabowski, J. i J. Pempera: *Zagadnienie przepływowe z ograniczeniami "bez magazynowania". Algorytm tabu search z multiruchami.* Automatyka / Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie, T. 9, z. 1:95–104, 2005, ISSN 1429-3447.

[54] Graham, R., E. L. Lawler, i A. Rinnooy Kan: *Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey.* W: Hammer, P. L., E. L. Johnson, i B. H. Korte (ed.): *Discrete Optimization II*, t. 5 w: *Annals of Discrete Mathematics*, str. 287–326. Elsevier, 1979.

[55] Gupta, S. K. i J. Kyparisis: *Single machine scheduling research.* Omega, 15(3):207–227, 1987, ISSN 0305-0483. http://www.sciencedirect.com/science/article/pii/0305048387900715.

[56] Harbaoui, H. i S. Khalfallah: *Tabu-search optimization approach for no-wait hybrid flow-shop scheduling with dedicated machines.* Procedia Computer Science, 176:706–712, 2020, ISSN 1877-0509.

[57] Harbaoui, H., S. Khalfallah, i O. Bellenguez-Morineau: *A case study of a hybrid flow shop with no-wait and limited idle time to minimize material waste.* W: *2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY)*, str. 207–212. IEEE, 2017.

[58] Hejazi, S. R. i S. Saghafian: *Flowshop-scheduling problems with makespan criterion: a review.* International Journal of Production Research, 43(14):2895–2929, 2005. https://doi.org/10.1080/00207540500056417.

[59] Holland, J.: *Genetic algorithms scientific American.* 1992.

[60] Holland, J. H. i others: *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992.

[61] Hoogeveen, H., M. Skutella, i G. J. Woeginger: *Preemptive scheduling with rejection.* Mathematical Programming, 94(2):361–374, 2003, ISSN 1436-4646. https://doi.org/10.1007/s10107-002-0324-z.

[62] Idzikowski, R.: *Capacitated Open Vehicle Routing Problem with Time Couplings.* W: Zamojski, W., J. Mazurkiewicz, J. Sugier, T. Walkowiak, i J. Kacprzyk (ed.): *Theory and Applications of Dependable Computer Systems*, str. 273–282, Cham, 2020. Springer International Publishing, ISBN 978-3-030-48256-5.

[63] Idzikowski, R., J. Rudy, i A. Gnatowski: *Solving Non-Permutation Flow Shop Scheduling Problem with Time Couplings.* Applied Sciences, 11(10):4425, 2021.

[64] Johnson, S. M.: *OPTIMAL TWO- AND THREE-STAGE PRODUCTION SCHEDULES WITH SETUP TIMES INCLUDED.* Naval Research Logistics Quarterly, 1:61–68, 1954.

[65] Kalczynski, P. J. i J. Kamburowski: *On the NEH heuristic for minimizing the makespan in permutation flow shops.* Omega, 35(1):53–60, 2007.

[66] Kara, I. i T. Bektas: *Integer linear programming formulations of multiple salesman problems and its variations.* European Journal of Operational Research, 174(3):1449–1458, 2006.

[67] Karaboga, D. i B. Akay: *A comparative study of artificial bee colony algorithm.* Applied mathematics and computation, 214(1):108–132, 2009.

[68] Karaboga, D. i B. Basturk: *On the performance of artificial bee colony (ABC) algorithm.* Applied soft computing, 8(1):687–697, 2008.

[69] Karlin, S.: *The structure of dynamic programing models.* Naval Research Logistics Quarterly, 2(4):285–294, 1955.

[70] Kennedy, J. i R. Eberhart: *Particle swarm optimization.* W: *Proceedings of ICNN'95-international conference on neural networks*, t. 4, str. 1942–1948. IEEE, 1995.

[71] Kirkpatrick, S., C. D. Gelatt, i M. P. Vecchi: *Optimization by simulated annealing.* science, 220(4598):671–680, 1983.

[72] Kizilate, G. i F. Nuriyeva: *On the nearest neighbor algorithms for the traveling salesman problem.* W: *Advances in Computational Science, Engineering and Information Technology*, str. 111–118. Springer, 2013.

[73] Klempous, R., K. Kluwak, R. Idzikowski, T. Nowobilski, i T. Zamojski: *Possibility analysis of danger factors visualization in the construction environment based on Virtual Reality model.* W: *8th IEEE International Conference on Cognitive Infocommunications, CogInfoCom 2017 - Proceedings*, t. 2018-Janua, 2017, ISBN 9781538612644.

[74] Kora, P. i P. Yadlapalli: *Crossover operators in genetic algorithms: A review.* International Journal of Computer Applications, 162(10), 2017.

[75] Koulamas, C.: *The single-machine total tardiness scheduling problem: Review and extensions.* European Journal of Operational Research, 202(1):1–7, 2010, ISSN 0377-2217. `http://www.sciencedirect.com/science/article/pii/S0377221709002641`.

[76] Krohling, R. A.: *Gaussian swarm: a novel particle swarm optimization algorithm.* W: *IEEE Conference on Cybernetics and Intelligent Systems, 2004.*, t. 1, str. 372–376. IEEE, 2004.

[77] Kumar, S., V. K. Sharma, i R. Kumari: *A novel hybrid crossover based artificial bee colony algorithm for optimization problem.* arXiv preprint arXiv:1407.5574, 2014.

[78] Laguna, M. i F. Glover: *Integrating target analysis and tabu search for improved scheduling systems.* Expert Systems with Applications, 6(3):287–297, 1993.

[79] Lai, X., J. K. Hao, i D. Yue: *Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem.* European Journal of Operational Research, 274(1):35–48, 2019, ISSN 0377-2217.

[80] Lalami, M. E. i D. El-Baz: *GPU implementation of the branch and bound method for knapsack problems.* W: *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, str. 1769–1777. IEEE, 2012.

[81] Lawler, E. L. i D. E. Wood: *Branch-and-bound methods: A survey.* Operations research, 14(4):699–719, 1966.

[82] Lee, T. i Y. Loong: *A review of scheduling problem and resolution methods in flexible flow shop.* International Journal of Industrial Engineering Computations, 10(1):67–88, 2019.

[83] Lim, A., B. Rodrigues, i C. Wang: *Two-machine flow shop problems with a single server.* Journal of Scheduling, 9(6):515–543, 2006, ISSN 10946136.

[84] Lin, Y. K. i F. Y. Hsieh: *Unrelated parallel machine scheduling with setup times and ready times.* International Journal of Production Research, 52(4):1200–1214, 2014. `https://doi.org/10.1080/00207543.2013.848305`.

[85] Lozano, M., F. Herrera, N. Krasnogor, i D. Molina: *Real-coded memetic algorithms with crossover hill-climbing.* Evolutionary computation, 12(3):273–302, 2004.

[86] Manne, A. S.: *On the Job-Shop Scheduling Problem.* Operations Research, 8(2):219–223, 1960. `https://doi.org/10.1287/opre.8.2.219`.

[87] Marcoulides, K. M.: *Latent growth curve model selection with Tabu search.* International Journal of Behavioral Development, str. 1–7, 2020, ISSN 14640651.

[88] Matsumoto, M. i T. Nishimura: *Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator.* ACM Transactions on Modeling and Computer Simulation, 8(1):3–30, Styczeń 1998, ISSN 10493301.

[89] Mirjalili, S., A. H. Gandomi, S. Z. Mirjalili, S. Saremi, H. Faris, i S. M. Mirjalili: *Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems.* Advances in Engineering Software, 114:163–191, 2017.

[90] Mirjalili, S., S. M. Mirjalili, i A. Lewis: *Grey wolf optimizer.* Advances in engineering software, 69:46–61, 2014.

[91] Moscato, P.: *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms.* Caltech concurrent computation program, C3P Report, (C3P 826), 1989.

[92] Moscato, P., C. Cotta, i A. Mendes: *Memetic algorithms.* W: *New optimization techniques in engineering*, str. 53–85. Springer, 2004.

[93] Nagata, Y.: *New EAX crossover for large TSP instances.* W: *Parallel Problem Solving from Nature-PPSN IX*, str. 372–381. Springer, 2006.

[94] Nawaz, M., E. E. Enscore, i I. Ham: *A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem.* Omega, 1983, ISSN 03050483.

[95] Nowicki, E. i C. Smutnicki: *A fast taboo search algorithm for the job shop problem.* Management Science, 42(6):797–813, Czerwiec 1996, ISSN 0025-1909.

[96] Nowicki, E. i C. Smutnicki: *An advanced tabu search algorithm for the job shop problem.* Journal of Scheduling, 8(2):145–159, 2005.

[97] Nowicki, E. i C. Smutnicki: *Some New Ideas in TS for Job Shop Scheduling.* W: Sharda, R., S. Voss, C. Rego, i B. Alidaee (ed.): *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*, str. 165–190. Springer US, Boston, MA, 2005, ISBN 978-0-387-23667-4. https://doi.org/10.1007/0-387-23667-8_7.

[98] Okubo, H., T. Miyamoto, S. Yoshida, K. Mori, S. Kitamura, i Y. Izui: *Project scheduling under partially renewable resources and resource consumption during setup operations.* Computers and Industrial Engineering, 83:91–99, 2015, ISSN 03608352.

[99] Osman, I. H. i C. N. Potts: *Simulated annealing for permutation flow-shop scheduling.* Omega, 17(6):551–557, 1989.

[100] Öztürk, C. i A. M. Örnek: *A MIP based heuristic for capacitated MRP systems.* Computers & Industrial Engineering, 63(4):926–942, 2012.

[101] Pan, Q. K. i R. Ruiz: *An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem.* Omega (United Kingdom), 44:41–50, 2014, ISSN 03050483.

[102] Pempera, J.: *An exact block algorithm for no-idle RPQ problem.* Archives of Control Sciences, 27(2), 2017.

[103] Pezzella, F., G. Morganti, i G. Ciaschetti: *A genetic algorithm for the flexible job-shop scheduling problem.* Computers & Operations Research, 35(10):3202–3212, 2008.

[104] Pham, D. T., A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, i M. Zaidi: *The bees algorithm.* Technical Note, Manufacturing Engineering Centre, Cardiff University, UK, 2005.

[105] Ram, D. J., T. H. Sreenivas, i K. G. Subramaniam: *Parallel simulated annealing algorithms.* Journal of parallel and distributed computing, 37(2):207–212, 1996.

[106] Reza Hejazi, S. i S. Saghafian: *Flowshop-scheduling problems with makespan criterion: a review.* International Journal of Production Research, 43(14):2895–2929, Lipiec 2005, ISSN 0020-7543.

[107] Ribaric, T. i S. Houghten: *Genetic programming for improved cryptanalysis of elliptic curve cryptosystems.* W: *2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings*, str. 419–426. Institute of Electrical and Electronics Engineers Inc., Lipiec 2017, ISBN 9781509046010.

[108] Ronconi, D. P. i L. R. S. Henriques: *Some heuristic algorithms for total tardiness minimization in a flowshop with blocking.* Omega, 37(2):272–281, 2009.

[109] Rossit, D. A., F. Tohmé, i M. Frutos: *The Non-Permutation Flow-Shop scheduling problem: A literature review.* Omega, 77:143–153, 2018, ISSN 0305-0483. `http://www.sciencedirect.com/science/article/pii/S0305048316307289`.

[110] Rossit, D. A., F. Tohmé, i M. Frutos: *The Non-Permutation Flow-Shop scheduling problem: A literature review.* Omega, 77:143–153, Czerwiec 2018, ISSN 03050483. `https://linkinghub.elsevier.com/retrieve/pii/S0305048316307289`.

[111] Ruiz, R. i T. Stützle: *An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives.* European Journal of Operational Research, 187(3):1143–1159, 2008, ISSN 03772217.

[112] Ruiz, R. i J. A. Vázquez-Rodríguez: *The hybrid flow shop scheduling problem.* European Journal of Operational Research, 205(1):1–18, 2010, ISSN 0377-2217. `http://www.sciencedirect.com/science/article/pii/S0377221709006390`.

[113] Sabuncuoglu, I. i M. Bayiz: *Job shop scheduling with beam search.* European Journal of Operational Research, 118(2):390–412, 1999.

[114] Sadollah, A., A. Bahreininejad, H. Eskandar, i M. Hamdi: *Mine blast algorithm: A new population based algorithm for solving constrained engineering optimization problems.* Applied Soft Computing, 13(5):2592–2612, 2013.

[115] Samarghandi, H. i T. Y. ElMekkawy: *An efficient hybrid algorithm for the two-machine no-wait flow shop problem with separable setup times and single server.* European Journal of Industrial Engineering, 5(2):111–131, 2011, ISSN 17515262.

[116] Sariklis, D. i S. Powell: *A heuristic method for the open vehicle routing problem.* Journal of the Operational Research Society, 51(5):564–573, 2000.

[117] Sawik, T. J.: *Scheduling flexible flow lines with no in-process buffers.* The International Journal of Production Research, 33(5):1357–1367, 1995.

[118] Serna, M. D. A. i C. A. S. Uran: *A memetic algorithm for the traveling salesman problem.* IEEE Latin America Transactions, 13(8):2674–2679, 2015.

[119] Sha, D. Y. i C. Y. Hsu: *A new particle swarm optimization for the open shop scheduling problem.* Computers & Operations Research, 35(10):3243–3261, 2008.

[120] Sharma, P. i A. Jain: *A review on job shop scheduling with setup times.* Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 230(3):517–533, 2016, ISSN 20412975.

[121] Smutnicki, C.: *Algorytmy szeregowania zadań.* Politechnika Wrocł awska, Wrocł aw, 2012.

[122] Song, Z. i N. Roussopoulos: *K-Nearest Neighbor Search for Moving Query Point.* W: Jensen, C. S., M. Schneider, B. Seeger, i V. J. Tsotras (ed.): *Advances in Spatial and Temporal Databases*, str. 79–96, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[123] Soukhal, A., A. Oulamara, i P. Martineau: *Complexity of flow shop scheduling problems with transportation constraints.* European Journal of Operational Research, 161(1):32–41, 2005.

[124] Stafford, E. F.: *On the development of a mixed-integer linear programming model for the flowshop sequencing problem.* Journal of the Operational Research Society, 39(12):1163–1174, 1988.

[125] Stanley, R. P.: *Catalan addendum.* Rap., MIT, 2013. `http://www-math.mit.edu/~rstan/ec/catadd.pdf`.

[126] Stanley, R. P. i S. Fomin: *Enumerative Combinatorics*. Cambridge University Press, Styczeń 1999, ISBN 9780521560696.

[127] Steinbiss, V., B. H. Tran, i H. Ney: *Improvements in beam search*. W: *Third International Conference on Spoken Language Processing*, 1994.

[128] Su, L. H. i Y. Y. Lee: *The two-machine flowshop no-wait scheduling problem with a single server to minimize the total completion time*. Computers and Operations Research, 35(9):2952–2963, 2008, ISSN 03050548.

[129] Taillard, E.: *Benchmarks for basic scheduling problems*. European Journal of Operational Research, 64(2):278–285, Styczeń 1993, ISSN 0377-2217.

[130] Tang, L., H. Xuan, i J. Liu: *A new Lagrangian relaxation algorithm for hybrid flowshop scheduling to minimize total weighted completion time*. Computers & Operations Research, 33(11):3344–3359, 2006.

[131] Tang, R., S. Fong, X. S. Yang, i S. Deb: *Wolf search algorithm with ephemeral memory*. W: *Seventh international conference on digital information management (ICDIM 2012)*, str. 165–172. IEEE, 2012.

[132] Tawarmalani, M. i N. V. Sahinidis: *A polyhedral branch-and-cut approach to global optimization*. Mathematical programming, 103(2):225–249, 2005.

[133] Tempelmeier, H. i L. Buschkühl: *Dynamic multi-machine lotsizing and sequencing with simultaneous scheduling of a common setup resource*. International Journal of Production Economics, 113(1):401–412, 2008, ISSN 09255273.

[134] Toth, P. i D. Vigo: *Branch-and-bound algorithms for the capacitated VRP*. W: *The vehicle routing problem*, str. 29–51. SIAM, 2002.

[135] Toth, P. i D. Vigo: *Models, relaxations and exact approaches for the capacitated vehicle routing problem*. Discrete Applied Mathematics, 123(1):487–512, 2002, ISSN 0166-218X.

[136] Vilcot, G. i J. C. Billaut: *A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem*. European Journal of Operational Research, 190(2):398–411, 2008.

[137] Vlk, M., A. Novak, i Z. Hanzalek: *Makespan Minimization with Sequence-dependent Non-overlapping Setups.* W: *Proceedings of the 8th International Conference on Operations Research and Enterprise Systems*, str. 91–101. SCITEPRESS - Science and Technology Publications, 2019, ISBN 978-989-758-352-0.

[138] Whitley, D., S. Rana, i R. B. Heckendorn: *The island model genetic algorithm: On separability, population size and convergence.* Journal of computing and information technology, 7(1):33–47, 1999.

[139] Woodcock, A. J. i J. M. Wilson: *A hybrid tabu search/branch & bound approach to solving the generalized assignment problem.* European journal of operational research, 207(2):566–578, 2010.

[140] Woodruff, D. L. i M. L. Spearman: *Sequencing and batching for two classes of jobs with deadlines and setup times.* Production and Operations Management, 1(1):87–102, 1992.

[141] Wu, X. i A. Che: *Energy-efficient no-wait permutation flow shop scheduling by adaptive multi-objective variable neighborhood search.* Omega, 94:102117, 2020.

[142] Xiao, Y. Y., R. Q. Zhang, Q. H. Zhao, i I. Kaku: *Permutation flow shop scheduling with order acceptance and weighted tardiness.* Applied Mathematics and Computation, 218(15):7911–7926, 2012.

[143] Yang, H., Y. Ye, i J. Zhang: *An approximation algorithm for scheduling two parallel machines with capacity constraints.* Discrete Applied Mathematics, 130(3):449–467, 2003, ISSN 0166-218X. `https://www.sciencedirect.com/science/article/pii/S0166218X02006017`.

[144] Yang, X. S. i X. He: *Bat algorithm: literature review and applications.* International Journal of Bio-inspired computation, 5(3):141–149, 2013.

[145] Zhou, Y., H. Chen, i G. Zhou: *Invasive weed optimization algorithm for optimization no-idle flow shop scheduling problem.* Neurocomputing, 137:285–292, 2014.

[146] Zhu, X. i W. E. Wilhelm: *Scheduling and lot sizing with sequence-dependent setup: A literature review.* IIE Transactions (Institute of Industrial Engineers), 38(11):987–1007, 2006, ISSN 0740817X.

Spis tablic

2.1 Instancja problemu PFSSP o rozmiarze $n = 5$ na $m = 3$ . . 26
2.2 Macierze czasów przezbrojeń między operacjami 30

4.1 Instancja problemu PFSSP-TC z Przykładu 4.1 57
4.2 Średnie czasy działania T (metod B&B, BS, TS oraz ABC, w sekundach) oraz średnie PRD (metod BS, TS oraz ABC) dla instancji ze zbioru A 67
4.3 Średnie PRD oraz czas działania T (w sekundach) dla metod przybliżonych BS, TS i ABC 69
4.4 Instancja problemu FSSP-TC z Przykładu 4.2 71
4.5 Reprezentacja permutacji $\boldsymbol{\pi}$ w pamięci komputera 75
4.6 Reprezentacja permutacji odwrotnej $\boldsymbol{\pi}^{-1}$ w pamięci komputera 75
4.7 Średnie czasy działania T (metod B&B oraz TS, w sekundach) oraz średnie PRD (metody TS) dla instancji ze zbioru A 84
4.8 Średnie PRD oraz czas działania T (w sekundach) dla metody TS z sąsiedztwem zamień, zamień sąsiedni oraz zamień blokowy dla instancji ze zbioru B 85
4.9 Teoretyczne przyspieszenia proponowanej metody równoległej w porównaniu z podejściem sekwencyjnym 91

5.1 Czasy wykonywania operacji dla problemu NPFSSP-S z Przykładu 5.1 98
5.2 Czasy wykonywania przezbrojeń dla problemu NPFSSP-S z Przykładu 5.1 98
5.3 Średnie PRD oraz czas działania T dla algorytmów TS oraz metody GA 107
5.4 Instancja problemu z Przykładu 5.3 dla $n = 4$ oraz $m = 3$. 112
5.5 Wydajność algorytmu heurystycznego w różnych grupach instancji 130

5.6 Średnie PRD oraz czas działania T dla algorytmu TS w dwóch wersjach oraz sformułowania MILP 133

6.1 Instancja stworzona na podstawie rzeczywistych danych. . . 140
6.2 Instancja problemu COVRP z Przykładu 6.1. 143
6.3 Średnie PRD oraz czasy działania metod TS oraz algorytmu zachłannego 149

Spis rysunków

2.1 Graf $\mathcal{G}(\pi)$ dla problemu PFSSP dla permutacji π 26
2.2 Harmonogram dla instancji PFSSP z Przykładu 2.1 27
2.3 Przykładowy graf $\mathcal{G}(\boldsymbol{\pi})$ dla problemu FSSP 27
2.4 Harmonogram dla instancji FSSP z przykładu 2.2 28
2.5 Klasyfikacja przezbrojeń 29
2.6 Graf $\mathcal{G}(\pi)$ dla problemu PFSSP z przezbrojeniami typu ST_{sd} 31
2.7 Harmonogram dla PFSSP z przezbrojeniami z przykładu 2.3 31
2.8 Graf $\mathcal{G}(\pi)$ dla problemu PFSSP bez przestojów 33
2.9 Harmonogram dla instancji PFSSP bez przestojów 33
2.10 graf $\mathcal{G}(\pi)$ dla problemu PFSSP bez czekania 34
2.11 Harmonogram dla instancji PFSSP bez czekania 34
2.12 Ścieżka krytyczna dla klasycznego PFSSP z Przykładu 2.1 . 37

3.1 Klasyfikacja algorytmów 40
3.2 Przykładowe schematy połączeń między wyspami 50

4.1 Harmonogram dla rozwiązania π z Przekładu 4.1 58
4.2 Przykładowy graf $\mathcal{G}(\pi)$ dla problemu PFSSP-TC 60
4.3 Harmonogram dla rozwiązania $\boldsymbol{\pi}$ z Przekładu 4.2 71
4.4 Przykładowy graf $\mathcal{G}(\boldsymbol{\pi})$ dla problemu FSSP-TC 73
4.5 Ogólna struktura L- oraz R-bloku 77
4.6 Przykład transformacji grafu rozwiązania $\mathcal{G}(\boldsymbol{\pi})$ na graf $\mathcal{G}^*(\boldsymbol{\pi})$ oraz początkowe wartości macierzy A 88

5.1 Harmonogram dla rozwiązania π z Przekładu 5.1 99
5.2 Przykładowy graf $\mathcal{G}(\pi)$ dla problemu NPFSSP-S 100
5.3 Dosunięty w lewo harmonogram dla instancji z Przykładu 5.3 oraz $\boldsymbol{\sigma} = (1, 1, 1, 2, 3, 2, 3, 2, 3)$ 112
5.4 Zależność między liczbą dopuszczalnych σ oraz liczbami Catalana 114

5.5 Ilustracja do dowodu Twierdzenia 5.5. Pełne łuki przedstawiają ograniczenia technologiczne i maszynowe. Łuki przerywaną linią reprezentują kolejność wykonywania przezbrojeń (czerwone dla $\boldsymbol{\sigma}$ oraz niebieskie dla $\hat{\boldsymbol{\sigma}}$). Ciemnoszare wierzchołki reprezentują operacje, a jasnoszare wierzchołki przedstawiają przezbrojenia 117
5.6 Czas działania metody bazującej na MILP dla różnych rozmiarów i grup instancji 131
5.7 Czas działania alg. heurystycznego dla różnych rozmiarów i grup instancji 132

6.1 139
6.2 Harmonogram dla rozwiązania π z Przekładu 6.1. 143
6.3 Reprezentacja wielkiej trasy 145
6.4 Widok na interfejs użytkownika systemu marszrutyzacji . . 148
6.5 Widok na interfejs użytkownika systemu marszrutyzacji . . 154

Spis algorytmów

1 Metoda Podziału i Ograniczeń 41
2 Programowanie Dynamiczne 42
3 NEH 44
4 Przeszukiwanie Lokalne 45
5 Przeszukiwanie z zabronieniami 46
6 Symulowane Wyżarzanie 48
7 Algorytm Populacyjny 49

8 Konstruowanie harmonogramu dosuniętego w lewo 62
9 Sztuczna Kolonia Pszczół 66
10 Konstruowanie harmonogramu dosuniętego w lewo 74
11 Wyznaczanie bloków 80

12 Konstruowanie harmonogramu dosuniętego w lewo 102
13 Procedura udoskonalenia 121
14 Algorytm zachłanny dla dwóch maszyn 124
15 Algorytm zachłanny dla wielu maszyn 125
16 Przeszukiwanie z Zabronieniami 126
17 Ocena sąsiedztwa 127

18 Algorytm zachłanny 146

Indeks

Algorytm
  Ewolucyjny, 49
  Genetyczny, 49
  Memetyczny, 51, 121
  Populacyjny, 49
  Wyspowy, 50
algorytm
  2-opt, 51
  Floyda-Warshalla, 86
  Najbliższego Sąsiada, 44, 145
  NEH, 44, 81
  rojowy, 51
  równoległy, 156
  sztucznej koloni pszczół, 51
  wspinaczkowy, 51
  zachłanny, 123, 145

bez
  czekania, 34, 95, 157
  przestojów, 32, 58, 71, 98
blok, 76, 116
  lewostronny, 76
  prawostronny, 76
  wewnętrzny, 76

CUDA, 92
cykl, 60, 73, 86, 100
czas przestoju
  maksymalny, 55, 57, 58, 70, 71, 98
  minimalny, 55, 57, 58, 70, 71, 98

dolne ograniczenie, 41
dopuszczalność, 113
długość uszeregowania, 109

funkcja
  akceptacji, 48
  celu, 35, 63
  kryterialna, 22, 35, 75, 101

generator
  Mersenne Twister, 129
  Taillarda, 67, 83, 132
GTR, 144
GUI, 147

harmonogram, 24, 57, 70, 97, 109, 142, 151

instancja, 67, 83
inteligencja
  roju, 51
  rozproszona, 51

jakość, 43, 67, 83

kadencja, 83
kolejność
  wykonywania
    przezbrojeń, 110
    zadań, 56, 70, 96, 112, 142, 151
kryterium
  aspiracji, 47
krzyżowanie, 50

liczby
  Catalana, 113
lista
  tabu, 47
  zakazów, 47, 83

makespan, 36
maszyna, 21
  równoległa, 24
Metoda
  Podziału i Ograniczeń, 39, 63, 81
metoda
  dokładna, 39, 81, 156
  przybliżona, 43, 156
  udoskonalenia, 119
  zachłanna, 44
MILP, 122
model
  grafowy, 25, 27, 32, 34, 58, 59, 72, 99, 155
  matematyczny, 56, 69, 96, 108, 138, 141, 155
  PRAM, 85
  RAM, 85
multistart, 47
mutacja, 50

notacja
  Grahama, 22, 35, 58, 71, 98

odchylenie, 67, 83
ograniczenie
  czasowe, 144
  dolne, 64, 81
  górne, 64, 81
  pojemności, 142
operacja, 21, 28
operatory
  genetyczne, 50
otoczenie, 45, 47, 48, 76, 127
  blokowe, 46
  wstaw, 46
  zamień, 45
    sąsiedni, 45

pamięć
  długoterminowa, 128
  krótkoterminowa, 128
permutacja
  odwrotna, 75
  optymalna, 71
pokolenie, 49
populacja, 49, 65
  początkowa, 49
problem
  dwupoziomowy, 112, 157
  gniazdowy, 23, 157
  jednomaszynowy, 23, 42
  komiwojażera, 42
  marszrutyzacji
    pojazdów, 141
    z ograniczoną pojemnością pojazdów, 141
  marszrutyzacji pojazdów, 42
  ogólny, 23
  otwarty, 23
  plecakowy, 42
  przepływowy, 23, 24, 36, 41, 55
    niepermutacyjny, 26, 69
    permutacyjny, 25, 56, 96, 108, 138
  równoległy, 23
programowanie
  dynamiczne, 42
  liniowe, 43
    całkowitoliczbowe, 43
przestrzeń
  rozwiązań, 113, 116
Przeszukiwanie
  Snopowe, 64
  z Zabroenieniami, 145
  z Zabronieniami, 65, 82, 104, 125

przeszukiwanie
lokalne, 45
z zabronieniami, 46
przezbrojenia, 28, 95, 97, 98, 141, 157
niezależne od kolejności, 98
rozłączne, 95, 109, 138
zależne od kolejności, 31

restart, 47, 128
rozrzut, 83
rozwiązanie
częściowe, 81
dopuszczalne, 110
niedopuszczalne, 145
optymalne, 71
ruch, 65, 75, 82, 104
wstaw, 127
zamień, 82
blokowy, 82
sąsiedni, 82
złożony, 145
ruletka, 49

schemat
chłodzenia, 48
selekcja, 49
skok powrotny, 47, 128
solver, 129, 132
Sortowanie
Bąbelkowe, 120
Szybkie, 44
sprzężenia czasowe, 28, 55, 56, 69, 95, 98, 157
Symulowane Wyżarzanie, 47
sztuczna
kolonia pszczół, 65
sąsiedztwo, 45, 66, 76, 82

turniej, 49
typy
funkcji celu, 35
maszyn, 24
ograniczeń, 28
problemów, 23
przepływowych, 24
przezbrojeń, 29

wyspa, 50
węzeł, 64, 81
własność, 155
blokowa, 46, 47, 76, 78, 155
eliminacyjna, 114, 116

zadanie, 21
zbiór
maszyn, 21, 56, 69, 96, 108, 138, 141
operacji, 21, 56, 69, 96, 108, 138, 141
zadań, 21, 56, 69, 96, 108, 138, 141

ścieżka, 61, 74, 86, 101
Hamiltona, 103
kratowa, 114
krytyczna, 36