Politechnika Wrocławska

INSTYTUT SZTUCZNEJ INTELIGENCJI
WYDZIAŁ INFORMATYKI I ZARZĄDZANIA
WROCŁAW, WYBRZEŻE WYSPIAŃSKIEGO 27

ROZPRAWA DOKTORSKA

Nowy szablon z kodowaniem nieporządnym jako remedium na typowe wady algorytmu genetycznego

Michał Przewoźniczek

promotor: prof. dr hab. inż. Halina Kwaśnicka

Wrocław, 21. maja 2012

Wstęp ... 3
1. Problem praktyczny – projektowanie przepływu bez rozgałęzień w szkieletowych sieciach komputerowych zorientowanych połączeniowo ... 7
1.1 Przyjęty model sieci szkieletowej i przepływu w sieci ... 8
1.2 Funkcja LFL ... 11
1.3 Sieci testowe i wymagania dla przepływu ... 13
2. Metody poszukiwania rozwiązań dla problemu projektowania przepływu wieloskładnikowego bez rozgałęzień ... 15
2.1 Metody oparte o metodę FD ... 15
2.2 Metody oparte o Relaksację Lagrange'a ... 16
2.3 Metody bazujące na podejściu ewolucyjnym ... 16
2.4 HEFAN (Hierarchical Evolutionary algorithm for Flow Assigment in Non-bifurcated commodity flow) ... 17
2.4.1 Ogólna charakterystyka algorytmu HEFAN ... 17
2.4.2 Reprezentacja osobników na poziomie „wysokim" i „niskim" ... 20
2.4.3 Baza tras ... 21
2.4.4 „Wysoki" poziom algorytmu ... 21
2.4.5 „Niski" poziom algorytmu ... 24
2.4.6 Określenie cech rozwiązywanego problemu dla kodowania użytego w algorytmie HEFAN ... 29
2.4.7 Wnioski z działania algorytmu HEFAN – motywacje dla dalszych prac ... 32
3. MuPPetS – propozycja nowego szablonu pracy dla algorytmów bazujących na idei algorytmu genetycznego ... 35
3.1 Wprowadzenie ... 35
3.2 Problemy testowe, funkcje zwodnicze ... 36
3.3 Algorytmy Messy i Fast Messy GA - opis ... 38
3.3.1 Algorytm mGA ... 39
3.3.2 Algorytm fmGA ... 41
3.3.3 Wady algorytmów mGA i fmGA ... 42
3.4 Algorytm BOA - opis ... 44
3.5 Algorytm MuPPetS – opis ... 45
3.5.1 Struktura danych algorytmu MuPPetS ... 45
3.5.2 Wzorce genów – główna idea algorytmu MuPPetS ... 47
3.5.2.1 Metody pozyskiwania wzorców genów ... 47
3.5.2.2 Wzorce genów – dlaczego są pożyteczne ... 50
3.5.3 Fazy algorytmu MuPPetS ... 51
3.5.4 Algorytm MuPPetS – dlaczego i jak działa ... 54
3.6 Klasyfikacja algorytmów fmGA, BOA i MuPPetS jako algorytmów typu „linkage learning" ... 57
3.7 Wyniki eksperymentów ... 57
3.7.1 Różnice pomiędzy wynikami uzyskanymi dla algorytmów BOA, fmGA i MuPPetS 58
3.7.2 Kodowanie, dostrajanie, procedura testowa ... 59
3.7.3 Wyniki i komentarze ... 62
3.7.3.1 Złożoność obliczeniowa dla algorytmów BOA i MuPPetS ... 63
3.7.3.2 Złożoność obliczeniowa i liczba wyliczeń funkcji przystosowania. ... 66
3.7.3.3 Jakość wyników zaproponowanych przez poszczególne algorytmy. ... 70
3.7.3.4 Wyniki – podsumowanie ... 73
3.7.3.5 Efekt kierunkowania uwagi ... 75
3.7.3.6 MuPPetS - autoadaptacja i globalna mutacja ... 76

3.8 MuPPetS - Podsumowanie 76
4. MuPPetS-FuN – propozycja nowego algorytmu do rozwiązywania problemu projektowania przepływu w szkieletowej sieci komputerowej bez rozgałęzień, opartego o schemat działania algorytmu MuPPetS oraz algorytm HEFAN 78
4.1 Budowa algorytmu MuPPetS-FuN 78
4.2 Wstępne testy algorytmu MuPPetS-FuN 80
4.3 MuPPetS-FuN Active 86
5. Porównanie efektywności algorytmu MuPPetS-FuN Active z innymi algorytmami 89
5.1 Porównanie efektywności algorytmu MuPPetS-FuN Active i algorytmu LRH 89
5.2 Porównanie efektywności algorytmu MuPPetS-FuN Active i algorytmu HEFAN 2.2 90
6. Podsumowanie 92
Bibliografia 96
Załącznik A - wyniki 104

Wstęp

Za początek badań nad algorytmami genetycznymi (ang. *Genetic Algorithm*) można przyjąć opublikowane pod koniec lat 50-tych XX wieku, prace [7, 25, 26, 27]. Biolodzy Barricelli i Fraser prezentowali w nich symulację procesów genetycznych przy pomocy komputerów. Celem prac nie było zaproponowanie nowego narzędzia obliczeniowego, ale symulacje były bliskie współczesnemu pojmowaniu algorytmu genetycznego. Kilkanaście lat później J.H. Holland opublikował kluczową dla rozwoju algorytmów genetycznych pracę prezentującą teorię schematów [41]. Od tego czasu prowadzono liczne badania, jednym z głównych pionierów był m.in. David Goldberg [11, 15, 34, 35, 36]. Do dziś, algorytmy genetyczne znalazły setki nowych zastosowań. Bogatszy opis genezy algorytmów genetycznych i ich działania można też znaleźć w [65, 89].

Jednym z głównych kierunków zastosowań dla algorytmów genetycznych jest poszukiwanie rozwiązań dla problemów NP-zupełnych. Przykładem takiego problemu jest leżący u podstaw rozważań przedstawionych w niniejszej pracy problem praktyczny – problem projektowania przepływu w szkieletowych sieciach komputerowych zorientowanych połączeniowo [76, 77, 78]. Dla zastosowań praktycznych ważne jest aby propozycje rozwiązań dla danego problemu były jak najwyższej jakości. W przypadku metod bazujących na idei algorytmu genetycznego, jednymi z ważniejszych powodów ograniczenia efektywności są przedwczesna zbieżność (ang. *preconvergence*) spowodowana niemożnością opuszczenia zajętego przez populację obszaru minimum lokalnego, oraz spadek wydajności operatorów genetycznych, takich jak krzyżowanie, spowodowany długim kodowaniem. Niniejsza praca zawiera analizę i wyszczególnienie cech charakterystycznych dla wybranego problemu praktycznego, oraz propozycję nowego szablonu pracy dla metod bazujących na idei ewolucji (MuPPetS [57]), pomniejszającą negatywny wpływ tych cech.

Zwykle autorzy konkretnej metody bazującej na idei algorytmu genetycznego wkładają wiele wysiłku, aby zwiększyć efektywność proponowanej metody (np. przeprowadzając szeroko zakrojone dostrajanie parametrów, dodając specjalizowane operatory lub w inny sposób wprowadzając mechanizmy biorące pod uwagę specyficzne cechy rozwiązywanego problemu). Jednak bez względu na to, jak różne mogą być problemy i proponowane dla ich rozwiązania metody, zwykle można wskazać ich jedną, wspólną cechę:

algorytm bazujący na idei algorytmu genetycznego może być użyteczny w zastosowaniach praktycznych wtedy, gdy potrafi proponować dobrej jakości rozwiązania dla problemów o cechach podobnych do „problemów laboratoryjnych", ale o znacznie większej skali.

Na przykład: dla problemu komiwojażera (ang. *TSP – Travelling Salesman Problem*) [28, 45, 62, 64, 93, 113] , polegającego na minimalizacji kosztu trasy pomiędzy zadaną listą miast, można wygenerować bardzo efektywny (należy zauważyć, że w zależności od potrzeb użytkownika termin „efektywny" może mieć różne znaczenie, np: „szybki i proponujący średniej jakości rozwiązania", „niekoniecznie szybki, ale proponujący rozwiązania wysokiej jakości", etc.) algorytm wyznaczający trasy przejazdu dla nie więcej niż 10 miast (przy założeniu że między każdą możliwą parą miast istnieje dokładnie jedno połączenie, przestrzeń rozwiązań dla 10 miast to: 10! = 3 628 800 przypadków). Najprawdopodobniej algorytm taki będzie miał znikomą wartość praktyczną, ponieważ:

1. Obecny stan techniki pozwala na znalezienie optymalnego rozwiązania za pomocą przeglądu zupełnego dla wymienionej powyżej przestrzeni rozwiązań.

2. Dla większości ewentualnych użytkowników (np. firm kurierskich) takiego algorytmu istotne będzie nie to, czy algorytm jest efektywny przy poszukiwaniu rozwiązań dla 10 miast, ale dla 100, 1000, lub więcej miast.

Istotnym problemem jaki można napotkać w literaturze [19, 34, 35, 36, 39, 40, 82, 85, 86] zajmującej się algorytmami bazującymi na idei algorytmu genetycznego jest nikła wartość praktyczna proponowanych metod, które są testowane jedynie na „laboratoryjnych" wersjach problemów (patrz także: 2.3).

Duża skala problemu (w przytoczonym przykładzie problemu komiwojażera będzie to duża liczba miast), implikuje zwiększenie liczby genów niezbędnych do zakodowania problemu. Natomiast duża liczba genów, implikuje znaczący spadek efektywności algorytmu genetycznego. Autor sam zetknął się z tym zjawiskiem w pracach dotyczących problemu projektowania przepływu w szkieletowych sieciach komputerowych [76, 77, 78]. Na przykład niektóre, występujące w praktyce, problemy projektowania przepływu posiadają przestrzeń rozwiązań przekraczającą 10^{2500} przypadków i wymagają dużej liczby genów do zakodowania pełnego rozwiązania (np. niektóre przypadki opisane w pracy [78] wymagały kodowania przy użyciu 2500 genów). Użycie standardowego algorytmu genetycznego do rozwiązania problemów takiej skali wydaje się (i zwykle jest) nieefektywne. Z tego powodu wielu autorów próbuje ominąć problem długiego kodowania przy użyciu różnego rodzaju wybiegów. Na przykład zamiast bezpośrednio kodować rozwiązanie można zakodować reguły jego tworzenia, lub innego rodzaju dane wejściowe dla deterministycznego algorytmu o niskiej złożoności obliczeniowej [87, 90]. Wyewoluowane za pomocą algorytmu genetycznego reguły są potem używane do wygenerowania pełnego rozwiązania problemu i oceny osobnika. Zwykle tego typu wybiegi ograniczają jednak przestrzeń rozwiązań. Może to istotnie wpłynąć na obniżenie efektywności całego algorytmu, ponieważ dobre jakościowo rozwiązania mogą znaleźć się poza przestrzenią poszukiwań [58, 59].

Nieporządne algorytmy genetyczne (ang. *mGA - Messy Genetic Algorithm*, do których zdaniem autora niniejszej pracy, w języku polskim znacznie lepiej pasowałaby nazwa „bałaganiarskie", która była wyraźną intencją Davida Goldberga) mogą zostać uznane za próbę zmierzenia się z problemem długiego kodowania [34, 35, 36]. Nie ma wątpliwości, że przetwarzanie jedynie części zamiast pełnego rozwiązania było jedną z głównych idei, która legła u podstaw mGA. Goldberg zaproponował koncepcję bloków budujących (ang. *building blocks*), które są „dobrymi" częściami pełnego rozwiązania. Zgodnie z tą koncepcją odnajdywanie i łączenie ze sobą bloków budujących ma prowadzić do znalezienia dobrego rozwiązania dla całego problemu. Niestety, propozycje Goldberga zawierają istotne wady. Na przykład w zaproponowanym przez Goldberga szablonie działania dla algorytmu mGA jedna z dwóch głównych faz algorytmu jest bliska przeglądowi zupełnemu. Kolejną wadą mGA jest niewielka możliwość zastosowania algorytmu do rozwiązywania problemów, które nie są zbudowane z podproblemów posiadających taki sam, lub bardzo zbliżony do siebie wpływ na ogólną wartość pełnego rozwiązania problemu. Wreszcie kolejną wadą jest fakt, że mGA nie jest efektywne jeśli poszczególne podproblemy są w jakiś sposób od siebie zależne. Reasumując dwie ostatnie wady oznaczają, że: jeśli mGA zostanie użyte do rozwiązania problemu, który nie jest w pełni separowalny, lub poszczególne podproblemy mają znacząco różny wpływ na wartość oceny całego rozwiązania, to najprawdopodobniej mGA będzie słabo radził sobie z rozwiązywaniem takich problemów. Na tej podstawie można stwierdzić, że mGA jest ciekawą propozycją teoretyczną, ale mającą niewielką wartość praktyczną. Powyższe stwierdzenie dotyczy również Szybkiego Nieporządnego Algorytmu Genetycznego (ang. *fmGA - Fast Messy Genetic Algorithm*) i zostało potwierdzone wynikami zaprezentowanymi w [57].

Teza główna pracy

Bazując na idei kodowania nieporządnego (ang. *messy coding*) jest możliwe zaprojektowanie nowej metody obliczeniowej, efektywniejszej od nieporządnego algorytmu genetycznego, innych metod ewolucyjnych (np. algorytm BOA, ang. *Bayesian Optimization Algorithm*), a także skuteczniejszej w zastosowaniach praktycznych i o mniejszej czułości na ustawienia parametrów sterujących.

Cel pracy

Celem niniejszej pracy jest zaproponowanie nowej metody obliczeniowej wykorzystującej podejście ewolucyjne, która eliminowałaby, lub pomniejszała znaczenie niektórych wad typowych dla algorytmu genetycznego, takich jak przedwczesna zbieżność, utykanie w optimum lokalnym, oraz istotny spadek efektywności spodowany wzrostem długości genotypu.

Zadania badawcze

Dla osiągnięcia celu rozprawy określono następujące zadania badawcze do wykonania:

1. Wybór NP-zupełnego problemu praktycznego, dla którego metoda bazująca na szablonie pracy algorytmu genetycznego będzie wykazywać takie wady, jak przedwczesna zbieżność, oraz spadek efektywności spowodowany wzrostem długości genotypu.

2. Określenie, które cechy problemu najistotniej, negatywnie wpływają na efektywność metody opartej na idei algorytmu genetycznego

3. Opracowanie nowej metody wykorzystującej podejście ewolucyjne, która eliminuje lub pomniejsza znaczenie tych cech problemu, które zostały wyszczególnione w ramach wykonania zadania 2

4. Porównanie skuteczności proponowanego rozwiązania (nowej metody obliczeniowej wykorzystującej podejście ewolucyjne) ze skutecznością innych metod opartych na idei ewolucji

    4.1. Wybór zadania obliczeniowego dla testów
    4.2. Wybór metod do porównania z proponowanym rozwiązaniem
    4.3. Badania
    4.4. Wnioski

5. Zastosowanie, zaproponowanego w ramach wykonania zadania 4, nowego szablonu pracy dla algorytmu ewolucyjnego do rozwiązania NP-zupełnego problemu praktycznego wybranego w punkcie 1

6. Porównanie skuteczności zaproponowanej metody w rozwiązywaniu NP-zupełnego problemu praktycznego wybranego w punkcie 1 z innymi podejściami znanymi z literatury

Poruszana w niniejszej pracy tematyka, oraz część prezentowanych i omawianych metod nie jest algorytmami, choć jest w ten sposób nazywana. Może to wprowadzać pewien

nieporządek w nazewnictwie metod. Zdaniem autora uniknięcie tego zjawiska nie jest niestety możliwe, ponieważ funkcjonujące w literaturze przedmiotu nazewnictwo (np. Algorytm Genetyczny, czy Algorytm Ewolucyjny) narzuca takie ramy opisu metod. W związku z powyższym autor pragnie zaznaczyć, że w istotnej części niniejszej pracy pojęcia „algorytm" i „metoda" są używane wymiennie, choć jest to sprzeczne z formalnym znaczeniem tych nazw.

Praca składa się z 6 rozdziałów.

W rozdziale 1 został opisany wybrany problem praktyczny – problem projektowania przepływu bez rozgałęzień w szkieletowych sieciach komputerowych.

Rozdział 2 zawiera opis metod rozwiązujących wybrany problem praktyczny, w tym zaproponowanej przez autora metody HEFAN, stanowiącej punkt wyjścia do dalszych badań. W tym samym rozdziale (podrozdział 2.4.6) znajduje się wyszczególnienie tych cech problemu praktycznego, które wynikły z przeprowadzonych badań. Rozdział zamyka analiza wyszczególnionych cech problemu praktycznego (podrozdział 2.4.7), oraz wskazanie motywacji dla prac nad propozycją nowego szablonu pracy dla metod bazujących na idei ewolucji.

Rozdział 3 prezentuje metodę MuPPetS (Multi Population Pattern Searching Algorithm). Jest to propozycja nowego szablonu pracy dla metod bazujących na idei ewolucji. Rozdział prezentuje również porównanie skuteczności metody MuPPetS z innymi metodami ewolucyjnymi w rozwiązywaniu znanych z literatury problemów testowych. W opinii autora rozdział 3 stanowi główne osiągnięcie pracy.

W rozdziale 4 przedstawiono dostosowanie zaproponowanej w rozdziale 3 metody MuPPetS do rozwiązania problemu praktycznego - problemu projektowania przepływu bez rozgałęzień w szkieletowej sieci komputerowej zorientowanej połączeniowo.

Rozdział 5 prezentuje wyniki przeprowadzonych badań z użyciem metody MuPPetS dla problemu projektowania przepływu, oraz porównania z innymi metodami znanymi z literatury.

Ostatni rozdział 6 zawiera podsumowanie pracy, oraz propozycje dalszych badań.

1. Problem praktyczny – projektowanie przepływu bez rozgałęzień w szkieletowych sieciach komputerowych zorientowanych połączeniowo

Jako problem praktyczny stanowiący punkt wyjścia do badań został wybrany problem projektowania przepływu bez rozgałęzień w szkieletowych sieciach komputerowych zorientowanych połączeniowo. Przepływ bez rozgałęzień to taki przepływ wieloskładnikowy, w którym każdy składnik przepływa od swojego źródła do ujścia wzdłuż tylko jednej trasy, w odróżnieniu od przepływu z rozgałęzieniami, gdzie każdy składnik może przepływać wzdłuż wielu tras.

Projektowanie przepływu (ang. *Flow Assigment*) można sklasyfikować jako jeden z czterech podstawowych problemów związanych z projektowaniem sieci rozległych [50, 112]. Poza projektowaniem przepływu są to:

- Wyznaczanie przepustowości łuków w sieci komputerowej (ang. *Capacity Assigment*)
- Jednoczesne wyznaczanie przepływów i przepustowości łuków w sieci komputerowej (ang. *Flow and Capacity Assigment*)
- Jednoczesne wyznaczanie przepływów, przepustowości łuków i topologii sieci komputerowej (ang. *Topology, Capacity and Flow Assigment*)

Jako metodę oceny jakości przepływu wybrano funkcję LFL (ang. *Lost Flow In Link*) [78, 100, 103, 105, 106, 107, 108, 110, 111, 112]. Powyższego wyboru dokonano z następujących powodów:

- Wybrany problem jest problemem NP-zupełnym [75]
- W przypadku sposobu kodowania problemu oraz eksperymentów zaproponowanych w pracach [76, 77, 78] niezbędne jest użycie dużej liczby genów (od 1260 do 2500)
- Metoda rozwiązująca wybrany problem projektowania przepływu może znaleźć zastosowanie w praktyce np. w ramach takich technologii jak ATM (ang. *Asynchronous Transfer Mode*), MPLS (ang. *Multiprotocol Label Switching*) i DWDM (ang. *Dense Wavelength Division Multiplexing*) [1, 4, 6, 10, 17, 32, 37, 51, 52, 53, 61, 70, 75, 81, 88, 92, 96, 99, 102, 114]
- Wybrana metoda oceny jakości przepływu (funkcja LFL) jest kryterium uwzględniającym jakość zapewnianych przez sieć usług (ang. *Quality of Service*), a ponadto dotyczy sieci przeżywalnych (ang. *survivable*), czyli takich które posiadają zdolność wykrycia uszkodzenia i takiego przekonfigurowania swoich zasobów, aby zminimalizować wpływ uszkodzenia na jakość swojego działania

1.1 Przyjęty model sieci szkieletowej i przepływu w sieci

Przyjęty model sieci szkieletowej i przepływu w sieci jest przedstawiony m.in. w [76, 77, 78, 112]. Główne problemy związane z przepływem w sieciach są omówione dokładniej w szeregu prac dotyczących problematyki sieci komputerowych. Są to m.in.: [2, 5, 23, 33, 37, 48, 49, 50, 54, 56, 75, 96, 112, 114].

Topologię sieci definiujemy jako graf *G=(V,A)*, gdzie *V* oznacza zbiór *n* węzłów sieci, a *A* jest zbiorem *m* uporządkowanych par określonym na zbiorze *V*, reprezentujących łuki sieci. Łukiem zorientowanym sieci nazywamy uporządkowaną parę węzłów $<v, z> \in A$, gdzie *v* to węzeł początkowy łuku $<v, z>$, a *z* to węzeł końcowy łuku $<v, z>$. Dla ułatwienia zapisu łuki sieci będą numerowane: $a = 1, ..., m$. Jeśli $a = <v, z>$, to węzeł początkowy łuku a oznaczamy przez $o(a) = v$, a węzeł końcowy przez $d(a) = z$.

Unigraf to graf, w którym każdej uporządkowanej parze węzłów przypisany jest co najwyżej jeden łuk. Graf właściwy to graf, który nie zawiera pętli. Pętla to taki łuk, w którym ten sam węzeł jest węzłem początkowym i końcowym łuku. W niniejszej pracy rozważamy tylko takie modele topologiczne sieci, które są unigrafami właściwymi.

Sieć definiujemy jako $S=<G; h_1, ..., h_w>$, gdzie unigraf właściwy G reprezentuje topologię sieci, a $h_i (i = 1, .., w)$ to funkcje, które przyporządkowują każdemu łukowi ze zbioru *A* nieujemną liczbę rzeczywistą.

Ograniczenie przepustowości łączy realizowane jest w niniejszym modelu sieci, poprzez wprowadzenie funkcji przepustowości opisanej na łukach: $c : A \rightarrow R^+ \cup \{0\}$. Przepustowość łuku $a = <v, z>$ określamy jako c_a lub $c(v, z)$.

Niniejsza praca dotyczy przepływów wieloskładnikowych, gdzie składnik to zbiór pakietów mających ten sam węzeł źródłowy i docelowy. Dla opisania przepływu wieloskładnikowego przyjmujemy następujące oznaczenia:

q – liczba składników
k – *k*-ty składnik przepływu wieloskładnikowego, gdzie $k = 1,...,q$
o(k) – węzeł początkowy (źródło) dla *k*-tego składnika
d(k) – węzeł końcowy (ujście) dla *k*-tego składnika
r_{ij} - średnie natężenie ruchu z węzła *i* do węzła *j*
r_k - średnie natężenie ruchu dla *k*-tego składnika. Należy zauważyć, że jeśli $o(k) = i$, $d(k) = j$ to $r_k = r_{ij}$.
$f^k : A \rightarrow R^+ \cup \{0\}$, gdzie $k = 1, ..., q$ - zespół funkcji, który nazywamy przepływem wieloskładnikowym
$f^k(v, z)$ - przepływ *k*-tego składnika w łuku $<v, z>$
$f(v, z) = \sum_{k=1}^{q} f^k(v, z)$ – sumaryczny przepływ w łuku $<v, z>$
$f_a = f(v, z)$ dla $a = <v, z>$ - sumaryczny przepływ w łuku $<v, z>$ zapisany przy pomocy indeksu
$\underline{f} = [f_1, f_2, ..., f_m]$ - całkowity przepływ dla wszystkich łuków sieci
$c(v, z)$ - przepustowość łuku $<v, z>$
c_a - dla $a = <v, z>$ przepustowość łuku $<v, z>$ zapisana przy pomocy indeksu

Każda z funkcji $f^k(v,z)$, k = 1, ..., q, spełnia następujące warunki:

$$\sum_{z\in D(v)} f^k(v,z) - \sum_{z\in B(v)} f^k(z,v) = \begin{cases} r_k & dla \quad v = o(k) \\ -r_k & dla \quad v = d(k), \\ 0 \quad w & pozostalych \quad przypadkach \end{cases} \tag{1}$$

gdzie
$D(v) = \{z: z \in V \; i \; <v, z> \in A\}$ (zbiór węzłów, do których prowadzą łuki wychodzące z węzła v), $B(v) = \{z: z \in V \; i \; <z, v> \in A\}$ (zbiór węzłów, z których prowadzą łuki skierowane do węzła v).

$$f^k(v,z) \geq 0 \quad \text{dla każdego } <v, z> \in A,\ k = 1, ..., q \tag{2}$$

Sumaryczny przepływ w każdym z łuków, musi spełniać ograniczenie związane z przepustowością łuku:

$$f(v,z) \leq c(v,z), \quad \text{dla każdego } <v, z> \in A \tag{3}$$

Jeśli przyjmiemy oznaczenie łuku przy pomocy indeksu, to powyższy warunek można zapisać:

$$f_a \leq c_a, \quad \text{dla każdego } a \in A \tag{4}$$

Zakładamy, że rozmiar przepływu i przepustowości łączy wyrażany jest w jednostkach uniwersalnych, które można interpretować np. jako ilość bitów, która może być przesłana w jednostce czasu.

Dla przepływu wieloskładnikowego znane są dwa sposoby opisu, które można znaleźć w literaturze: „łuk-trasa” (ang. *link-path*) i „węzeł-łuk” (ang. *node-link*) [75, 112]. W niniejszej pracy używany jest opis łuk-trasa. Na potrzeby opisu łuk-trasa przyjmujemy następujące oznaczenia i definicję trasy:

Trasa. Niech $v_1, v_2, ..., v_u$, (u > 1), będzie ciągiem węzłów spełniających warunek: $v_l \neq v_{l'}$ (dla l=1,...,u, l'=1,...,u, $l \neq l'$) i takich że: $< v_l, v_{l+1} >$ jest łukiem zorientowanym dla każdego l=1,...,u-1. Ciąg węzłów i łuków $v_1, < v_1, v_2 >, v_2, ..., v_{u-1}, < v_{u-1}, v_u >, v_u$ nazywamy trasą. Należy zauważyć, że zgodnie z definicją trasa nie może zawierać dwóch takich samych węzłów, ani łuków.

P – zbiór połączeń typu unicast

p – pojedyncze połączenie typu unicast, takie że $p \in P$, $p = 1,2,...,q$. Każde połączenie p jest definiowane przez trójkę: *o(p), d(p)* i Q_p.

o(p) – węzeł początkowy połączenia *p*
d(p) – węzeł końcowy połączenia *p*
Q_p - zapotrzebowanie połączenia *p*
$\mathbf{Q} = [Q_1, Q_2, ..., Q_q]$ - wektor zapotrzebowań dla wszystkich połączeń
Π_p - zbiór indeksów tras dla połączenia *p*
π_p^k - *k*-ta trasa dla połączenia *p*, $k \in \Pi_p$, zakładamy, że $o(\pi_p^k)=o(p)$, oraz $d(\pi_p^k)=d(p)$
$o(\pi_p^k)$ – węzeł początkowy trasy π_p^k
$d(\pi_p^k)$ – węzeł końcowy trasy π_p^k
$\Pi = \bigcup_{p \in P} \Pi_p$ - zbiór indeksów wszystkich propozycji tras dla wszystkich połączeń *p*

Dla określenia, która z dostępnych tras π_p^k została przypisana dla danego połączenia *p* przypisujemy każdej trasie π_p^k zmienną:

$$x_p^k = \begin{cases} 1, & \text{jeżeli połączenie p przepływa wzdłuż trasy } \pi_p^k \\ 0, & \text{w przeciwnym wypadku} \end{cases} \tag{5}$$

Zmienne x_p^k muszą spełniać następujące warunki:

$$x_p^k \in \{0,1\}$$
dla każdego $p \in P$, $k \in \Pi_p$ (6)

$$\sum_{k \in \Pi_p} x_p^k = 1 \tag{7}$$
dla każdego $p \in P$

Ograniczenie (7) wynika z faktu, że w niniejszej pracy zajmujemy się przepływem bez rozgałęzień.

Dla określenia czy łuk *a* przynależy do trasy π_p^k, wprowadzamy zmienną binarną:

$$\delta_{pa}^k = \begin{cases} 1, & \text{jeżeli trasa } \pi_p^k \text{ zawiera łuk } a \\ 0, & \text{w przeciwnym przypadku} \end{cases} \tag{8}$$

W związku z powyższym możemy określić wartość sumarycznego przepływu w łuku *a* jako:

$$f_a = \sum_{p \in P} \sum_{k \in \Pi_p} \delta_{pa}^k x_p^k Q_p \tag{9}$$
dla każdego $a \in A$

1.2 Funkcja LFL

Dla zdefiniowania funkcji LFL (ang. *Lost Flow In Link*) przyjmujemy:

Sumaryczny przepływ we wszystkich łukach wychodzących z węzła *v*:

$$g_v^{\text{out}} = \sum_{i:o(i)=v} f_i \tag{10}$$

Sumaryczna przepustowość wszystkich łuków wychodzących z węzła *v*:

$$e_v^{\text{out}} = \sum_{i:o(i)=v} c_i \tag{11}$$

Sumaryczny przepływ we wszystkich łukach wchodzących do węzła *v*:

$$g_v^{\text{in}} = \sum_{i:d(i)=v} f_i \tag{12}$$

Sumaryczna przepustowość wszystkich łuków wchodzących do węzła *v*:

$$e_v^{\text{in}} = \sum_{i:d(i)=v} c_i \tag{13}$$

Rezydulna przepustowość łuku *a*:

$$c_a - f_a,$$
gdzie $a \in A$ (14)

Rozważając możliwości lokalnego odtworzenia przepływu w przypadku awarii wybranego łuku $a \in A$ należy zauważyć, że jeśli rezydualna przepustowość wszystkich łuków wychodzących z węzła początkowego łuku *a* jest mniejsza niż przepływ, który przepływał przez łuk *a*, to ta część tego przepływu, która wykracza ponad sumę rezydualnych przepustowości pozostałych łuków wychodzących z węzła początkowego łuku *a*, zostanie utracona. Dla łuku *a* definiujemy funkcję LA_a^{out}, która określa ilość przepływu z łuku *a*, która zostanie utracona w węźle *o(a)* w przypadku awarii łuku *a*.

$$LA_a^{\text{out}}(\underline{f}) = \varepsilon\left(g_{o(a)}^{\text{out}} - (e_{o(a)}^{\text{out}} - c_a)\right),$$
gdzie
$$\varepsilon(x) = \begin{cases} 0 & \text{for} \quad x \le 0 \\ x & \text{for} \quad x > 0 \end{cases} \tag{15}$$

Analogicznie definiujemy funkcję LA_a^{in}, która określa ilość przepływu przechodzącego przez łuk *a*, który zostanie utracony z powodu zbyt małej rezydualnej przepustowości łuków wchodzących do węzła *d(a)* (końcowego węzła łuku *a*).

$$LA_a^{\text{in}}(\underline{f}) = \varepsilon\left(g_{d(a)}^{\text{in}} - (e_{d(a)}^{\text{in}} - c_a)\right) \tag{16}$$

Na podstawie LA_a^{out} i LA_a^{in} możemy zdefiniować funkcje: $LN_v^{\text{out}}(\underline{f})$ i $LN_v^{\text{in}}(\underline{f})$ określające odpowiednio: potencjalnie tracony przepływ dla wszystkich łuków opuszczających węzeł v, oraz potencjalnie tracony przepływ dla wszystkich łuków wchodzących do węzła v.

$$LN_v^{\text{out}}(\underline{f}) = \sum_{a:o(a)=v} \varepsilon\left(g_v^{\text{out}} - (e_v^{\text{out}} - c_a)\right) = \sum_{a:o(a)=v} LA_a^{\text{out}}(\underline{f}) \tag{17}$$

$$LN_v^{\text{in}}(\underline{f}) = \sum_{a:d(a)=v} \varepsilon\left(g_v^{\text{in}} - (e_v^{\text{in}} - c_a)\right) = \sum_{a:d(a)=v} LA_a^{\text{in}}(\underline{f}) \tag{18}$$

Na podstawie funkcji $LN_v^{\text{out}}(\underline{f})$ i $LN_v^{\text{in}}(\underline{f})$, możemy zdefiniować funkcję określającą poziom przygotowania całej sieci do wykonania lokalnego odtworzenia przepływu:

$$LFL(\underline{f}) = \sum_{v\in V} \left(LN_v^{\text{out}}(\underline{f}) + LN_v^{\text{in}}(\underline{f})\right)/2 \tag{19}$$

Ponadto:

$$LFL(\underline{f}) = \sum_{a\in A} \left(LA_a^{\text{out}}(\underline{f}) + LA_a^{\text{in}}(\underline{f})\right)/2 \tag{20}$$

Na podstawie formułujemy następujące zadanie optymalizacji:

$$\min_{\underline{f}} \quad LFL(\underline{f}) \tag{21}$$

Przy następujących ograniczeniach:

$$\sum_{k\in\Pi_p} x_p^k = 1 \tag{22}$$

dla każdego $p\in P$

$$x_p^k \in \{0,1\} \tag{23}$$

dla każdego $p\in P$, $k \in \Pi_p$

$$f_a = \sum_{p\in P}\sum_{k\in\Pi_p} \delta_{pa}^k x_p^k Q_p \tag{24}$$

dla każdego $a \in A$

$$f_a \le c_a, \tag{25}$$

dla każdego $a \in A$

Równania (19) i (21) ograniczają rozważania projektowania przepływu do przepływu wieloskładnikowego bez rozgałęzień. Wzór (24) definiuje wartość przepływu w każdym łuku sieci. Warunek (25) wprowadza ograniczenie związane z przepustowością łączy.

1.3 Sieci testowe i wymagania dla przepływu

Jak już wspomniano, do oceny jakości rozwiązań wybrano funkcję LFL. Wybrano również zestaw testowych sieci, na których przeprowadzono eksperymenty. Zgodnie z literaturą przedmiotu i celem pracy, wybrano sieci, które można uznać za duże [91, 112]. Dołożono wszelkich starań aby do testów użyć sieci o takich podstawowych parametrach (minimalna, maksymalna i średnia incydencja węzła), które mogą zostać uznane typowe [91, 115]. Ponadto testowane były sieci o regularnej i nieregularnej siatce. Parametry oraz rodzaj sieci mają istotny wpływ na liczbę możliwych do zestawienia tras, a co za tym idzie mogą powodować, że różne metody będą cechować się różną jakością rozwiązań dla różnych rodzajów sieci.

Dodatkowo przeprowadzono 3 rodzaje eksperymentów A, B i C. Każdy z rodzajów eksperymentów inaczej ustawia przepustowość łuków sieci, wymaganą liczbę połączeń do zestawienia, oraz sposób wyboru połączeń. Ustawienia każdego z rodzajów eksperymentów w inny sposób wpływają na rozmiar przestrzeni rozwiązań, a także tworzą inne warunki do zestawiania tras w sieci.

Należy zauważyć, że sieci testowe, oraz rodzaje eksperymentów na nich przeprowadzone zostały tak dobrane, aby sprawdzić jakość rozwiązań proponowanych przez każdą z testowanych metod w możliwie różnych warunkach, które jednocześnie mogą również być uznane za typowe z punktu widzenia stanu wiedzy technicznej i praktyki. W konsekwencji ma to zapewnić maksymalnie wiarygodną ocenę skuteczności i efektywności każdej z testowanych metod, a także ich potencjału dla zastosowań praktycznych.

Badania zaprezentowane w niniejszej pracy zostały przeprowadzone na sieciach zawierających 36 węzłów. Sieci takie mogą zostać uznane za duże [91, 112]. Do badań wykorzystano sześć sieci: NET 104, NET 114, NET 128, NET 144, NET 162 i NET 120. Pięć pierwszych z nich ma topologię nieregularnej siatki, a sieć NET 120 ma topologię regularnej siatki typu Manhattan. Sieci NET 114, NET 128, NET 144 i NET 162 zostały utworzone na podstawie sieci bazowej (NET 104), poprzez dodanie nowych łuków. Najważniejsze informacje dotyczące parametrów i eksperymentów z użyciem poszczególnych sieci zostały zamieszczone w Tabela 1.

Tabela 1. Parametry testowanych sieci

Sieć	**NET 104**	**NET 114**	**NET 128**	**NET 144**	**NET 162**	**NET 120**
Liczba węzłów	36	36	36	36	36	36
Liczba łuków	104	114	128	144	162	120
Minimalna incydencja węzła	2	2	3	3	3	2
Maksymalna incydencja węzła	5	5	6	6	6	4
Średnia incydencja węzła	2.89	3.17	3.56	4.00	4.50	3.33
Topologia	nieregularna siatka					regularna siatka

Należy zauważyć, że podstawowe parametry sieci, na których prowadzono badania (minimalna, maksymalna i średnia incydencja węzła) może zostać uznana za typową [91,

115]. Dla każdej sieci przeprowadzono 3 rodzaje eksperymentów: A, B i C (patrz Tabela 2). W eksperymentach A i B wszystkie łuki testowanych sieci posiadały taką samą przepustowość równą 4800 jp (jednostek przepustowości). W eksperymentach typu C łuki testowanych sieci mają przepustowość o wartości k*1200 jp, gdzie k = 1,...,8. Przepustowości zostały dobrane ze względu na kanały optyczne OC-48 i OC-12 stosowane powszechnie w sieciach transportowych [37, 97, 112]. W eksperymentach typu A zestawianych jest po jednym połączeniu dla każdej możliwej pary węzłów (1260 połączeń), zapotrzebowanie wszystkich połączeń na przepływ jest identyczne. W eksperymentach typu B i C węzeł startowy i końcowy dla każdego połączenia wybierane są losowo, a ich liczba wynosi 2500. Zestawienie ustawień w zależności od eksperymentu znajduje się w tabeli Tabela 2.

Tabela 2. Parametry sieci i zapotrzebowań w poszczególnych eksperymentach

Eksperyment	Przepustowość łuków sieci	Liczba połączeń	Sposób wyboru połączeń	Zapotrzebowanie
A	4800	1260	1 dla każdej pary	równe dla wszystkich połączeń
B	4800	2500	losowo	losowe
C	k*1200, gdzie k=1,…,8	2500	losowo	losowe

Opisane sieci (NET 104, NET 114, NET 128, NET 144, NET 162 i NET 120), wraz z opisanymi konfiguracjami (eksperymenty typu A, B i C) były przedmiotem badań, których wyniki są ogólnie dostępne [76, 77, 78, 100, 101, 103, 105, 106, 107, 108, 110, 111, 112].

2. Metody poszukiwania rozwiązań dla problemu projektowania przepływu wieloskładnikowego bez rozgałęzień

W literaturze przedmiotu można znaleźć liczne rodzaje algorytmów proponujących rozwiązania dla problemu projektowania przepływu w szkieletowych sieciach komputerowych. Są to algorytmy różnego rodzaju: od metod opartych na algorytmach deterministycznych takich jak algorytm FD (ang. *Flow Deviation*), poprzez algorytmy aproksymacyjne z zastosowaniem relaksacji Lagrange'a, aż po algorytmy oparte na elementach sztucznej inteligencji. W niniejszym rozdziale zaprezentowane zostały metody mieszczące się w głównych nurtach rozwoju badań nad projektowaniem przepływu w szkieletowych sieciach komputerowych.

2.1 Metody oparte o metodę FD

Algorytm FD (ang. *Flow Deviation*) [24] został zaproponowany w latach 70. jako rozwiązanie problemu projektowania przepływu z rozgałęzieniami. Po pewnych modyfikacjach może być jednak również stosowany do rozwiązywania problemów projektowania przepływu bez rozgałęzień [77, 106, 112]. Algorytm FD, lub jego elementy zostały wykorzystane do wyznaczania rozwiązań dla problemu projektowania przepływu w wielu pracach [9, 48, 50, 75, 76, 77, 78].

Algorytm FD proponuje dobór tras dla przepływu z rozgałęzieniami na podstawie metryki bazującej na pochodnej funkcji kryterialnej. Zgodnie z tą metryką dobiera się najkrótsze trasy łączące poszczególne węzły. Można do tego celu zastosować dowolny algorytm SPF (ang. *Shortest Path Fisrt*), np. algorytm Dijkstry.

Dla problemu projektowania przepływu z rozgałęzieniami algorytm FD proponuje rozwiązania optymalne. W przypadku odmian algorytmu FD rozwiązujących problem projektowania przepływu bez rozgałęzień metody te nie gwarantują wyznaczenia optymalnego rozwiązania, jednak ich jakość może z powodzeniem konkurować z rozwiązaniami proponowanymi przez inne algorytmy [77].

Na mechanizmie algorytmu FD bazują m.in. algorytmy UFD_INI i UFD_LFL [112], z tym że algorytm UFD_INI używa jako metryki łuku pochodnej cząstkowej funkcji średniego opóźnienia pakietu (26), a algorytm UFD_LFL pochodnej cząstkowej funkcji LFL (28).

$$DEL(f) = \frac{1}{\kappa} \sum_{a \in A} \frac{f_a}{c_a - f_a}, \tag{26}$$

gdzie κ jest natężeniem sumarycznego strumienia pakietów wprowadzanych do sieci z zewnątrz

$$f_{kryt} = \partial DEL / \partial f_a \tag{27}$$

$$f_{kryt} = \partial LFL / \partial f_a \tag{28}$$

2.2 Metody oparte o Relaksację Lagrange'a

Relaksacja Lagrange'a to popularna w literaturze metoda pozwalająca na wyznaczenie rozwiązania problemu za pomocą wyznaczania dolnych ograniczeń [21, 38, 46, 74, 75, 78, 112]. Metoda relaksacji Lagrange'a bazuje na teorii dualności, zgodnie z którą optymalne rozwiązanie problemu dualnego jest również optymalnym rozwiązaniem problemu pierwotnego. W literaturze przedmiotu można znaleźć liczne przykłady zastosowań relaksacji Lagrange'a do konstrukcji algorytmów proponujących rozwiązania dla problemu projektowania przepływu [16, 29, 30, 31, 37, 42, 43, 75, 84, 112].

Badania i opis algorytmu LRH (ang. *Lagrange Relaxation Heuristic*), opartego na relaksacji Lagrange'a wyznaczającego przepływ bez rozgałęzień dla funkcji oceny LFL można znaleźć w pracach [78, 112].

W [112] zostało zaproponowane następujące dualne zadanie optymalizacji:

$$LFL^D(\lambda) = \min_{\lambda} LFL(f) + \sum_{a \in A} \lambda_a \left(\sum_{p \in P} \sum_{k \in \Pi_p} \delta_{pa}^k x_p^k Q_p - f_a \right) \tag{29}$$

Należy zauważyć, że zbiór rozwiązań dopuszczalnych dla problemu (21) jest podzbiorem rozwiązań problemu (29). Ponadto przy założeniu, że dla każdego łuku $a \in A$, $\lambda_a \geq 0$, to dla każdego rozwiązania dopuszczalnego dla (21) wartość $\sum_{a \in A} \lambda_a \left(\sum_{p \in P} \sum_{k \in \Pi_p} \delta_{pa}^k x_p^k Q_p - f_a \right)$ jest nie większa niż 0, a wartość $LFL^D(\lambda)$ nie jest większa niż dla (21). W związku z tym, zgodnie z teorią dualności (29) jest dolnym ograniczeniem (20). Najlepsze dolne ograniczenie jest osiągane dla wektora λ^*, dla którego $LFL^D(\lambda^*) = \max_{\lambda} LFL^D(\lambda)$. Do wyznaczenia współczynników Lagrange'a algorytm LRH używa algorytmu optymalizacji subgradientowej [112]. Algorytm optymalizacji subgradientowej na końcu każdej swojej iteracji proponuje rozwiązanie problemu dualnego, na jego podstawie można wyznaczyć rozwiązanie problemu pierwotnego. Jeżeli rozwiązanie problemu pierwotnego jest dopuszczalne, to jest wykorzystywane jako rozwiązanie początkowe dla algorytmu UFD_LFL [112] (patrz: rozdział 2.1). Jeżeli rozwiązanie problemu pierwotnego nie jest dopuszczalne, to jest ono wykorzystywane jako rozwiązanie początkowe dla algorytmu UFD_INI [112], który ma za zadanie zwrócenie dopuszczalnego rozwiązania i dopiero to rozwiązanie jest wykorzystywane jako rozwiązanie początkowe dla algorytmu UFD_LFL. Zgodnie z [112] taka konstrukcją algorytmu polega na idei wykorzystania relaksacji Lagrange'a wraz z optymalizacją subgradientową jako inteligentnego narzędzia do przeszukiwania przestrzeni rozwiązań, natomiast użycie algorytmów UFD_INI i UFD_LFL ma zapewnić dopuszczalność rozwiązania i minimalizację wartości funkcji LFL.

2.3 Metody bazujące na podejściu ewolucyjnym

W literaturze można znaleźć liczne propozycje metod rozwiązywania problemu projektowania przepływu bez rozgałęzień oparte o idee obliczeń ewolucyjnych. Są to m.in. algorytmy ewolucyjne oparte o typową metodę kodowania za pomocą łańcuchów liczb naturalnych [19, 20, 76, 77, 78, 82, 85, 86]. W części algorytmów należących do tej grupy osobniki nie stanowią bezpośredniej reprezentacji problemu, ale [20, 86] wagi przypisane

poszczególnym łukom sieci, które później są używane przez inny algorytm do wyznaczenia ostatecznego rozwiązania.

Należy również zauważyć, że w wielu pracach [19, 82, 85, 86] można znaleźć wyniki badań dla problemów o znacznie mniejszej przestrzeni rozwiązań, niż w przypadku prac prezentujących autorski algorytm **H**ierarchical **E**volutionary algorithm for **F**low **A**ssigment in **N**on-bifurcated commodity flow (HEFAN) [76, 77, 78]. Liczba węzłów w żadnej z sieci, dla których prowadzone są badania w pracach [19, 82, 85, 86], nie przekracza 20, a często jest to znacznie mniej. Również liczba połączeń, które są do zestawienia jest niewielka – dla wymienionych prac nie przekracza 1100.

Autor niniejszej pracy nie napotkał na publikację, w których zaprezentowany algorytm bazujący na idei algorytmu ewolucyjnego stosowałby inne niż typowe dla algorytmów wywodzących się z idei algorytmu genetycznego operatory, takie jak mutacja i krzyżowanie. Zaletą takich rozwiązań jest ich prostota, wadą natomiast jest niższa jakość wyników spowodowana brakiem wykorzystania cech charakterystycznych dla rozwiązywanego problemu. W trakcie studiów literaturowych nie napotkano również na algorytmy stosujące w ramach używanego algorytmu ewolucyjnego innej struktury niż pojedynczej populacji.

2.4 HEFAN (Hierarchical Evolutionary algorithm for Flow Assigment in Non-bifurcated commodity flow)

Zaproponowany przez autora algorytm HEFAN [76, 77, 78] należy do grupy algorytmów wykorzystujących w działaniu metody z zakresu sztucznej inteligencji. Istnieją jednak istotne różnice pomiędzy budową algorytmu HEFAN, a budową pozostałych algorytmów opisanych w punkcie 2.3. Zastosowane w algorytmie HEFAN mechanizmy wprowadzające nowe operatory i elementy hierarchii w działaniu całego algorytmu stanowią, zdaniem autora, istotną nowość w literaturze problemu. W związku z wysoką efektywnością algorytmu HEFAN w porównaniu z innymi, znanymi metodami bazującymi na algorytmie FD i relaksacji Lagrange'a, algorytm ten został wybrany jako punkt wyjścia do badań, których celem było zaproponowanie nowej metody projektowania przepływu bez rozgałęzień w sieciach komputerowych. W związku z powyższym algorytm HEFAN został opisany dokładnie w niniejszym rozdziale. Opisana została wersja 2.2 algorytmu HEFAN zaprezentowana w pracy [78]. Wersja 1.0 algorytmu HEFAN została opisana w [76, 77]. Główną różnicą pomiędzy wersjami 1.0 i 2.2 jest wprowadzenie w wersji 2.2 funkcji *L(k,p)* pozwalającą oceniać jakość tras. Wszędzie tam, gdzie w wersji 2.2 trasy z bazy tras wybierane są przy użyciu funkcji *L(k,p)*, w wersji 1.0 są wybierane losowo z równym prawdopodobieństwem wyboru dla każdej trasy.

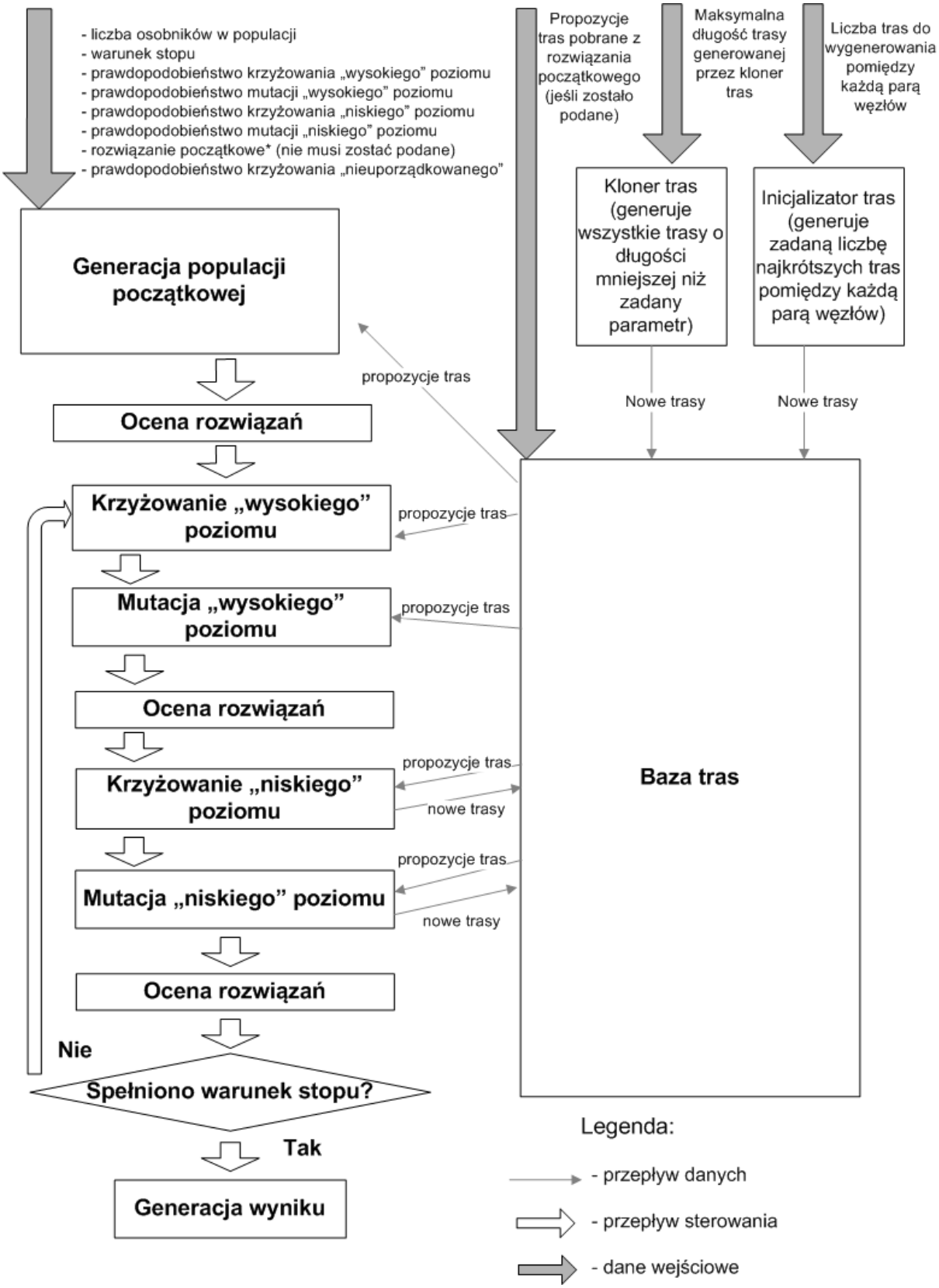
2.4.1 Ogólna charakterystyka algorytmu HEFAN

Celem działania algorytmu HEFAN jest znalezienie jak najlepszego rozwiązania z punktu widzenia wybranej metody oceny jakości rozwiązania, które jednocześnie nie łamie zadanych ograniczeń (patrz rozdział **Błąd! Nie można odnaleźć źródła odwołania.**). Charakterystyczną cechą algorytmu jest jego hierarchiczna struktura, która składa się z dwóch poziomów: „wysokiego" i „niskiego". Poziom „wysoki" to typowy dla algorytmu genetycznego zbiór osobników będących jednocześnie propozycjami rozwiązań dla zadanego problemu (w tym przypadku jest to zestaw tras). Poziom „niski" algorytmu HEFAN jako

osobnika traktuje pojedynczą trasę. Poziom „niski” posiada własny sposób oceny jakości poszczególnych osobników (tras), oraz własne operatory krzyżowania i mutacji, rozmiar jego populacji nie jest jednak stały (liczba tras wraz z kolejnymi iteracjami może jedynie rosnąć lub pozostać nie zmieniona), tak jak ma to miejsce w przypadku populacji osobników w typowym algorytmie genetycznym.

Należy zauważyć, że trudność (bez względu na przyjętą w celu określenia „trudności” miarę) zadania znalezienia jak najlepszego, ze względu na zadane kryterium, zestawu tras, jest ściśle powiązana z dostępnym dla algorytmu zbiorem tras, które mają posłużyć do zbudowania pełnego rozwiązania. Optymalnym zbiorem tras, byłby taki, który dla każdej zadanej pary węzłów zawiera dokładnie jedną, taką propozycję trasy, która jest częścią rozwiązania optymalnego. Wygenerowanie takiego zbioru nie jest jednak możliwe ze względów praktycznych, wymagałoby bowiem dokonania przeglądu zupełnego dostępnej przestrzeni rozwiązań. Podział algorytmu HEFAN na dwa poziomy działania „wysoki” i „niski” jest próbą odpowiedzi na powyższe trudności – poziom „wysoki” ma za zadanie poszukiwać jak najlepszego rozwiązania dla problemu na podstawie dostępnego zbioru tras. Poziom „niski” ma służyć, jako narzędzie do proponowania nowych tras, które mogą poprawić jakość rozwiązań proponowanych przez poziom „wysoki”.

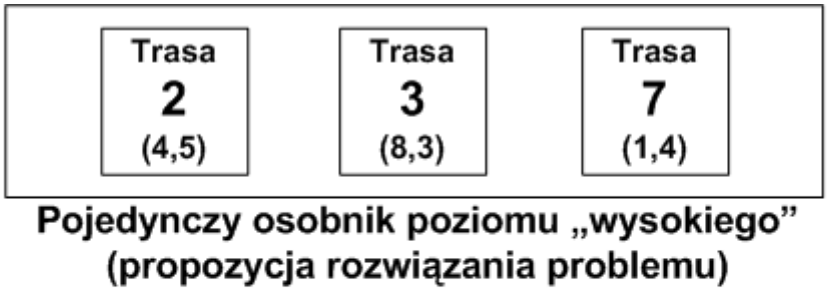
Ogólna struktura i schemat działania algorytmu HEFAN jest zamieszczona na Rys. 1.

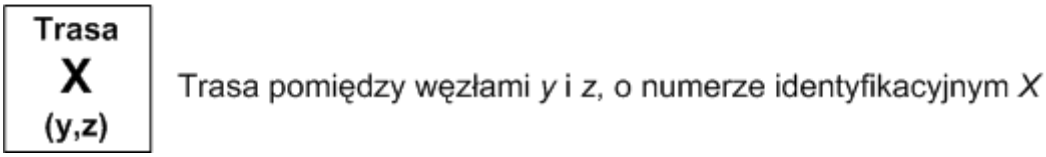


Rys. 1 Ogólna struktura algorytmu HEFAN

2.4.2 Reprezentacja osobników na poziomie „wysokim” i „niskim”

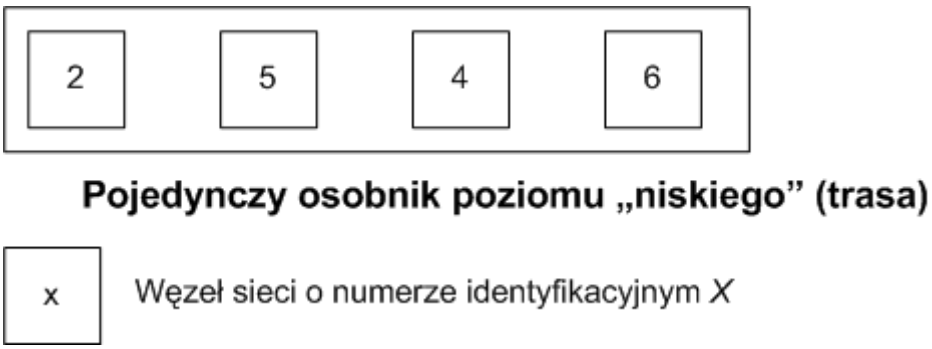
Rozwiązanie jest reprezentowane poprzez listę tras proponowanych do zestawienia pomiędzy zadanymi parami węzłów. Rozwiązanie jest reprezentowane przez pojedynczego osobnika na poziomie „wysokim” algorytmu. Przykład osobnika (rozwiązania) na poziomie „wysokim” (przy zadanych uporządkowanych parach węzłów [(4,5), (8,3), (1,4)]) znajduje się na Rys. 2.





Rys. 2. Reprezentacja osobnika na poziomie „wysokim”

Na poziomie „niskim” osobnikiem jest pojedyncza trasa. Każda trasa jest reprezentowana przez listę węzłów, przez które przechodzi zaczynając od węzła startowego, a kończąc na węźle końcowym. Przykład osobnika (trasy łączącej węzły 2 i 6) na poziomie „niskim” jest przedstawiony na Rys. 3.



Rys. 3. Reprezentacja osobnika na poziomie „niskim”

2.4.3 Baza tras

Zadaniem bazy tras (patrz Rys. 1) jest przechowywanie tras, które są znane algorytmowi. Każda trasa w bazie tras występuje w dokładnie jednej kopii. Jeśli w trakcie działania algorytmu nastąpi próba dodania do niej trasy, która już znajduje się w bazie tras, to trasa taka nie zostanie powtórnie dodana. Początkowo baza jest inicjalizowana pewną liczbą tras, których generacja następuje w trakcie inicjalizacji algorytmu. Za inicjalizację bazy tras odpowiedzialne są dwa elementy algorytmu: „kloner tras" i „inicjalizator tras", omówione poniżej. Do bazy wprowadzane są także wszystkie trasy rozwiązania początkowego, jeśli jakieś zostało przez użytkownika zaproponowane. W trakcie działania algorytmu do bazy dodawane są nowe trasy, które generowane są przez poziom „niski" algorytmu. Trasa raz wprowadzona do bazy tras pozostaje w niej aż do końca działania algorytmu.

Kloner tras. Kloner tras dostarcza do bazy tras, wszystkie możliwe trasy o długości nie większej niż wartość zadana przez użytkownika. Przez długość trasy rozumie się liczbę węzłów (wliczając w to węzeł startowy i końcowy) pomniejszoną o jeden.

Inicjalizator tras. Inicjalizator tras dostarcza do bazy tras zadaną przez użytkownika liczbę najkrótszych tras, pomiędzy każdą możliwą parą węzłów.

2.4.4 „Wysoki" poziom algorytmu

„Wysoki" poziom algorytmu HEFAN jest zbudowany w sposób typowy dla algorytmów genetycznych. Na tym poziomie używane są również typowe dla operatory, których opis może być znaleziony w licznych pozycjach w literaturze [14, 60, 65]. Każdy osobnik zawiera pełną propozycję rozwiązania (choć może ona łamać dowolną liczbę ograniczeń). Przyjęta konwencja kodowania została opisana w punkcie 2.4.2.

Wybór osobników do puli rodzicielskiej odbywa się przy użyciu metody ruletki. Wartość przystosowania każdego osobnika jest wyliczana na podstawie wartości funkcji LFL reprezentowanego przez tego osobnika rozwiązania. W celu zdefiniowania funkcji opisującej wartość przystosowania osobnika zdefiniujemy najpierw funkcję kary za złamanie ograniczenia przepustowości łączy sieci (30):

$$PEN_v(\underline{f}) = \sum_{a:o(a)=v} \varepsilon(f_a - c_a),$$

gdzie

$$\varepsilon(x) = \begin{cases} 0 & \text{for} \quad x \le 0 \\ x & \text{for} \quad x > 0 \end{cases}, \tag{30}$$

f_a - przepływ w łuku a,

c_a - przepustowość łuku a

Następnie zdefiniujemy funkcje $LN'(\underline{f})$ i $LFL'(\underline{f})$, analogiczne do (18) i (19), które zostały przedstawione w punkcie 1.2:

$$LN'_v(\underline{f}) = \begin{cases} \dfrac{LN_v^{\text{in}}(\underline{f}) + LN_v^{\text{out}}(\underline{f})}{2} & \text{dla} \quad PEN_v(\underline{f}) = 0 \\ \left(\dfrac{LN_v^{\text{in}}(\underline{f}) + LN_v^{\text{out}}(\underline{f})}{2} + PEN_v(\underline{f}) * 10\right)^2 & \text{dla} \quad PEN_v(\underline{f}) > 0 \end{cases} \tag{31}$$

$$LFL'(\underline{f}) = \sum_{v \in V} LN'_v(\underline{f}) \tag{32}$$

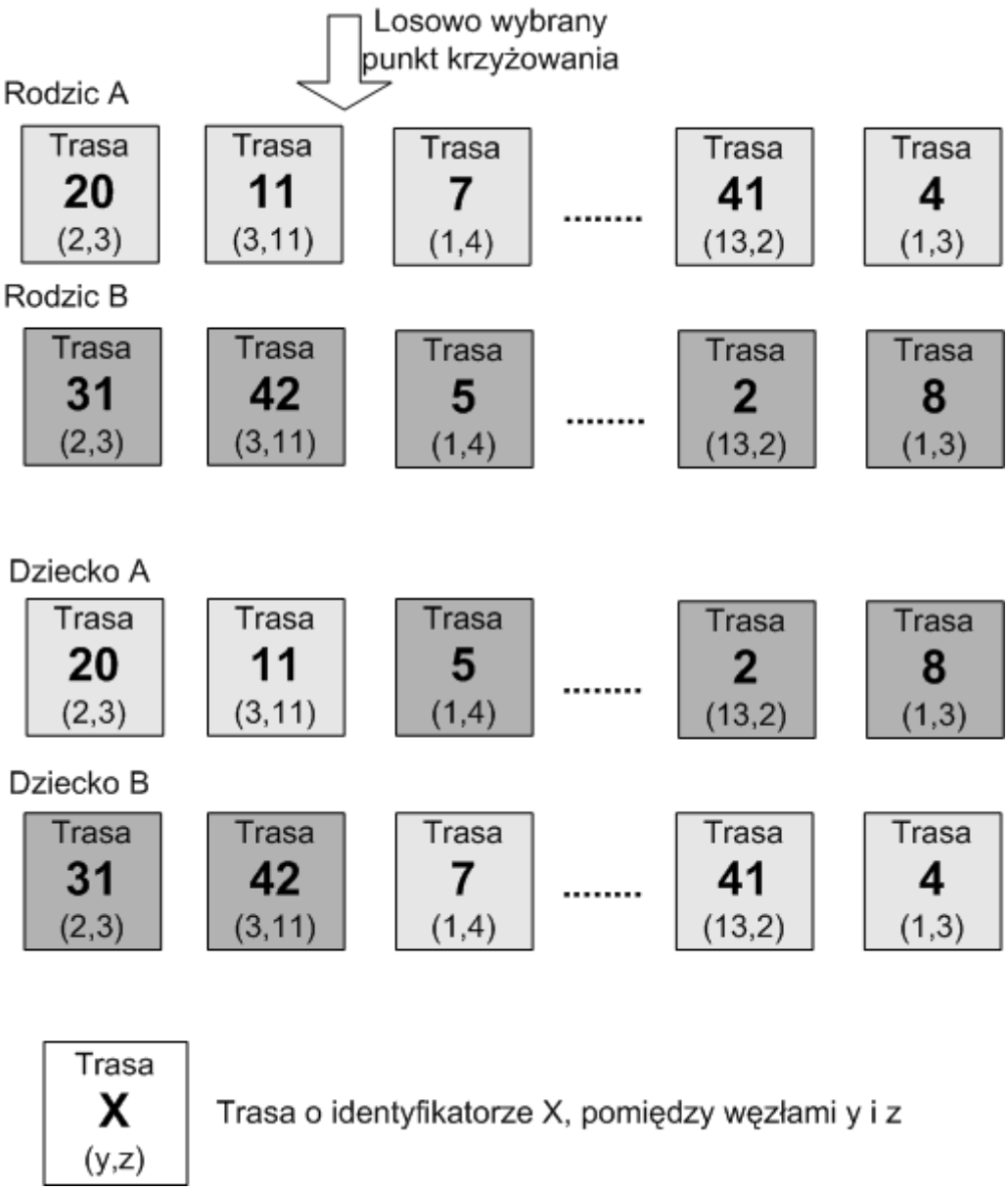
Na tej podstawie definiujemy funkcję opisującą wartość przystosowania osobnika w populacji:

$$F(p) = \frac{1}{LFL'(p) + 1},$$

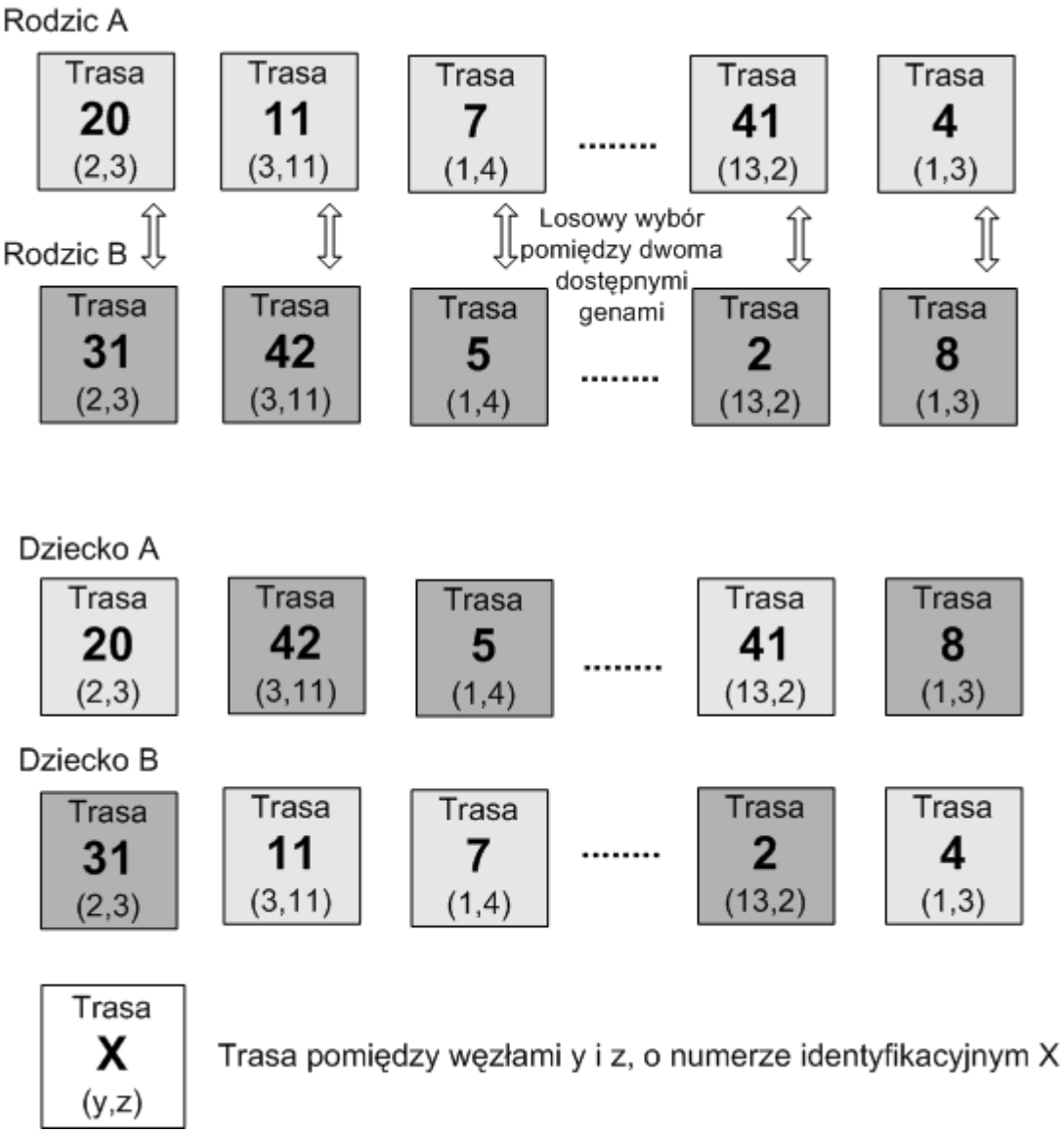
gdzie
p – pojedynczy osobnik w populacji
$LFL'(p)$ - wartość funkcji $LFL'(\underline{f})$, dla rozwiązania zakodowanego przez osobnika p

(33)

„Wysoki” poziom algorytmu HEFAN używa dwóch operatorów krzyżowania: krzyżowania z pojedynczym punktem krzyżowania (ang. *single-point crossover*) i krzyżowania losowego (ang. *uniform crossover*). Wybór, który operator krzyżowania ma zostać użyty jest dokonywany losowo, na podstawie proporcji ustalonych przez użytkownika. W przypadku użycia operatora krzyżowania z pojedynczym punktem krzyżowania, oba osobniki będące rodzicami zostają rozcięte w tym samym, losowo wybranym punkcie, a dzieci powstają poprzez sklejenie ze sobą odpowiednich fragmentów genotypu należących do różnych rodziców. Ta operacja została przedstawiona na Rys. 4. W przypadku użycia operatora krzyżowania losowego, dzieci są tworzone poprzez losowe określenie, które dziecko ma dostać gen od którego rodzica. Ta operacja została przedstawiona na Rys. 5.



Rys. 4. Krzyżowanie osobników na poziomie „wysokim” – pojedynczy punkt krzyżowania



Rys. 5. Krzyżowanie osobników na poziomie „wysokim” – krzyżowanie losowe

Mutacja osobników na poziomie „wysokim” odbywa się poprzez losowe wybranie genu (trasy) i wymienienie go na inny, losowo wybrany gen (trasę) posiadający ten sam węzeł startowy i końcowy jak mutowany gen. Nowy gen (trasa) jest proponowany przez Bazę tras, bez użycia funkcji *L(k,p)*, opisanej w punkcie 2.4.5 (prawdopodobieństwo wybrania trasy jest identyczne dla wszystkich tras spełniających ograniczenie dotyczące węzła startowego i końcowego).

2.4.5 „Niski” poziom algorytmu

„Niski” poziom algorytmu HEFAN jest oparty na podejściu hierarchicznym zaprezentowanym w [83]. Istnienie poziomu „niskiego” nie powoduje, że algorytm jest hierarchiczny w ścisłym sensie tego słowa, jednak wprowadza do niego mechanizmy, które nie są spotykane w typowym algorytmie ewolucyjnym.

Dla operacji wykonywanych na poziomie „niskim” algorytmu, osobnikiem, a więc podmiotem wykonywanych operacji, nie jest propozycja rozwiązania problemu, ale

pojedyncza trasa (która na poziomie „wysokim” reprezentuje pojedynczy gen). Osobnik/trasa jest zakodowana zgodnie z konwencją przedstawioną w punkcie 2.4.2. Identyfikatory węzłów opisujących trasę są traktowane jak geny. W związku z tym należy zauważyć, że liczba genów może być różna dla poszczególnych osobników, oraz że wartość pojedynczego genu/identyfikatora węzła determinuje jakie wartości mogą przyjąć sąsiadujące z nim geny (sąsiadować mogą ze sobą jedynie te geny/identyfikatory węzłów, które posiadają bezpośrednie połączenie pomiędzy sobą).

Pula osobników na poziomie „niskim” nie jest ustalona i może rosnąć, jeśli pojawi się osobnik/trasa, który jeszcze nie znajduje się w Bazie tras (patrz: rozdz. 2.4.3).

Etap krzyżowania osobników na poziomie „niskim” (patrz: Rys. 1) odbywa się zgodnie z procedurą przedstawioną na Rys. 6. Należy zauważyć, że etap krzyżowania na poziomie „niskim” prowadzi do utworzenia nowej populacji na poziomie „wysokim”, oraz do wprowadzenia do Bazy tras, nowych propozycji tras. Używając wartości funkcji przystosowania i metody ruletki wybierany jest jeden osobnik poziomu „wysokiego” (<<1>>). Jeśli los zadecyduje, że krzyżowanie na poziomie „niskim” ma się nie odbyć, to taki osobnik od razu, bez żadnych zmian, jest dodawany do nowej populacji osobników poziomu „wysokiego” (<<8>>). Jeśli krzyżowanie na poziomie „niskim” ma się odbyć, to wybierany jest losowo jeden z genów/tras osobnika wysokiego (<<2>>). Następnie dla wszystkich tras w Bazie tras, których węzeł startowy i końcowy są takie same jak dla wybranego w kroku <<2>> genu/trasy, obliczana jest wartość funkcji przystosowania *L(k,p)*. Wartość funkcji *L*, dla *k-tej* trasy połączenia *p*, obliczana jest dla przepływu zdefiniowanego przez osobnika poziomu „wysokiego” wybranego w kroku <<1>>, pozbawionego trasy dla połączenia *p* wylosowanej w kroku <<2>> z wstawioną zamiast niej trasą *k*, według wzoru:

$$L(k,p)=\frac{1}{\sum_{a\in A}\delta_{pa}^{k}L'(k)},$$

gdzie (34)

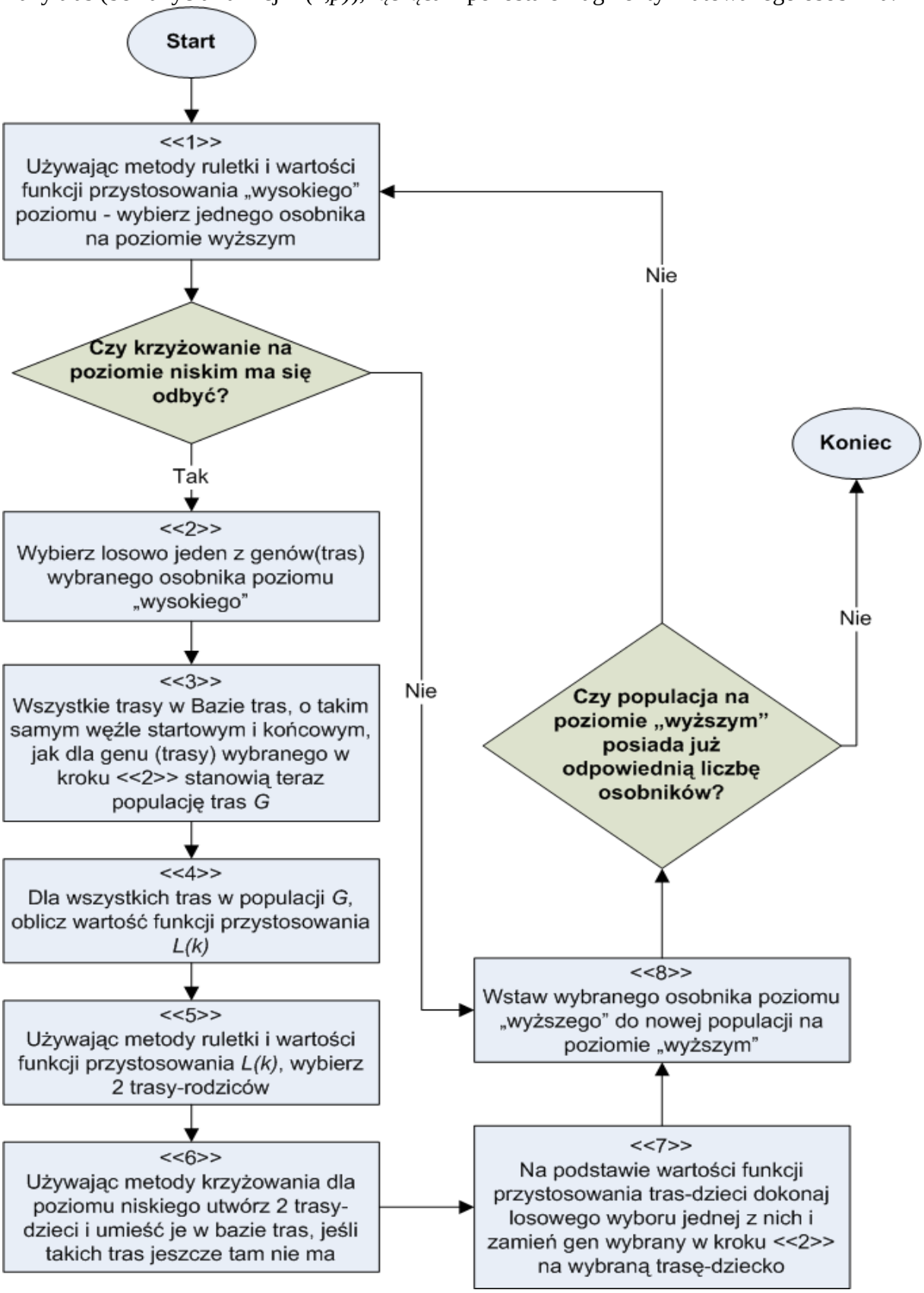
$$L'(k)=\begin{cases}1/(c_a-f_a+1)^2 & dla \quad c_a\geq f_a\\ (f_a-c_a+1) & dla \quad c_a<f_a\end{cases}$$

Wreszcie wszystkich tras w Bazie tras, których węzeł startowy i końcowy są takie same jak dla wybranego w kroku <<2>> genu/trasy wybierane są, metodą ruletki <<5>> dwie trasy-rodzice.

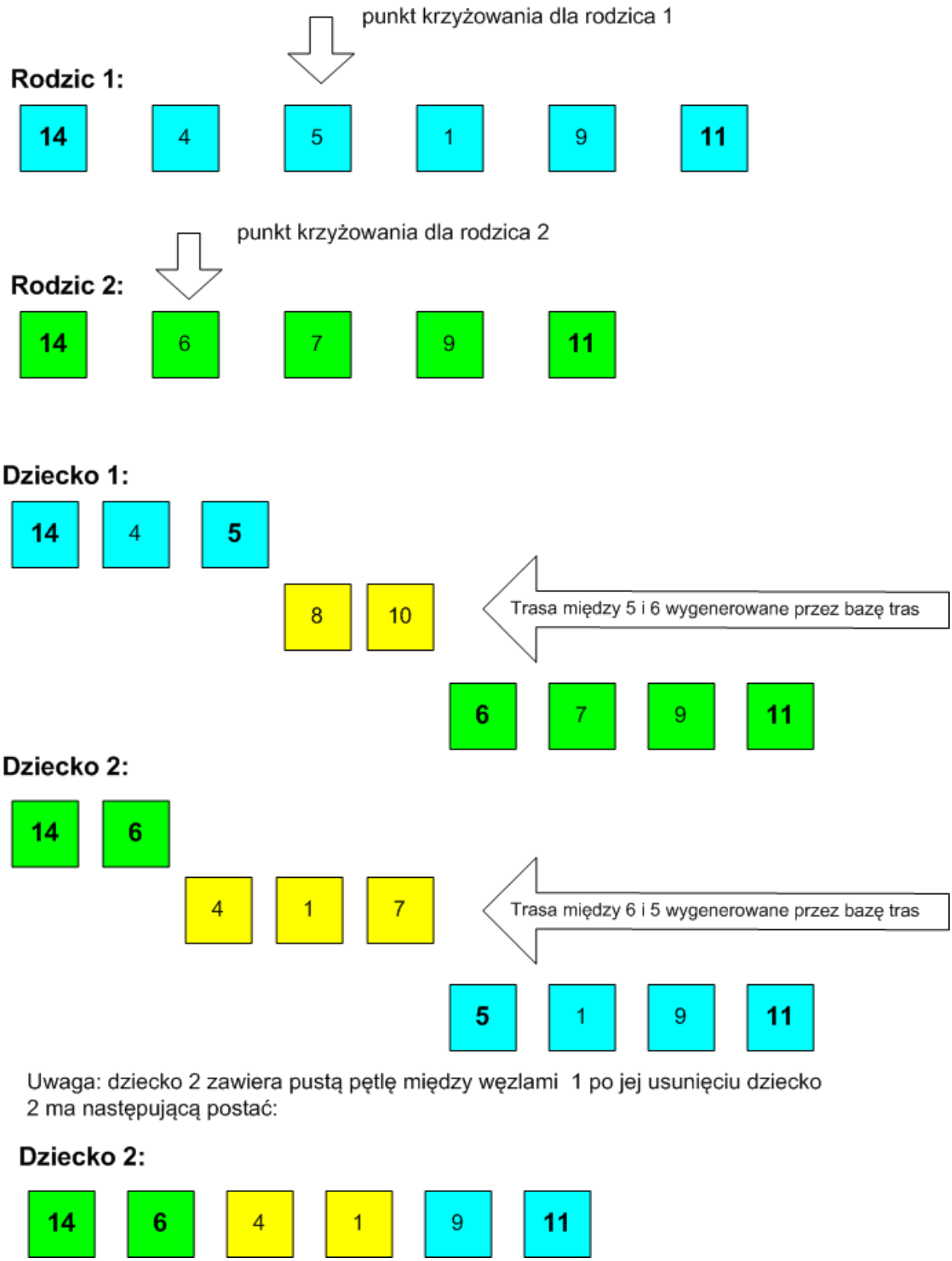
Działanie operatora krzyżowania dwóch tras zostało przedstawione na Rys. 7. Dla obu osobników-rodziców losuje się oddzielnie punkt krzyżowania, a następnie dokonuje próbę sklejenia powstałych fragmentów w sposób analogiczny do tego, który jest stosowany tradycyjnym operatorze krzyżowania. Kolejność genów/identyfikatorów węzłów kodujących trasę nie może być jednak dowolna. Jeśli dwa sklejane fragmenty tras nie pasują do siebie, to ich sklejenie następuje przy pomocy dodatkowej trasy proponowanej przez Bazę tras. Taka dodatkowa trasa służąca sklejaniu dwóch niepasujących do siebie fragmentów tras ma węzły początkowy i końcowy takie, jak nie pasujące do siebie końcówki sklejanych fragmentów tras. Dodatkowa trasa jest wybierana z Bazy tras za pomocą metody ruletki, na podstawie wartości funkcji *L(k,p)*.

W trakcie fazy mutacji na poziomie „niskim”, dla każdego osobnika w populacji na poziomie „wyższym” sprawdzane jest prawdopodobieństwo wystąpienia mutacji na poziomie niskim. Jeśli mutacja na poziomie „niskim” ma wystąpić to wybierany jest losowo gen/trasa/osobnik poziomu „niskiego”, dla którego przeprowadzona zostanie mutacja.

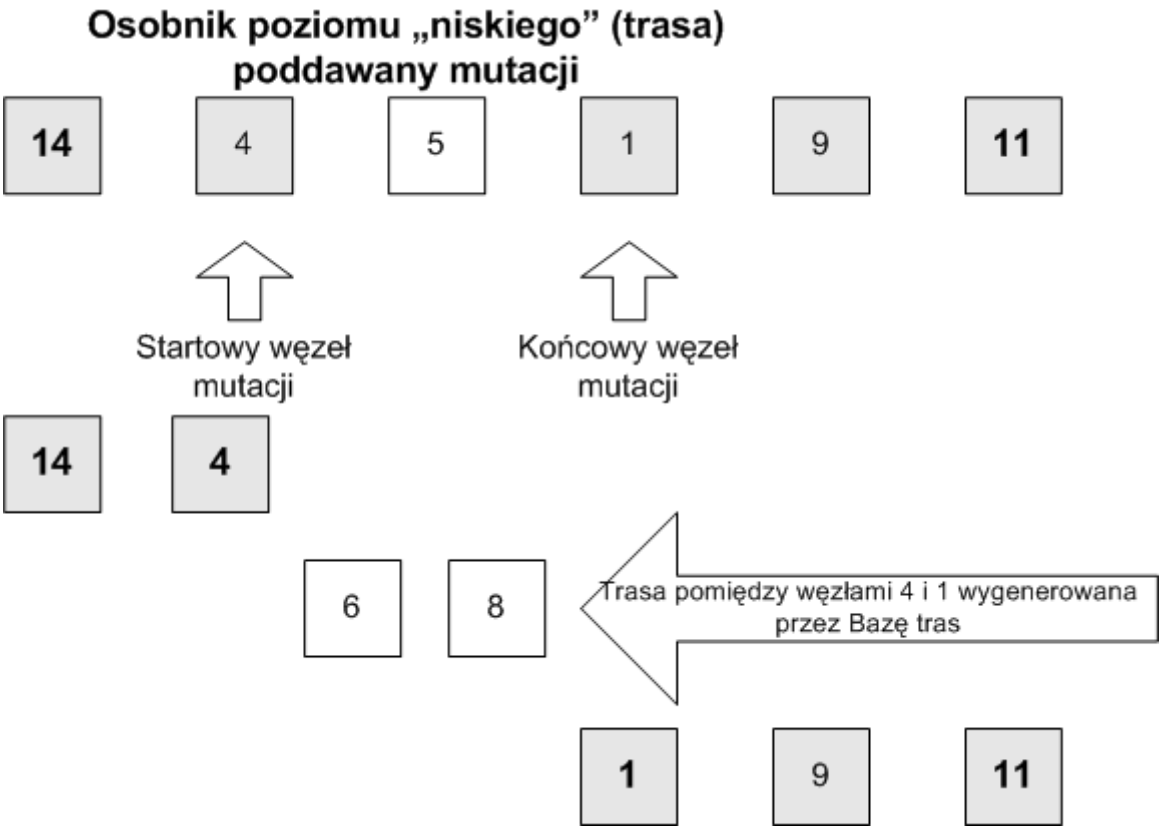
Działanie operatora mutacji na poziomie „niskim” jest przedstawione na Rys. 8. Najpierw dla osobnika (trasy) wybierane są losowo 2 geny (węzły). Wszystkie geny znajdujące się pomiędzy tymi węzłami są usuwane, a na ich miejsce jest wstawiana trasa, losowo wybrana z Bazy tras (bez użycia funkcji *L(k,p)*), łącząca 2 pozostałe fragmenty mutowanego osobnika.



Rys. 6. Procedura krzyżowania na poziomie “niskim”



Rys. 7 Krzyżowanie osobników/tras na poziomie „niskim”



Rys. 8 Mutacja osobników/tras na poziomie „niskim”

2.4.6 Określenie cech rozwiązywanego problemu dla kodowania użytego w algorytmie HEFAN

W niniejszym podrozdziale zostaną opisane te cechy rozwiązywanego problemu, które są zależne od przyjętego sposobu kodowania. Należy zauważyć, że gdyby przyjąć inne metody kodowania problemu poniższy zestaw cech mógłby wyglądać inaczej.

Na podstawie przeprowadzonych badań, opublikowanych w pracach [76, 77, 78], można określić następujące cechy przedstawionego w niniejszej pracy problemu projektowania przepływu w szkieletowych sieciach komputerowych:

A. Problem jest kodowany przy użyciu stosunkowo dużej liczby genów (w przypadku wyników przedstawionych w pracach [76, 77, 78], było to nawet 2500 genów)
B. Problem zawiera schematy o dużej długości definiującej i niskim rzędzie [41, 65] silnie powiązanych ze sobą genów, które są trudne do znalezienia i w praktyce warunkują jakość proponowanego przez algorytm rozwiązania.
C. Znacząca liczba genów przynależy do schematów o niskim rzędzie (również o rzędzie 1), które są łatwe do znalezienia przez algorytm bazujący na idei algorytmu genetycznego.
D. Dla określenia jakości proponowanego rozwiązania problemu niezbędne jest podanie pełnej propozycji rozwiązania. Oznacza to, że nie jest możliwe ocenianie rozwiązań cząstkowych.

Cecha A problemu projektowania przepływu przedstawionego w niniejszej pracy jest faktem i w związku z tym nie podlega dyskusji. Problem pojawia się jednak w kwestii określenia, co należy rozumieć przez dużą, małą, średnią, etc. liczbę genów. Wartościowanie tego typu jest naturalne dla człowieka, jednak ścisła interpretacja takich ocen nie jest zadaniem prostym. W związku z powyższym, na potrzeby niniejszej pracy zostanie przyjęte, że *duża liczba genów to taka, przy której tradycyjny operator krzyżowania staje się zawodny*.

Operacja dostrajania parametrów algorytmu w pracach [77, 78] jednoznacznie wskazywała, że prawdopodobieństwo wystąpienia krzyżowania losowego zamiast krzyżowania z jednym punktem, powinno być znacząco większe niż 0 (w zależności od rodzaju problemu, inicjalizacji z użyciem rozwiązania początkowego i innych okoliczności, prawdopodobieństwo to wahało się od 0,3 do 0,75). Na tej podstawie można wnioskować, że skoro metoda krzyżowania losowego, która ze swej natury preferuje schematy niskiego rzędu (szansa na przetrwanie schematu w przypadku takiego krzyżowania to $2^{-o(S)+1}$, gdzie *o(S)* oznacza rząd schematu *S* [41, 65]), to musi istnieć duża liczba dobrych (posiadających średnie dopasowanie schematu wyższe niż średnie przystosowanie populacji) schematów, dla których małe prawdopodobieństwo przetrwania przy użyciu krzyżowania losowego jest i tak większe, niż w przypadku użycia tradycyjnego operatora z jednym punktem krzyżowania. Są to więc schematy o dużej długości definiującej $\delta(S)$, ale o niskim rzędzie. Na podstawie takiego rozumowania można uznać, że tradycyjny operator krzyżowania stał się zawodny, co prowadzi do spełnienia warunku, przy którym możemy stwierdzić, że problem jest zakodowany przy użyciu dużej liczby genów.

Istnienie **cechy B** problemu można wnioskować na podstawie następujących przesłanek:

- Gdyby w problemie nie istniały schematy o dużej długości definiującej i niskim rzędzie, to należałoby oczekiwać, że w procesie dostrajania algorytmu

przedstawionym w pracach [76, 77, 78] konfiguracje z prawdopodobieństwem krzyżowania dla poziomu „wysokiego" bliskim wartości 0 byłyby równie dobre lub dominujące nad tymi z wysokim prawdopodobieństwem wystąpienia takiej operacji. Przyczyną takiego stanu byłby fakt, że operacja krzyżowania, której celem jest właśnie wymiana grup genów pomiędzy osobnikami byłaby zbędna, a do efektywnych poszukiwań dobrych rozwiązań wystarczałby sam operator mutacji (w tym przypadku do mutacji należy zaliczyć również operacje wykonywane na poziomie „niskim"). W związku z faktem, że w procesie dostrajania otrzymano wartości dla prawdopodobieństwa krzyżowania na poziomie „wysokim" 0,5 w pracach [76, 77], oraz 0,3 i 0,5 w pracy [78], można wnioskować, że istnieją grupy blisko powiązanych genów, których przekazywanie pomiędzy osobnikami w populacji wspomaga operator krzyżowania na poziomie „wysokim".

- Zgodnie z rozumowaniem przedstawionym w uzasadnieniu istnienia cechy A problemu, wysokie wartości prawdopodobieństwa krzyżowania losowego, w wybranych w procesie dostrajania konfiguracjach, pozwalają oczekiwać, że istnieją schematy o dużej długości i niskim rzędzie. Gdyby takie grupy nie istniały należałoby oczekiwać, że operacja krzyżowania losowego będzie zbędna, lub wręcz szkodliwa (pozwala na przetrwanie jedynie schematów o bardzo niskim rzędzie) i nie będzie poprawiać uzyskanych wyników. Prawdopodobieństwo krzyżowania losowego w przedziale od 0,3 do 0,75 uzyskane w procesie dostrajania algorytmu, zaprezentowane w pracach [77, 78], wskazuje, że operacja krzyżowania losowego wywiera pozytywny wpływ na efektywność działania algorytmu HEFAN.
- Dane zaprezentowane w tabelach Tabela 3 i Tabela 4 wskazują, że niewielkie (z punktu widzenia liczby genów) zmiany w genotypie mogą prowadzić do znaczącej poprawy jakości rozwiązania. Biorąc pod uwagę fakt, że zmiany te nie zostały odnalezione przez algorytm LRH można wysunąć uzasadnione przypuszczenie, że są one trudne do znalezienia.

Tabela 3. Odsetek identycznych tras w rozwiązaniach[*] proponowanych przez poszczególne algorytmy (algorytmy były inicjalizowane rozwiązaniem zaproponowanym algorytmu LRH)

	LRH	Standard EA	Hefan 1.0	Hefan 2.2
LRH	X	0.956	0.9736	0.9496
Standard EA	0.956	X	0.9344	0.9142
Hefan 1.0	0.9736	0.9344	X	0.93
Hefan 2.2	0.9496	0.9142	0.93	X

Tabela 4. Średnia wartość względnej poprawy rozwiązania[*] (algorytmy były inicjalizowane rozwiązaniem zaproponowanym przez algorytm LRH). Wartości w tabeli wyliczone na podstawie wzoru (35).

Standard EA	Hefan 1.0	Hefan 2.2
73.9497	23.2069	81.0648

[*] Dotyczy wyników zaprezentowanych w pracy [78]

[*] Dotyczy wyników zaprezentowanych w pracy [78]

$$\sigma = \frac{F(p_{a\lg}) - F(p_{LRH})}{F(p_{LRH})},$$

gdzie

σ - względna poprawa rozwiązania

$F(p)$ - przekształcona wartość funkcji LFL, używana przez algorytm HEFAN (zgodnie ze wzorem (33)) (35)

$p_{a\lg}$ - rozwiązanie proponowane przez algorytm porównywany z algorytmem LRH

p_{LRH} - rozwiązanie proponowane przez algorytm LRH

Tabela 5. Konfiguracje poszczególnych algorytmów* użytych do optymalizacji rozwiązania zaproponowanego przez algorytm LRH

Algorytm	Konfiguracja: krzyżowanie "wysokie" / mutacja "wysoka"/ krzyżowanie losowe / krzyżowanie „niskie" / mutacja „niska"	Czas obliczeń dla komputera AMD 3800+ 64 2GB RAM [s]	Rozwiązanie początkowe	Inicjalizacja przy użyciu *s* najkrótszych tras
Standard EA	0,3 / 0,2 / 0,5 / nd / nd	360	LRH	4
HEFAN 1.0	0,3 / 0,1 / 0,0 / 0,5 / 0,4	360	LRH	4
HEFAN 2.2	0,3 / 0,2 / 0,3 / 0,5 / 0,4	360	LRH	4

Cecha C problemu może istnieć lub nie w zależności od przyjętego kryterium oceny rozwiązania problemu projektowania przepływu, topologii sieci, oraz poziomu jej obciążenia. W niniejszej pracy prezentujemy badania, w których sieć jest znacząco obciążona, tzn. że tylko znikoma część rozwiązań jest dopuszczalna (oszacowanie jaka dokładnie część możliwych rozwiązań jest dopuszczalna jest trudne do przeprowadzenia ze względów praktycznych) ze względu na ograniczenie związane z przepustowością łuków sieci. W związku z tym należy oczekiwać, że proponowane rozwiązania będą zwykle używały najkrótszych możliwych do zestawienia tras pomiędzy węzłami sieci, szczególnie tam, gdzie możliwe do zestawienia trasy są stosunkowo krótkie (mają długość nie większą niż 3). Oznacza to, że dla pewnej liczby zestawianych tras wybór będzie bardzo wąski, często ograniczony do wyboru jednej „oczywistej" trasy (np. tam gdzie istnieje bezpośrednie połączenie pomiędzy dwoma węzłami). Oczekiwania te zostały potwierdzone przez wyniki uzyskane w ramach badań zaprezentowanych w pracach [76, 77, 78].

Wyniki zaprezentowane w tabeli

Tabela 6 pokazują, że jeśli porównywać parami rozwiązania zaproponowane przez 4 różne algorytmy, to będą one posiadały nie mniej niż 21% wspólnych tras. W przypadku algorytmów Standard EA, HEFAN 1.0 i HEFAN 2.2 oznacza to również, że genotypy osobników reprezentujących rozwiązania są parami identyczne w co najmniej 21%. Można oczekiwać, że część z tych genów, to

* Dotyczy wyników zaprezentowanych w pracy [78]

schematy o niskim rzędzie, które są łatwe do znalezienia przez algorytm genetyczny.

Tabela 6. Odsetek identycznych tras w rozwiązaniach[*] proponowanych przez poszczególne algorytmy, algorytmy nie były inicjalizowane rozwiązaniem początkowym.

	LRH	Standard EA	Hefan 1.0	Hefan 2.2
LRH	X	0,269	0,3014	0,3674
Standard EA	0,269	X	0,2182	0,2348
Hefan 1.0	0,3014	0,2182	X	0,24
Hefan 2.2	0,3674	0,2348	0,24	X

Tabela 7. Konfiguracje poszczególnych algorytmów[†] zaprezentowanych w tabeli Tabela 6

Algorytm	Konfiguracja: krzyżowanie "wysokie" / mutacja "wysoka"/ krzyżowanie losowe / krzyżowanie „niskie" / mutacja „niska"	Czas obliczeń dla komputera AMD 3800+ 64 2GB RAM [s]	Rozwiązanie początkowe	Inicjalizacja przy użyciu *s* najkrótszych tras
Standard EA	0,5 / 0,2 / 0,5 / nd / nd	760	Brak	4
HEFAN 1.0	0,5 / 0,1 / 0,0 / 0,5 / 0,4	760	Brak	4
HEFAN 2.2	0,5 / 0,2 / 0,5 / 0,5 / 0,4	760	Brak	4

Cecha D problemu projektowania przepływu przedstawionego w niniejszej pracy jest faktem i w związku z tym nie podlega dyskusji. Należy jednak zauważyć, że jest to cecha wielu innych problemów obliczeniowych.

Zdaniem autora niniejszej pracy, cechy A, B, C i D, wyróżnione powyżej, można uznać za reprezentatywne cechy problemów, które są realnym wyzwaniem w zastosowaniach praktycznych. Należy jednak pamiętać, że rozważanie istnienia powyższych cech ma sens jedynie wtedy, gdy do rozwiązania problemu używana jest metoda korzystająca z kodowania za pomocą genów.

2.4.7 Wnioski z działania algorytmu HEFAN – motywacje dla dalszych prac

Przeprowadzone przy wykorzystaniu algorytmu HEFAN badania [76, 77, 78], oraz cechy problemu wyszczególnione w rozdziale 2.4.6 były dla autora motywacją do dalszych prac badań. Poniżej wymienione są główne idee, które były podstawą dla zaproponowania nowej metody szablonu pracy dla algorytmu genetycznego.

Motywacja 1 – przeglądanie podprzestrzeni. Jedną z cech problemu jest istnienie schematów o dużej długości definiującej, ale niskim rzędzie. Schematy te są „trudne do znalezienia" dla metody bazującej na idei algorytmu genetycznego i w praktyce warunkują jakość proponowanego rozwiązania (cecha B). W związku z powyższym wydaje się pożądane, aby nowa metoda dawała możliwość efektywnego przeglądania podprzestrzeni

[*] Dotyczy wyników zaprezentowanych w pracy [78]
[†] Dotyczy wyników zaprezentowanych w pracy [78]

rozwiązań dla już istniejących rozwiązań w celu ich poprawy. Dobrą propozycją wydaje się być przypisanie do każdego pełnego rozwiązania grupy osobników kodujących jedynie fragment pełnego rozwiązania. Osobniki takie będą nazywane w niniejszej pracy „wirusami" (patrz: rozdział 3.5.1). Wirusy będą oceniane w taki sam sposób jak normalne osobniki (zawierające pełne rozwiązanie), wartości dla niezdefiniowanych genów będą takie, jak w normalnym osobniku, do którego dany wirus jest przypisany. W przypadku gdyby dany wirus posiadał lepszą wartość funkcji przystosowania niż „normalny" osobnik, do którego jest przypisany, odpowiednie geny w genotypie „normalnego" osobnika będą zastępowane genami wirusa.

Należy zauważyć, że takie podejście nie tylko umożliwia przeglądanie podprzestrzeni w celu poprawy wartości przystawania poszczególnych „normalnych" osobników, ale również spełnia warunek nakładany przez cechę D – ocenie podlegają jedynie pełne rozwiązania.

Należy również zauważyć, że powyższe warunki spełnia znana z literatury propozycja kodowania nieporządnego (ang. *messy coded*) używana w nieporządnym algorytmie genetycznym (ang. messy GA) i szybkim nieporządnym algorytmie genetycznym (ang. *fast messy GA*) [34, 35, 36] zaproponowanymi przez Davida Goldberga. Jednak idea, zgodnie z którą powyższe kodowanie jest wykorzystywane, jest różna dla obu metod. W pracach Goldberga nieporządnie zakodowane osobniki (ang. *messy coded individual*) są wykorzystywane jako nośniki pozwalające na sklejanie tzw. bloków budujących (ang. *building blocks*) uzyskanych we wcześniejszych fazach metody. Tymczasem proponowana w nieniejszej pracy metoda ma na celu przeglądanie podprzestrzeni i optymalizację już istniejących rozwiązań. Pełny opis różnic pomiędzy proponowaną przez autora metodą, a pracami Davida Goldberga, oraz opis metod mGA i fmGA znajduje się w rozdziałach 3.3 i 3.5.

Motywacja 2 – zapamiętywanie zależności między genami. Jeżeli w ramach używanego kodowania istnieją schematy o dużej długości definiującej i niskim rzędzie (cecha B), oraz jeżeli kodowanie wymaga dużej liczby genów (cecha A), to oznacza to, że tradycyjne operatory krzyżowania będą tym mniej skuteczne, im dłuższe będzie kodowanie całego problemu i im dłuższe będą długości definiujące oraz rzędy schematów warunkujących jakość rozwiązania. Należy więc odpowiedzieć na pytanie czy i jak należy wymieniać geny pomiędzy „normalnymi" (zawierającymi pełne rozwiązania) osobnikami. Biorąc pod uwagę rozmiar problemu nie wydaje się rozsądne ograniczanie się do zaledwie jednego „normalnego" osobnika z przypisaną do niego populacją wirusów – istnieje ryzyko, że taki osobnik utknie w obszarze lokalnego optimum, z którego nie będzie w stanie się wydostać. Skoro w populacji pełnych rozwiązań powinno istnieć więcej niż jedno rozwiązanie w celu uniknięcia przedwczesnej zbieżności i utknięcia w obszarze optimum lokalnego, to muszą istnieć mechanizmy wymiany danych pomiędzy takimi osobnikami, a skoro tradycyjne operatory krzyżowania nie będą w tym zakresie skuteczne, niezbędna jest alternatywna propozycja uwzględniająca potencjalne zależności pomiędzy genami.

Istnienie, bądź brak silniejszych, lub słabszych zależności pomiędzy poszczególnymi genami w genotypie jest kwestią zależną od wielu czynników. Przede wszystkim należy pamiętać, że zarówno w przypadku rozpatrywanego problemu, jak i innych problemów występujących w zastosowaniach praktycznych trudno oczekiwać, aby jakaś część problemu była w pełni separowalna. Ponadto silna zależność wewnątrz jakiejś grupy genów może być warunkowana przez kontekst (wartości przyjmowane przez pozostałe geny w genotypie). Niewątpliwie jednak próba zdobycia nawet nieprecyzyjnych informacji o potencjalnych zależnościach pomiędzy genami jest lepsza od zupełnego braku takiej informacji i oparciu działania operatorów krzyżowania wyłącznie o pozycję genu w genotypie.

Należy zauważyć, że zaproponowane wcześniej wirusy to narzędzie służące do przeszukiwania podprzestrzeni przestrzeni rozwiązań i wyszukiwania trudnych do znalezienia schematów o niskim rzędzie. Mechanizm zdobywania informacji o potencjalnych zależnościach pomiędzy genami można więc oprzeć o założenie, że grupa genów, która jest wymieniana jednorazowo przez wirusa to grupa genów, które są od siebie bardziej zależne niż inne. Jeśli dla danej grupy genów takie stwierdzenie jest prawdziwe (a także jeśli zależność wewnątrz danej grupy genów jest mało zależna od ich kontekstu), to można oczekiwać, że użycie takiej informacji o zwiększonej zależności wewnątrz danej grupy genów spowoduje efektywniejszą wymianę genów pomiędzy „normalnymi" osobnikami. W takiej sytuacji znaczenie informacji o zależności wewnątrz danej grupy genów powinno zostać zwiększone i w efekcie częściej używane dla kolejnych operacji wymiany genów pomiędzy „normalnymi" osobnikami. W przypadku gdyby informacja o silnych wzajemnych powiązaniach w obrębie danej grupy genów była nieprawdziwa lub nieużyteczna, należy oczekiwać, że użycie takiej informacji nie spowoduje efektywniejszej wymiany genów pomiędzy „normalnymi" osobnikami. W takim przypadku znaczenie informacji o zależności wewnątrz danej grupy genów powinno zostać zmniejszone i w efekcie rzadziej używane dla kolejnych operacji wymiany genów pomiędzy „normalnymi" osobnikami.

Powyższe motywacje połączone z analizą nowego wykorzystania możliwości kodowania nieporządnego stanowiły podstawę dla nowej metody zaprezentowanej w następnym rozdziale.

3. MuPPetS – propozycja nowego szablonu pracy dla algorytmów bazujących na idei algorytmu genetycznego

W niniejszym rozdziale przedstawiony jest algorytm MuPPetS (Multi Population Pattern Searching Algorithm), który stanowi propozycję nowego szablonu dla działania algorytmu genetycznego. Algorytm MuPPetS został zaprojektowany ze szczególnym uwzględnieniem problemów posiadających cechy wskazane w rozdziale 2.4.6. oraz na podstawie motywacji wskazanych w rozdziale 2.4.7.

Dla testów proponowanego nowego szablonu algorytmu genetycznego zostało wybrane kodowanie binarne. Przyczyną takiego wyboru jest fakt, że algorytmy genetyczne używające kodowania binarnego są bogato przedstawione w literaturze problemu. Niemniej jednak przedstawiony w niniejszym rozdziale algorytm MuPPetS został tak zaprojektowany, aby możliwa była jego łatwa adaptacja do innego rodzaju kodowania.

Efektyność algorytmu MuPPetS zostanła porównana z algorytmami fmGA (ang. fast messy Genetic Algorithm), oraz BOA (ang. *Bayesian Optimization Algorithm*). Zaprezentowane w niniejszej pracy wyniki badań dotyczące porównania efektywności algorytmów MuPPetS, BOA i fmGA, zostały opublikowane w pracy [57]. Algorytm fmGA został wybrany do testów, ponieważ algorytm MuPPetS w istotnym stopniu nawiązuje do idei kodowania używanej przez fmGA, a także używa niektórych jego operatorów. Algorytm BOA [72, 73] został wybrany, ponieważ stanowi stosunkowo nową metodę z zakresu algorytmów uczących się powiązań pomiędzy genami (ang. *linkage learning*) [11, 34, 35, 36, 39, 40], a przede wszystkim wydaje się być algorytmem cechującym się wysoką efektywnością [13, 47]. Jednym z algorytmów, który również był brany pod uwagę jako konkurent dla algorytmu MuPPetS w testach wydajnościowych, był algorytm LLGA [39][40], który jest ciekawą propozycją bazującą na nadspecyfikacji. Używana przez algorytm konwencja kodowania EPE-n, powoduje, że każdy gen poza jedną swoją kopią na najbardziej znaczącej pozycji, która jest wyrażona w fenotypie, posiada co najmniej n swoich kopii dla każdej swojej możliwej wartości, która w fenotypie wyrażona nie jest. Oznacza to że dla problemu kodowanego przy użyciu 2500 genów, gdzie każdy gen może przyjąć co najmniej 20 różnych wartości, osobniki w algorytmie LLGA, używającym kodowania EPE-2, byłyby kodowane na, co najmniej, 100 000 genów. Ten fakt oznacza, że dla problemów praktycznych wymagających długiego kodowania algorytm LLGA może być trudny w użyciu i z tego względu porównanie algorytmu MuPPetS z algorytmem LLGA nie zostało wykonane.

3.1 Wprowadzenie

Na potrzeby niniejszej pracy, za pracą [22] zostaje przyjęta następująca klasyfikacja. Wszystkie metody bazujące na idei ewolucji z jednoczesnym wykorzystaniem mocy obliczeniowej komputera noszą nazwę Obliczeń Ewolucyjnych (ang. *EC – Evolutionary Computation*), a każdy algorytm należący do tej grupy jest Algorytmem Ewolucyjnym (ang. *Evolutionary Algorithm*). Algorytmy Ewolucyjne można podzielić na: Algorytmy Genetyczne (ang. *GA – Genetic Algorithm*), Strategie Ewolucyjne (ang. *ES - Evolutionary Strategy*) i Programowanie Ewolucyjne (ang. *EP – Evolutionary Programming*). Na potrzeby niniejszej pracy przyjmuje się, że aby uznać algorytm za genetyczny musi on posiadać następujące cechy:

- Bazować na populacji osobników
- Osobnik w populacji stanowi zakodowane rozwiązanie problemu
- Osobnik jest zakodowany w postaci łańcucha genów
- Geny mogą przyjmować wartości zgodnie z ustalonego zakresu
- Osobniki w populacji muszą być poddawane procesowi selekcji w celu wyłonienia rodziców, na podstawie których zostanie utworzona następne pokolenie
- Selekcja jest procesem niedeterministycznym, jednak lepsze osobniki są preferowane

Należy zauważyć, że zgodnie z przyjętą powyżej klasyfikacją rozróżnienie pomiędzy algorytmem genetycznym, a innym nie bazuje na przyjętym sposobie kodowania. A więc algorytm może być genetyczny zarówno, jeśli osobniki są zakodowane binarnie, przy pomocy liczb rzeczywistych, lub w inny sposób.

Oddzielną grupę stanowią Strategie Ewolucyjne. Podstawowe różnice pomiędzy algorytmem genetycznym a strategią ewolucyjną to:

- Każdy osobnik w populacji zostaje rodzicem i posiada zadaną, tę samą, liczbę dzieci
- Proces selekcji następuje po reprodukcji (a nie przed jak w algorytmie genetycznym)
- Proces selekcji jest deterministyczny – w populacji zostaje zadana liczba osobników, tych które są najlepsze spośród zbiorczej populacji rodziców i dzieci, lub tylko dzieci
- Prawdopodobieństwo mutacji i krzyżowania jest elementem chromosomu poszczególnych osobników
- Niedopuszczalne rozwiązania są usuwane z populacji (w przypadku GA na rozwiązania takie nakłada się kary lub się je naprawia)

3.2 Problemy testowe, funkcje zwodnicze

Funkcje zwodnicze (ang. *decepteive function*) zostały zaproponowane jako narzędzie testowe dla algorytmów genetycznych, które używają kodowania binarnego [15]. Są one tak skonstruowane, aby odnalezienie optymalnego rozwiązania było szczególnie trudne dla algorytmów bazujących na idei algorytmu genetycznego. Jako problem testowy będą używane konkatenacje poszczególnych funkcji zwodniczych o długości trzy i pięć. Na potrzeby niniejszej pracy przyjęto założenie, że nie jest możliwa ocena rozwiązania częściowego, a więc do oceny rozwiązania niezbędne jest podanie pełnego rozwiązania (patrz: rozdz. 2.4.6, cecha D).

Wartość funkcji zwodniczej zależy od liczby jedynek w chromosomie (ang. *unitation*):

$f(x):\{0,1\}^l \to \Re$, gdzie l to długość chromosomu

Aby funkcja *f(l)* była zwodnicza, musi spełnić następujące warunki [15]:

$$f(l) > \max[f(0), f(1)] \tag{36}$$

$$f(l) > \max[f(2), f(l) + f(l-1) - f(1)] \tag{37}$$

$$f(i) \geq f(j) \quad \text{dla} \quad 1 \leq i \leq [l/2] \quad \text{oraz} \quad i < j < l-1 \tag{38}$$

Na przykład, dla *l=3*, funkcja, która w zależności od liczby „1” w chromosomie przyjmuje wartości: *f*(0)=0,68, *f*(1) = 0,33, *f*(2)=0, i *f*(3)=1 jest zwodnicza. Zwyczajowo przyjmuje się, że wartość optymalną funkcja zwodnicza osiąga dla samych ‘1’, a optimum lokalne posiada w przypadku, gdy argument złożony jest z samych ‘0’.

Do budowy problemów testowych, użytych zostało 8 różnych funkcji bazowych. Wszystkie funkcje bazowe są funkcjami zwodniczymi i zostały przedstawione w tabeli Tabela 8.

Tabela 8. Wartości przyjmowane przez funkcje bazowe, użyte do konstrukcji problemów testowych, w zależności od liczby ‘1’ w chromosomie

Liczba ‘1’ w chromosomie	3-bit low (3*l*)	3-bit low hard (3*lh*)	3-bit high (3*h*)	3-bit high hard (3*hh*)	5-bit low (5*l*)	5-bit low hard (5*lh*)	5-bit high (5*h*)	5-bit high hard (5*hh*)
0	0,33	0,98	3,33	9,80	0,4	0,99	4	9,88
1	0,17	0,49	1,67	4,90	0,3	0,74	3	7,41
2	0	0	0	0	0,2	0,49	2	4,94
3	1	1	10	10	0,1	0,25	1	2,47
4					0	0	0	0
5					1	1	10	10

Funkcje bazowe zostały podzielone na dwie klasy biorące pod uwagę dwa różne kryteria:

- „hard”/”not hard” na podstawie różnicy wartości funkcji w optimum globalnym i w optimum lokalnym
- „low”/”high” w zależności od wartości przyjmowanych przez całą funkcję bazową

Jeśli funkcja jest określona jako „hard”, oznacza to, że różnica w wartości pomiędzy optimum globalnym a optimum lokalnym takiej funkcji jest niewielka. W związku z tym, nawet jeśli optimum dla takiej funkcji zostanie znalezione, to osobniki przyjmujące optymalną wartość osiągają bardzo niewielką przewagę nad osobnikami, które przyjmują wartość optimum lokalnego. W konsekwencji schematy zawierające optymalne rozwiązanie dla omawianej funkcji bazowej będą znacząco gorzej rozprzestrzeniać się w populacji.

Różnica pomiędzy funkcjami bazowymi typu „low” i „high” polega na tym, że funkcje typu „high” przyjmują 10-krotnie większe wartości niż analogiczne funkcje typu „low”.

Problemy testowe, dla których rozwiązania miały proponować testowane algorytmy, zostały utworzone poprzez konkatenację funkcji bazowych. Utworzone zostały problemy o długościach: 30, 50, 150 i 300 bitów. Te problemy zostały podzielone na 2 klasy:

- „normal”/”hard” w zależności od tego, czy problem testowy zawiera funkcje bazowe typu „hard”, czy nie
- „flat”/”spike” w zależności od tego, czy problem testowy zawiera wyłącznie funkcje typu „low”, czy też „high” i „low”

Tabela 9. Problemy testowe

Długość/typ	flat normal	flat hard	spike normal	spike hard
30 (only 3)	10*3l	10*3lh	3*3l + 7*3h	3*3lh + 7*3hh
*50 (only 3)	17*3l	17*3lh	5*3l + 12*3h	5*3lh + 12*3hh
50 (3 and 5)	10*3l + 4*5l	10*3lh + 4*5lh	3*3l + 7*3h + 3*5h + 1*5l	3*3lh + 7*3hh + 3*5hh + 1*5lh
50 (only 5)	10*5l	10*5lh	3*5l + 7*5h	3*5lh + 7*5hh
150 (only 3)	50*3l	50*3lh	15*3l + 35*3h	15*3lh + 35*3hh
150 (3 and 5)	40*3l + 6*5l	40*3lh + 6*5lh	12*3l + 28*3h + 2*5l + 4*5h	12*3lh + 28*3hh + 2*5lh + 4*5hh
150 (only 5)	30*5l	30*5lh	9*5l + 21*5h	9*5lh + 21*5hh
300 (only 3)	100*3l	100*3lh	30*3l + 70*3h	30*3lh + 70*3hh
300 (3 and 5)	50*3l + 30*5l	50*3lh + 30*5lh	15*3l + 35*3h + 9*5l + 21*5h	15*3lh + 35*3hh + 9*5lh + 21*5hh
300 (only 5)	60*5l	60*5lh	18*5l + 42*5h	18*5lh + 42*5hh

* - konkatenacja złożona z 17 funkcji 3-bitowych jest, oczywiście, zakodowana na 51 bitach

Dodatkowo, w części przeprowadzonych eksperymentów, wyżej opisanym problemom testowym został doklejony „ogon”. Długość „ogona” jest równa oryginalnej długości problemu. Na przykład: dla problemów150-bitowych długość ogona wynosi 150-bitów, a dla problemów 30-bitowych – 30 bitów. W ten sposób długość problemu, do którego doklejony został „ogon”, podwaja się. Wartość funkcji stanowiącej „ogon” problemu jest obliczana według wzoru:

$$T_l(u) = \frac{u}{l}, \tag{39}$$

gdzie l jest długością funkcji “ogon”, a u jest liczbą ‘1’

Należy zauważyć, że funkcja „ogon” jest łatwa do rozwiązania dla każdego algorytmu bazującego na idei algorytmu genetycznego – nie posiada optimów lokalnych, a dodatkowo można ją traktować jak w pełni separowalną konkatenację funkcji o długości 1. W ramach prowadzonych badań problemy testowe zostały rozszerzone o funkcję łatwą do rozwiązania, ponieważ rozwiązania dla problemów spotykanych w praktyce, często posiadają „oczywistą” (a więc właśnie łatwą do rozwiązania na takiej zasadzie jak w przypadku funkcji typu „ogon”) część. Należy również zauważyć, że wzbogacenie problemów testowych funkcjami typu „ogon” można uznać za symulację cechy C wskazanej w rozdziale 2.4.6.

3.3 Algorytmy Messy i Fast Messy GA - opis

Algorytmy mGA (ang. *Messy Genetic Algorithm*) i fmGA (ang. *Fast Messy Genetic Algorithm*) zostały zaproponowane przez Davida Goldberga [34, 35, 36]. Jedną z ich głównych cech charakterystycznych jest sposób reprezentacji osobników – niektóre geny

mogą być reprezentowane wiele razy, a inne wcale. Dodatkowo, pozycja genu w genotpyie jest niezwiązana pozycją w fenotypie reprezentowaną przez dany gen.

3.3.1 Algorytm mGA

W algorytmie mGA chromosom jest reprezentowany jako lista par: [(*p*, *v*) (*p*, *v*) (*p*, *v*) (*p*, *v*)], gdzie *p* i *v* są odpowiednio, pozycją i wartością genu. Przykład osobnika zakodowanego za pomocą algorytmu mGA dla chromosomu o długości 10 genów jest podany poniżej:
- genotyp: [**(5,1)** (2,0) **(1,0)** (2,0) **(2, 1) (9, 0)**]

Powyższy genotyp definiuje następujący fenotyp:
- fenotyp: **01**??**1**???**0**?

Jak widać, niektóre geny mogą być nadreprezentowane (ang. *overspecification*) (gen numer 2 posiada aż 3 reprezentacje w genotypie), a niektóre geny w ogóle nie posiadają swojej reprezentacji (ang. *underspecification*) (geny: trzeci, czwarty, szósty, siódmy, ósmy i dziesiąty). Jeśli gen jest reprezentowany w genotypie, to wartość genu w fenotypie jest określana przez jedną ze wszystkich reprezentacji genu. Przyjmujemy, że jest to reprezentacja najbliższa końca genotypu. Dlatego w powyższym przykładzie wartość genu numer 2 w fenotypie wynosi 1, a nie 0.

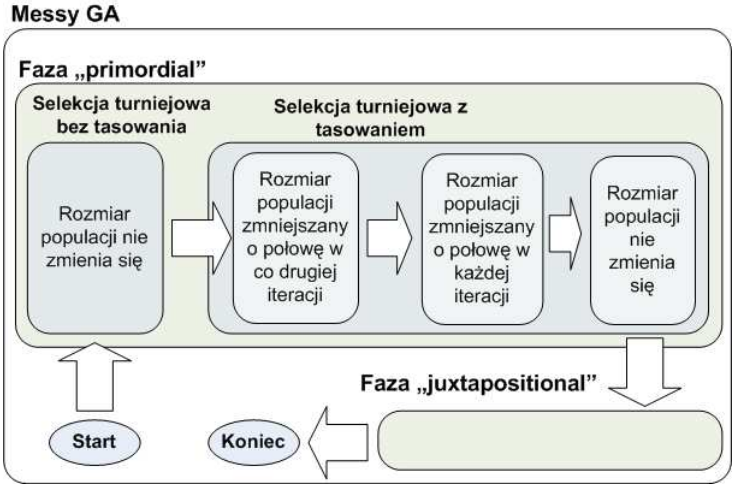
Należy zauważyć, że w literaturze przedmiotu [34, 35, 36] można znaleźć odmienną regułę określania wartości genów w fenotypie niż ta przedstawiona powyżej. Reguła ta (ang. *first come, first served*) mówi, że wartość fenotypowa jest określana przez gen najbliższy początkowi genotypu. Obie możliwe reguły są równoważne i nie mają żadnego wpływu na działanie algorytmu, pod warunkiem, że wybrana reguła jest konsekwentnie używana w całym algorytmie.

W przypadku genów, które nie są reprezentowane w genotypie, pojawia się pytanie, jak mają być one reprezentowane w fenotypie. Goldberg [35] sugeruje kilka możliwych rozwiązań tego problemu. Pierwsza propozycja zakłada, że wartość funkcji przystosowania może zostać określona dla rozwiązania cząstkowego. Druga propozycja zakłada użycie szybkiego algorytmu przeszukiwania lokalnego dla uzupełnienia brakujących w fenotypie genów. Trzecia propozycja bazuje na tzw. szablonach (ang. *Competetive Template*), zwanych dalej w niniejszej pracy „CT”. CT są pełnymi propozycjami rozwiązań dla rozwiązywanego problemu i są zakodowane tak, jak osobniki w klasycznym algorytmie genetycznym. Na przykład, jeśli CT zawiera jedynie zera, to fenotyp osobnika z podanego powyżej przykładu będzie ostatecznie wyglądał następująco: **01**00**1**00**0**00. Należy zauważyć, że jednym z założeń dla proponowanego w ramach niniejszej pracy algorytmu jest, że oceniać można jedynie pełne rozwiązania (patrz: rozdz. 2.4.6, cecha D), oraz że rozwiązania wymagają długiego kodowania (cecha A). Z tych powodów treść niniejszej pracy, będzie koncentrować się na ostatniej propozycji radzenia sobie z brakiem reprezentacji niektórych genów w genotypie – użyciu CT.

Algorytm mGA dzieli się na dwie fazy – fazę „primordial” oraz „juxtapositional”. Faza „primordial” ma za zadanie wyprodukowanie dobrych bloków budujących (patrz: rozdz. 3.1) o zadanej długości. Faza „juxtapositional” ma na celu przetworzenie bloków budujących zaproponowanych w fazie „primordial” i skonstruowanie pełnego rozwiązania problemu.

W fazie „primordial” algorytm mGA poszukuje dobrych propozycji dla bloków budujących. Poszukiwanie jest realizowane poprzez stworzenie na początku fazy „primordial”

wszystkich możliwych ciągów genów o długości *k*, określonej przez użytkownika. Te ciągi mogą być interpretowane przez algorytm mGA jako osobniki w kolejnej fazie. Liczba ciągów *s*, która jest produkowana w fazie „primordial”, w przypadku kodowania binarnego, wynosi: $s = \binom{k}{l} * 2^k$, gdzie l jest liczbą genów niezbędną do zakodowania pełnej propozycji rozwiązania problemu, a *k* jest wymaganą długością ciągów, określaną przez użytkownika algorytmu. Po utworzeniu wszystkich możliwych ciągów/osobników o długości *k*, wygenerowana w ten sposób populacja algorytmu mGA jest przetwarzana w kolejnych etapach fazy „primordial”. Etapy te, zostały zaprezentowane na Rys. 9.



Rys. 9 Etapy fazy „primordial” algorytmu mGA

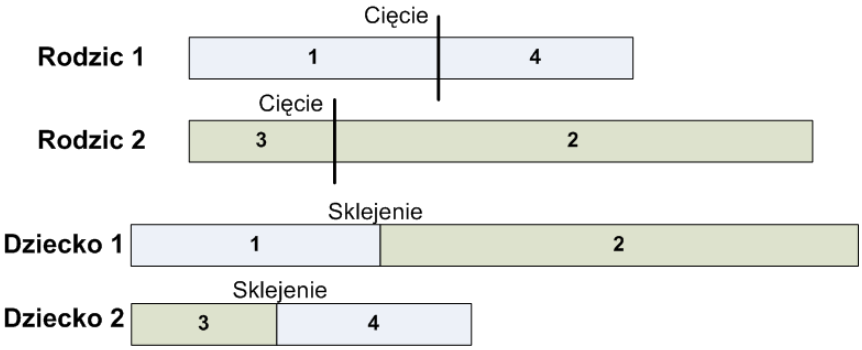
Na początku rozmiar populacji nie zmienia się, a porównywane są ze sobą jedynie sąsiadujące osobniki. Lepszy z nich zajmuje oba miejsca w kolejnej populacji – swoje i swojego przeciwnika (tzw. selekcja turniejowa). W następnym etapie dodane zostaje tasowanie osobników, a rozmiar populacji jest zmniejszany o połowę w co drugiej iteracji (w iteracjach, w których populacja jest zmniejszana o połowę zwycięzca otrzymuje jedno miejsce w kolejnej populacji). W kolejnym etapie rozmiar populacji jest zmniejszany o połowę w każdej iteracji. Wreszcie w ostatnim etapie rozmiar populacji znów jest stały. Liczba iteracji, która przypada na każdy etap fazy „primordial” jest określana przez użytkownika algorytmu. Należy zauważyć, że w fazie „primordial” nie są używane żadne operatory genetyczne.

W trakcie fazy „juxtapositional” populacja jest przetwarzana jak w klasycznym algorytmie genetycznym z tymi wyjątkami, że osobniki są inaczej zakodowane i zamiast operatora krzyżowania używa się operatorów cięcia i sklejania (ang. *cut and splice operators*). Operatory mutacji (typowe to: zmiana wartości genu, zmiana pozycji genu, oraz dodawanie/usuwanie losowo wybranego genu) zwykle nie są używane [34, 35, 36], ponieważ przeszukiwanie lokalne jest de facto dokonywane w trakcie fazy „primordial”.

Algorytm mGA w zastępstwie operatora krzyżowania używa operatorów cięcia i sklejania. Operator sklejania działa w następujący sposób: z zadanym prawdopodobieństwem p_s skleja dwa ciągi genów. Na przykład dwa ciągi genów [(5,1) (2,0) (1,0)] i [(2,1) (9,0)] po sklejeniu dadzą ciąg [(5,1) (2,0) (1,0) (2,1) (9,0)]. Operator cięcia służy do rozbijania ciągów genów na krótsze kawałki. Dla określonego przez użytkownika algorytmu parametru p_k i długości ciągu λ, prawdopodobieństwo przecięcia ciągu p_c wynosi $p_c = p_k(\lambda - 1)$ z

zastrzeżeniem, że p_c nie może przyjąć wartości większej niż 1. Na przykład dla $p_k = 0{,}05$ i $\lambda =$ 6, ciąg genów [(5,1) (2,0) (1,0) (2,0) (2,1) (9,0)] ulegnie przecięciu z prawdopodobieństwem $p_c = 0{,}25$. Zakładając, że los wskazał pozycję numer 2 jako punkt cięcia, w wyniku działania operatora otrzymamy następujące ciągi genów: [(5,1) (2,0)] i [(1,0) (2,0) (2,1) (9,0)].

Na Rys. 10 pokazane jest działanie operatorów cięcia i sklejania. Jeśli oba osobniki–rodzice zostały przecięte, to prawdopodobieństwo sklejenia jest sprawdzane dla par 1-2 i 3-4. Jeśli ciągi 1-2 zostaną sklejone, to otrzymany w wyniku tej operacji ciąg jest wstawiany do populacji, a prawdopodobieństwo sklejenia jest sprawdzane dla pary 3-4. Jeśli para 1-2 nie jest sklejana to wtedy ciąg 1 jest wstawiany do nowej populacji, a prawdopodobieństwo sklejania jest sprawdzane dla pary 2-3. Analogiczne operacje są wykonywane, jeśli jeden, lub żaden z dwojga rodziców nie zostanie przecięty.



Rys. 10 Działanie operatorów cięcia i sklejania (ang. *Cut&Splice operators*)

3.3.2 Algorytm fmGA

Najbardziej znaczącym wąskim gardłem w algorytmie mGA jest faza „primordial" i konieczność generacji ogromnej liczby osobników. W algorytmie fmGA (ang. *Fast Messy Genetic Algorithm*) [36] faza „primordial" została zastąpiona procedurą Kombinatorycznie Kompletnej Inicjalizacji (ang. *PCI - probabilistically complete initialization*), zwaną dalej PCI. W PCI, każdy osobnik jest inicjalizowany liczbą $l' = l - k$ losowo wybranych genów. W ramach procesu inicjalizacji nadreprezentacja któregokolwiek genu nie jest dopuszczalna. Następnie, przez ustaloną przez użytkownika pewną liczbę iteracji, wygenerowane osobniki podlegają selekcji turniejowej, której celem jest eliminacja słabszych propozycji i zwiększenie liczby tych osobników, które zawierają więcej „dobrych" bloków budujących. Następnie pewna liczba losowo wybranych genów jest usuwana z każdego osobnika (zwykle długość osobników jest skracana o połowę). Etap selekcji turniejowej i usuwania genów jest powtarzany aż do momentu, kiedy wszystkie osobniki mają długość *k*. Należy zauważyć, że selekcja turniejowa odbywa się wyłącznie dla osobników, które posiadają co najmniej θ wspólnych genów (ale niekoniecznie tych samych wartości dla tych genów), jeśli ten warunek nie jest spełniony, to drugi osobnik jest losowany powtórnie, tak długo, aż warunek zostanie spełniony, lub zostanie osiągnięta maksymalna liczba losowań. Wartość θ jest obliczana zgodnie ze wzorem:

$\theta = \left\lceil \frac{\lambda^2}{l} + c'(\alpha')\sigma \right\rceil$, gdzie λ jest liczbą genów, c(α) to kwantyl rzędu 1-α jednostronnego standardowego rozkładu gaussowskiego, σ jest definiowane zgodnie ze wzorem (41): (40)

$$\sigma = \frac{\lambda^2 (l-\lambda)^2}{l^2 (l-1)} \quad (41)$$

Algorytm fmGA pomija fazę "primordial", zastępując ją alternatywną procedurą PCI, co niewątpliwie likwiduje, lub przynajmniej znacząco ogranicza jedną, z istotnych wad algorytmu mGA, którą jest ogromna liczba ciągów, jakie należy wygenerować w trakcie fazy „primordial". Jednak nie zmienia to faktu, że fmGA w takim samym stopniu jest obciążone innymi wadami algorytmu mGA. Omówienie tych wad znajduje się w podrozdziale 3.3.3.

3.3.3 Wady algorytmów mGA i fmGA

Jedną z istotnych wad algorytmów mGA i fmGA jest zastosowany mechanizm selekcji turniejowej w fazie "primordial", oraz odpowiednio PCI. Zadaniem fazy "primordial", oraz procedury PCI jest zapewnienie następnym fazom algorytmu dostępu do dobrych bloków budujących. Niestety, obie te fazy działają dobrze dla problemów zbudowanych z podproblemów, które mają identyczny lub prawie identyczny wpływ na ogólną wartość funkcji przystosowania (jest to pokazane w [57], a wyniki te są również prezentowane w niniejszej pracy). Jest to wymóg, którego nie spełnia żaden problem praktyczny znany autorowi niniejszej pracy. Przyczyna wadliwego działania obu sposobów inicjalizacji stosowanych w algorytmach mGA i fmGA, zostanie pokazana na przykładzie:

Przypuśćmy, że problem jest zbudowany z konkatenacji dwóch podproblemów, które są funkcjami zwodniczymi o tym samym rzędzie i kształcie z tym że:

$$S_1(u_1) = 10 * f_x(u_1)$$
$$S_2(u_2) = f_x(u_2),$$

gdzie S_1, S_2 to, odpowiednio, pierwszy i drugi podproblem, u_1, u_2 są liczbą '1' występujących w części fenotypu dotyczącej rozwiązania podproblemu S_1, S_2, a f_x jest funkcją zwodniczą o rzędzie-*x*.
Należy oczekiwać, że dla takiego problemu jak zdefiniowany powyżej, w trakcie selekcji turniejowej, w fazie „primordial" lub procedurze PCI, większość, lub nawet wszystkie bloki budujące odnoszące się do problemu S_2 zostaną zastąpione przez bloki budujące odnoszące się do problemu S_1. Przyczyną takiego stanu rzeczy będzie fakt, że podproblem S_1 ma znacznie większy wpływ na wartość funkcji przystosownaia niż podproblem S_2.

Inną wadą algorytmów fmGA i mGA jest konieczność definiowania wartości *k* (długości bloków budujących, które są produktem fazy „primordial" i procedury PCI). Jeśli algorytm jest inicjalizowany dla *k=3* i problem zawiera pewną liczbę podproblemów w postaci funkcji zwodniczych rzędu 5 to algorytmy mGA i fmGA najprawdopodobniej nie odnajdą

optymalnego rozwiązania. Co więcej, nawet dla *k=5* algorytm najprawdopodobniej również nie znalazłby optymalnego rozwiązania. Na przykład: dla 25-bitowego problemu złożonego z 5 funkcji zwodniczych rzędu 5 i losowo zainicjalizowanego CT (patrz: rozdz. 3.3.1):

11011 11101 01111 11110 00100

najlepszym możliwym 5-genowym blokiem budującym będzie:

??1?? ???1? 1???? ????1 ??0??.

Taki blok budujący spowoduje, że rozwiązanie będzie optymlane dla 4 pierwszych podproblemów, ale dla ostatniego z nich zostanie zaproponowane rozwiązanie suboptymalne. W takim przypadku najlepsze możliwe rozwiązanie problemu, składające się z samych '1' najprawdopodobniej nie zostanie odnalezione. Kargupta w [47] proponuje, aby zainicjalizować algorytm fmGA *k=1* a następnie zwiększać wartość parametru *k* o 1 przy każdym kolejnym wywołaniu algorytmu jednocześnie używając CT, który jest efektem wywołania algorytmu w poprzedniej iteracji. Z jednej strony propozycja Kargupty wydaje się być jedynym logicznym wyjściem z sytuacji (choć, zdaniem autora, lepsze byłoby ciągłe wywoływanie algorytmu dla tej samej wartości k, poczynając od 1 i zwiększanie tej wartości dopiero wtedy, gdy dla danej wartości *k* nie jest już możliwe uzyskanie poprawy wartości funkcji przystosowania dla CT). Jednak z drugiej strony jest to właściwie wyrok śmierci na jakiekolwiek próby użycia algorytmu mGA, czy fmGA w zastosowaniu praktycznym. Po pierwsze dlatego, że nie wiemy jaka powinna być wartość graniczna dla parametru *k* (teoretycznie powinna być to długość problemu, ale to oznaczałoby konieczność przeprowadzenia tak ogromnej liczby operacji, że w praktyce nie miałoby to żadnego sensu). Po drugie, jeśli problem nie jest w pełni separowalny to efektywność mGA i fmGA zostanie najprawdopodobniej dodatkowo obniżona, ponieważ zależności pomiędzy poszczególnymi podproblemami (pytanie, czy w takiej sytuacji w ogóle można mówić jeszcze o podproblemach jest istotne, ale nie mieści się w zakresie zainteresowań niniejszej pracy i autor pozostawia je otwartym) mogą utworzyć podproblemy o większej długości. Na przykład: problem będący konkatenacją z dwóch podproblemów:

$$S_1(u_1) = \begin{cases} f_5(u_1) & dla \quad u_2 \neq 5 \\ f_5(5-u_1) & dla \quad u_2 = 5 \end{cases},$$

$$S_2(u_2) = f_5(u_2),$$

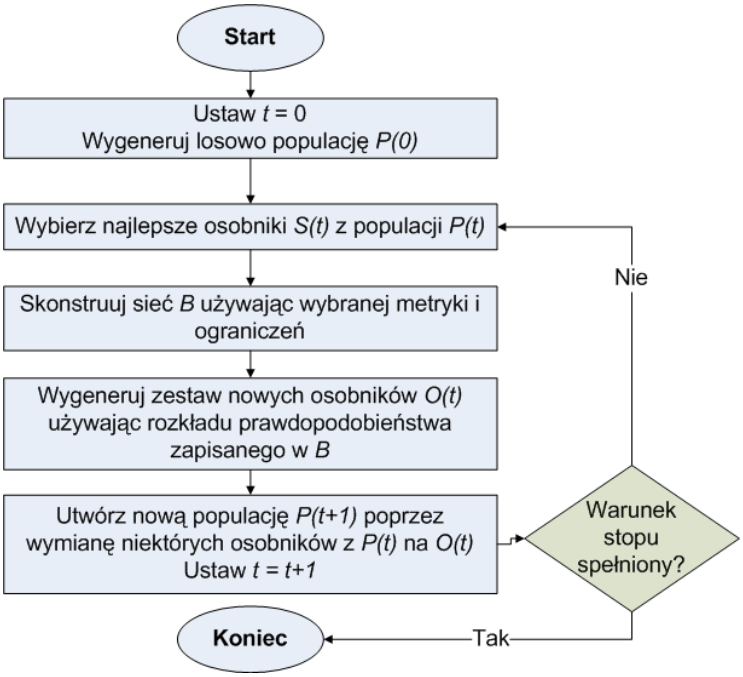
gdzie S_1, S_2 to odpowiednio wartość pierwszego i drugiego popdroblemu, u_1, u_2 to liczba '1' w części fenotypu odpowiadającej problemowi pierwszemu i drugiemu, a f_5 jest funkcją zwodniczą rzędu 5.

Powyższy problem przyjmuje wartość optymalną dla argumentu: 00000 11111. Jednak próbując rozwiązać powyższy problem korzystając z rady Kargupty (aby zacząć od *k=1* i zwiększać wartość tego parametru o 1 przy każdej iteracji aż do *k=5*), najprawdopodobniej rozwiązaniem zaproponowanym przez algorytm mGA/fmGA będzie ciąg 11111 11111. Należy zauważyć, że kłopot z uzyskaniem jak najlepszego rozwiązania będzie pogłębiał się wraz z rosnącą liczbą podproblemów, zależności między nimi, zwiększeniem liczby genów niezbędnych do zakodowania całego problemu, lub jeśli długości podproblemów będą się różnić.

Idee leżące u podstaw konstrukcji mGA i fmGA są, zdaniem autora niniejszej pracy, niezwykle interesujące. Jednak nikła wartość (by nie powiedzieć niemal żadna) dla zastosowań praktycznych skłania do poszukiwań innych propozycji, które eliminowałyby lub ograniczały wady mGA i fmGA. Taka próbą jest przedstawiony w kolejnym rozdziale algorytm BOA [72, 73].

3.4 Algorytm BOA - opis

Algorytm BOA (Bayesian Optimization Algorithm) jest metodą hybrydową [72, 73], bazującą na użyciu sieci bayesowskich do reprezentowania zależności pomiędzy genami. Na Rys. 11 zaprezentowany jest schemat działania algorytmu BOA.



Rys. 11. Schemat działania algorytmu BOA [73]

Do wybrania grupy lepszych osobników *S(t)* z populacji *P(t)* może zostać użyty dowolny algorytm selekcji. Sieć bayesowska *B* jest konstruowana na bazie wybranych osobników *S(t)*. Dowolna metryka może zostać użyta jako miara jakości sieci *B*. Algorytm szukający propozycji sieci *B*, również może zostać wybrany dowolnie.

Sieci bayesowskie [71] opisują zależności pomiędzy zmiennymi zawartymi w zbiorze danych, który podlega modelowaniu. Mogą zostać użyte do opisu danych i do wygenerowania nowych zmiennych, które będą posiadać podobne własności jak te zmienne, na podstawie których została wygenerowana sieć bayesowska. Każda zmienna odnosi się do jednej pozycji w ciągach danych, które tworzą rozwiązanie. Krawędź pomiędzy węzłami w sieci bayesowskiej oznacza pojedynczą relację pomiędzy dwiema zmiennymi. BOA używa wyłącznie skierowanych, acyklicznych sieci bayesowskich, które kodują łączny rozkład prawdopodobieństwa:

$$p(X) = \prod_{i=0}^{n-1} p(X_i \mid \Pi_{X_i}), \tag{42}$$

gdzie
$X = (X_0, X_1, ..., X_{n-1})$ jest wektorem zmiennych, Π_{X_i} jest zbiorem rodziców zmiennych X_i w sieci, a $p(X_i \mid \Pi_{X_i})$ jest warunkowym prawdopodobieństwem dla X_i przy zadanym zbiorze Π_{X_i}

Najbardziej naturalnym sposobem na zbudowanie sieci bayesowskiej to sprawdzenie wszystkich dostępnych możliwości i wybór najlepszej z nich. Należy jednak zauważyć, że liczba wszystkich możliwych do zbudowania sieci zwykle będzie ogromna. Dlatego ich liczba jest ograniczana poprzez wprowadzenie parametru *k*, określającego maksymalny stopień wierzchołka w sieci. W niniejszej pracy, dla porównania jakości poszczególnych sieci stosowana jest metryka K2 [73].

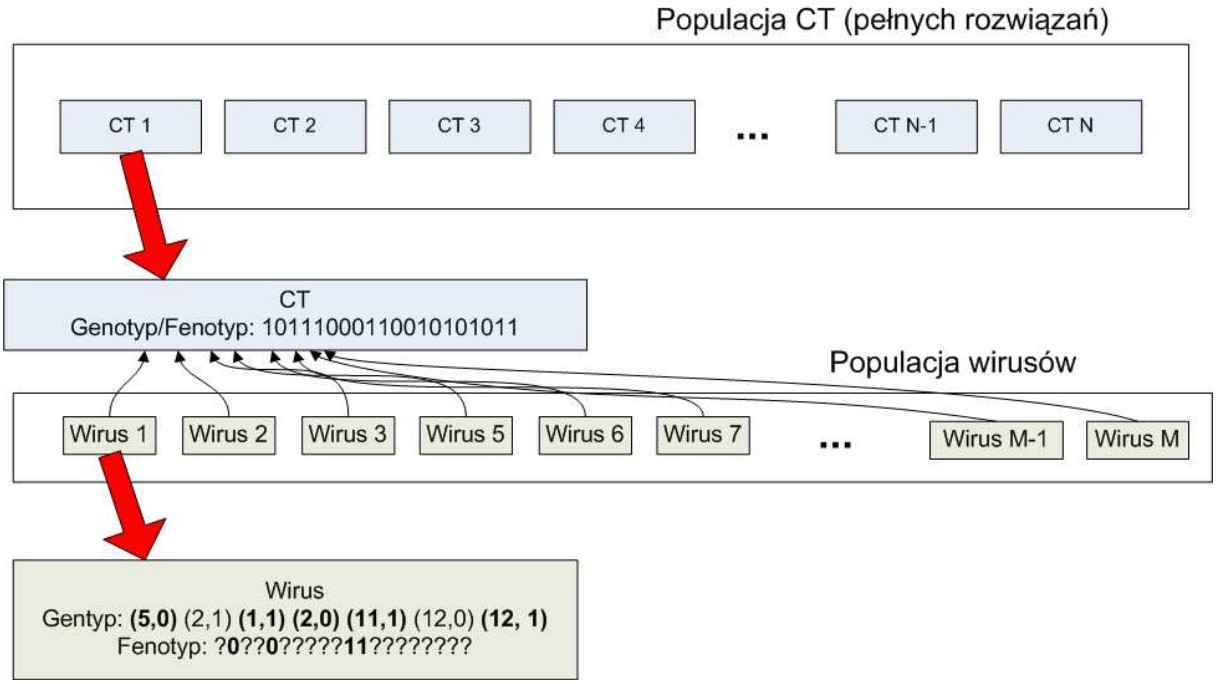
Algorytm BOA jest bardzo skutecznym algorytmem w rozwiązywaniu problemów o kodowaniu binarnym, zwłaszcza konkatenacji funkcji zwodniczych [72, 73]. Posiada jednak również istotne wady. Jedną z nich jest definiowany przez użytkownika maksymalny stopień wierzchołka sieci (*k*). Im większa wartość tego parametru tym dokładniej sieć może oddawać zależności pomiędzy genami i ich wartościami, z drugiej jednak strony im większa wartość tego parametru tym więcej czasu zajmuje utworzenie sieci.

3.5 Algorytm MuPPetS – opis

Algorytm MuPPetS (**Mu**lti **P**opulation **P**a**t**tern **S**earching Algorithm) [57] został zaprojektowany do rozwiązywania problemów charakteryzujących się cechami przedstawionymi w rozdziale 2.4.6. niniejszej pracy. Oznacza to, że problem projektowania przepływu w szkieletowych sieciach komputerowych jest problemem, dla którego algorytm MuPPetS powinien być szczególnie skuteczny.

3.5.1 Struktura danych algorytmu MuPPetS

Struktura danych algorytmu MuPPetS została zaprezentowana na Rys. 12.



Rys. 12 Struktura danych algorytmu MuPPetS

Wyższy poziom struktury danych algorytmu MuPPetS składa się z pewnej liczby szablonów (patrz: rozdz. 3.3.1), zwanych w niniejszej pracy CT. CT stanowią kompletne propozycje rozwiązania problemu z typowym dla algorytmu genetycznego kodowaniem. Każdy CT posiada własną, przypisaną mu, populację wirusów. Sposób kodowania wirusów jest taki sam jak w przypadku osobników w nieporządnym algorytmie genetycznym (ang. *messy GA*). Osobniki w nieporządnym algorytmie genetycznym zwane są nieporządnymi osobnikami (ang. *messy individual*). Na potrzeby niniejszej pracy będzie jednak używane określenie „wirus". Według autora niniejszej pracy taka zmiana nazewnictwa jest uprawniona i pożądana z powodu następujących przyczyn [57]:

- **Analogia z naturą**. Wirusy w naturze nie mogą istnieć i rozmnażać się bez żywiciela. W algorytmie MuPPetS wirus nie znaczy wiele bez swojego kontekstu – CT, do którego jest przypisany.
- **Różnice pomiędzy rolą wirusów w algorytmie MuPPetS, a rolą nieporządnych osobników w algorytmach mGA i fmGA**. Na podstawie [34, 35, 36] można stwierdzić, że nieporządne osobniki (*messy individual*) w algorytmach mGA i fmGA, choć mogą zawierać luki w reprezentacji poszczególnych genów (ang. *underspecification*), mają na celu utworzenie kompletnych rozwiązań. Ponadto w wymienionych pracach nieporządne osobniki nie są poddawane działaniu operatora mutacji, a ich inicjalizacja następuje po kosztownej obliczeniowo fazie primordial, bądź PCI (różnych dla obu algorytmów, ale wspólnych co do idei i efektu działania). Dla odróżnienia, w algorytmie MuPPetS wirusy mają z założenia kodować jedynie fragment rozwiązania, stosowane operatory mutacji są istotną częścią algorytmu, a proces inicjalizacji wirusów jest w dużym stopniu losowy i mało kosztowny obliczeniowo.

3.5.2 Wzorce genów – główna idea algorytmu MuPPetS

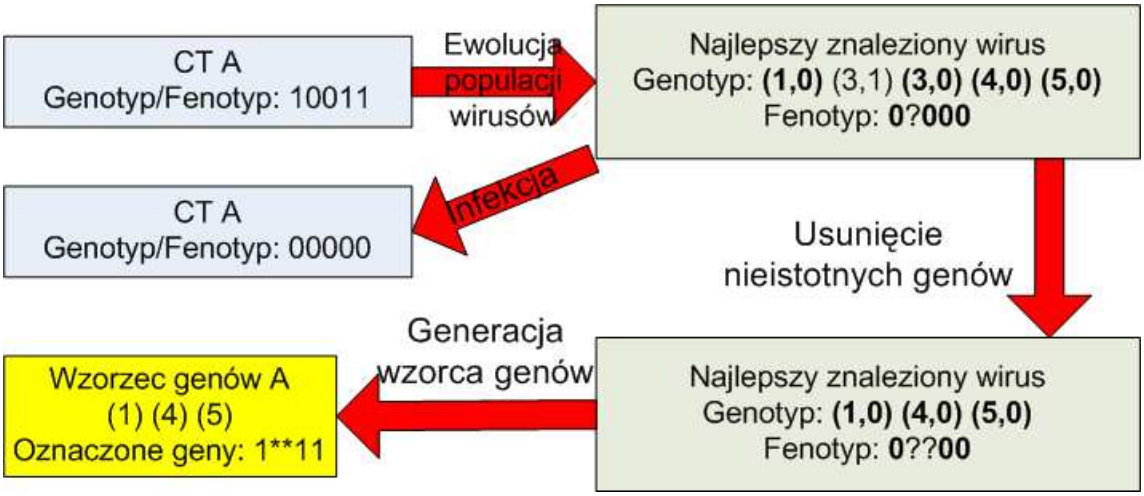
Wzorce genów (ang. *Gene patterns*) [57] i ich użycie to główna idea, wokół której koncentruje się budowa algorytmu MuPPetS. Celem użycia wzorców genów jest odwzorowanie możliwych zależności pomiędzy genami. Informacja o tych zależnościach pozwala na podniesienie efektywności operatorów genetycznych takich jak krzyżowanie. Zrozumienie idei wzorców genów oraz mechanizmów ich przetwarzania prowadzi do zrozumienia całej budowy algorytmu.

Wzorzec genów może być zdefiniowany jako wirus, którego genotyp zawiera jedynie pozycje genów (bez wartości odpowiadających poszczególnym allelom). Przyjęta w niniejszej pracy konwencja kodowania wzorca genów jest następująca: [(*p*) (*p*) (*p*) (*p*)], gdzie *p* jest pozycją genu w fenotypie. Na przykład następujący wzorzec genów: [(1) (1) (2)] zaznacza dwie pozycje – pierwszą i drugą. Należy zauważyć, że ten sam wzorzec genów może zaznaczać pewne pozycje dowolną liczbę razy (na przykład pozycja '1' w podanym przykładzie jest zaznaczona dwa razy). W algorytmie MuPPetS zdecydowano się na przechowywanie wzorców genów w sposób podobny do kodowania wirusów, zamiast bezpośredniego kodowania pozycja oznaczona/pozycja nieoznaczona (tak jak w tradycyjnym algorytmie genetycznym koduje się rozwiązania), ponieważ w niektórych fazach algorytmu MuPPetS wzorce genów są używane do tworzenia wirusów. W takiej sytuacji wydaje się pożądanym, aby pozycje genów we wzorcu były przechowywane w sposób, w jaki ustaliła to ewolucja wirusa, na podstawie którego wzorzec genów został utworzony.

Ważnym do podkreślenia pozostaje fakt, że wzorce genów to nie bloki budujące (ang. *building blocks*) [34, 35, 36]. Wzorzec genów to zakodowana informacja o możliwych zależnościach pomiędzy genami na poszczególnych pozycjach. Ta informacja jest użyteczna w wymianie informacji pomiędzy poszczególnymi CT i umożliwia zwiększenie efektywności operatora krzyżowania.

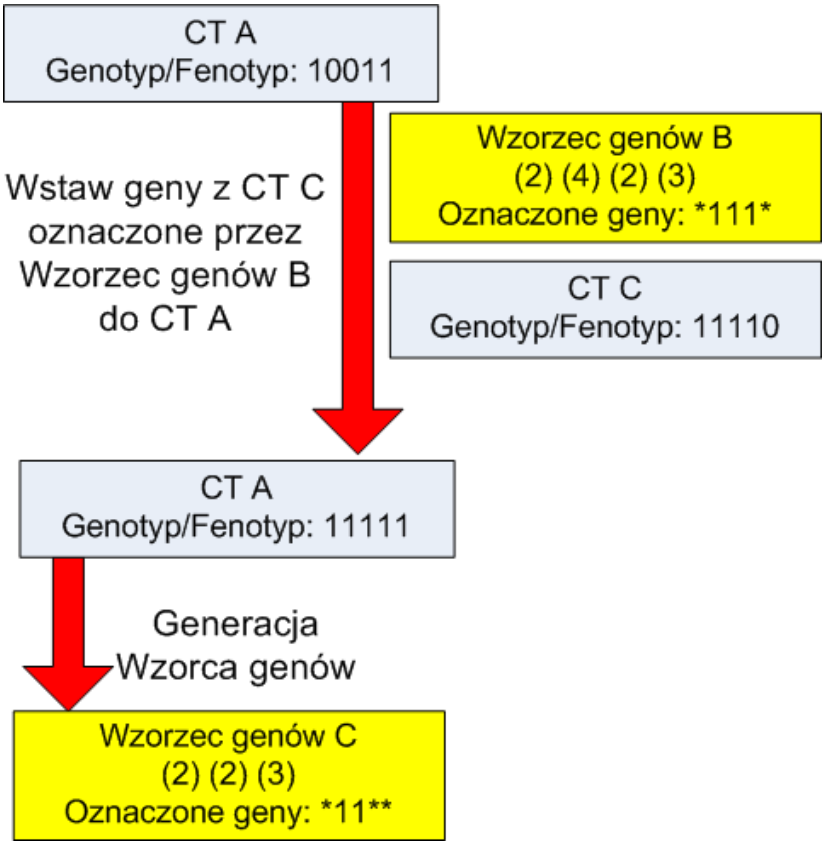
3.5.2.1 Metody pozyskiwania wzorców genów

Załóżmy, że dla pewnego CT, po przeprowadzeniu pewnej liczby operacji na populacji wirusów, znaleziony został wirus o większej niż odpowiadający mu CT wartości funkcji przystosowania. Taka sytuacja została przedstawiona na Rys. 13.



Rys. 13. Tworzenie wzorca genów na podstawie wirusa

Pierwszą operacją prowadzącą do wygenerowania wzorca genów jest usunięcie „nieistotnych genów” z genotypu wirusa. Za nieistotne uznaje się wszystkie geny w genotypie wirusa, których allele odpowiadają pozycjom, które są identyczne w fenotypie wirusa i w fenotypie CT (w przypadku CT fenotyp jest równoważny genotypowi). Operacja usunięcia nieistotnych genów jest przeprowadzana w związku z domniemaniem, że geny w fenotypie wirusa, które nie zmieniają niczego w fenotypie CT są mniej istotne dla pozostałych genów fenotypu wirusa. Domniemanie to opiera się na fakcie, że geny, które są wspólne dla fenotypów wirusa i CT i tak zostałyby dostarczone wirusowi przez CT, a więc ich obecność nie zmienia niczego w wartości funkcji przystosowania wirusa. Z drugiej strony, uprawnionym wydaje się domniemanie, że geny, które zmieniają genotyp/fenotyp CT, prowadząc tym samym do poprawienia wartości funkcji przystosowania tego CT (operacja infekcji Rys. 13.), mogą być od siebie zależne bardziej niż od pozostałych genów.

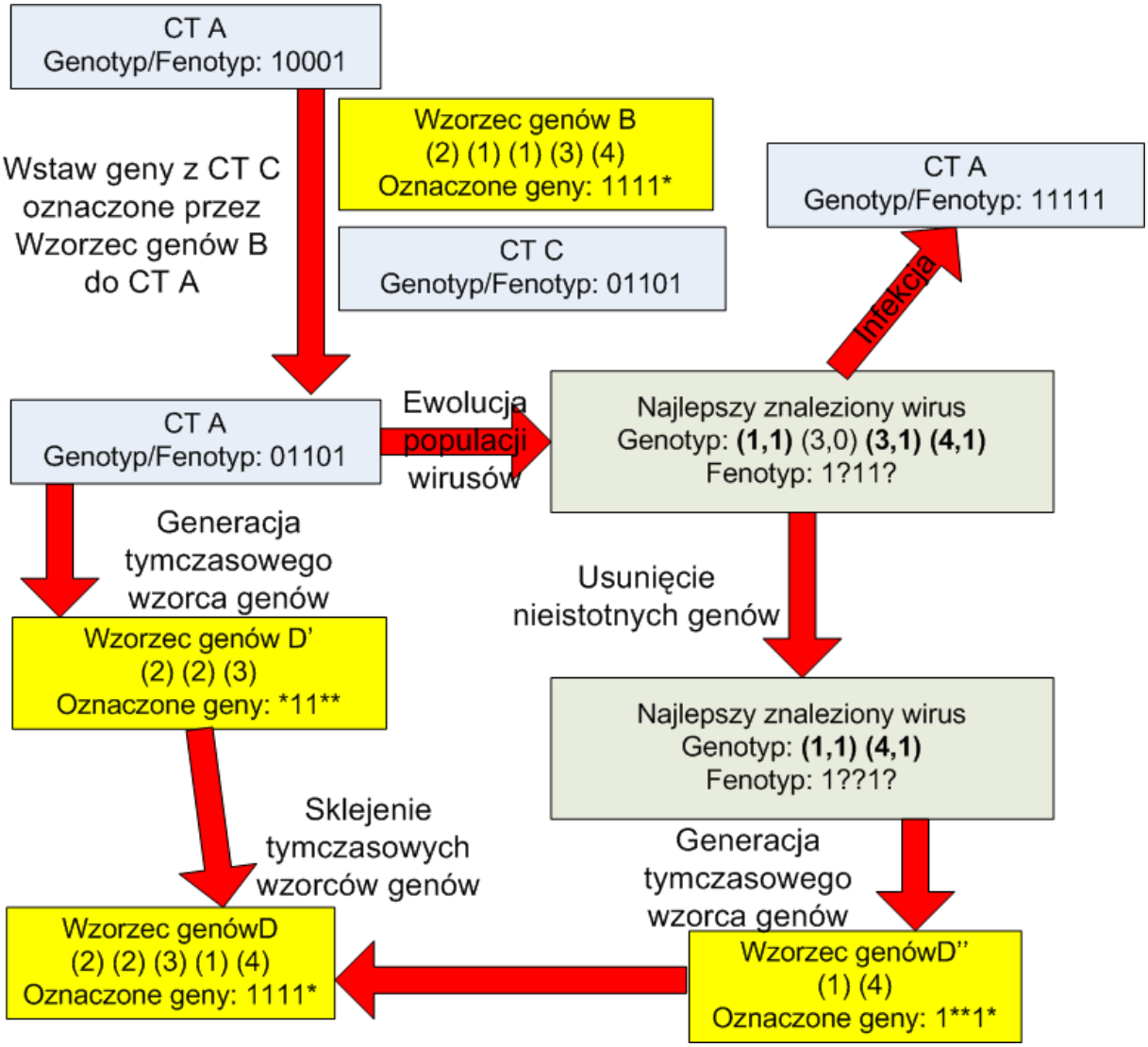


Rys. 14. Tworzenie nowego wzorca genów na podstawie istniejącego wzorca genów i CT

Na Rys. 14. przedstawiona jest sytuacja, w której nowy wzorzec genów jest pozyskiwany na podstawie istniejącego wzorca genów oraz CT. Operacja jest podobna do pozyskiwania wzorca genów wyłącznie na podstawie wirusa, z tą różnicą, że geny wstawiane do CT A pochodzą nie z wirusa, ale są to geny z CT C oznaczone przez wzorzec genów B. Generacja nowego wzorca genów następuje (podobnie jak w poprzednim przypadku) wyłącznie w sytuacji, kiedy operacja wstawiania genów z CT C do CT A prowadzi do poprawy wartości funkcji przystosowania CT A. Nowy wzorzec genów C jest kopią wzorca genów B z usuniętymi tymi allelami, które zostały oznaczone przez wzorzec B, ale posiadały identyczne wartości w CT A i CT C.

Jeśli operacja wstawienia genów do CT A oznaczonych przez wzorzec genów B w CT C (Rys. 14.) spowoduje, że wartość funkcji przystosowania CT A nie zmieni się lub spadnie,

wtedy przetwarzana jest populacja wirusów i pozyskanie nowego wzorca genów opiera się również o genotyp najlepszego znalezionego wirusa (Rys. 13.). Złożenie obu powyższych operacji przedstawione jest na Rys. 15.



Rys. 15. Tworzenie nowego wzorca genów na podstawie istniejącego wzorca genów, CT i wirusa

Wzorce genów otrzymane na podstawie operacji kopiowania genów oznaczonych przez wzorzec genów B z CT C do CT A i na podstawie najlepszego znalezionego wirusa (na Rys. 15. oznaczone jako „tymczasowe") są sklejane dając w efekcie ostateczną propozycję nowego wzorca genów D. Idea stojąca za opisaną powyżej operacją może zostać wyrażona w następujący sposób:

Jeśli geny wstawione do CT A z CT C spowodowały spadek wartości funkcji przystosowania to znaczy, że częściowo uszkodziły co najmniej jeden blok budujący (ang. *building block* – patrz punkt 3.1), lub wymieniły co najmniej jeden blok budujący z lepszego na gorszy. Jeśli po tej operacji geny wstawione z najlepszego znalezionego wirusa spowodowały poprawę wartości funkcji przystosowania, to można domniemywać, że bloki budujące zostały całkowicie lub częściowo naprawione. Jeśli po przeprowadzeniu obu operacji wartość funkcji przystosowania jest wyższa niż przed ich przeprowadzeniem, to można domniemywać, że bloki budujące uszkodzone w pierwszej

operacji zostały wymienione lub zmienione na lepsze. Jeśli bloki budujące zostały najpierw uszkodzone poprzez wymianę pewnych genów, a następnie naprawione poprzez wymianę innych genów (dotyczących innych pozycji w fenotypie), to można domniemywać, że wszystkie wspomniane geny (i te których wymiana spowodowała zniszczenia, i te których wymiana spowodowała naprawę) są ze sobą bardziej powiązane niż inne geny w genotypie, a zatem jest rozsądnym umieszczenie ich w jednym wzorcu genów.

W kontekście powyższego rozumowania należy zauważyć, że:

- Jako całość przedstawiona operacja jest do pewnego stopnia zbliżona do koncepcji globalnej mutacji (czasowe pogorszenie wartości funkcji przystosowania, które ma na celu opuszczenie lokalnego optimum i znalezienie globalnego optimum, lub innego lokalnego optimum, najlepiej o wyższej wartości funkcji przystosowania)
- Problemy obliczeniowe spotykane w świecie rzeczywistym zwykle nie są w pełni separowalne, a więc nie jest możliwe znalezienie optimum poprzez naruszenie, naprawę, bądź wymianę jednego, konkretnego bloku budującego.

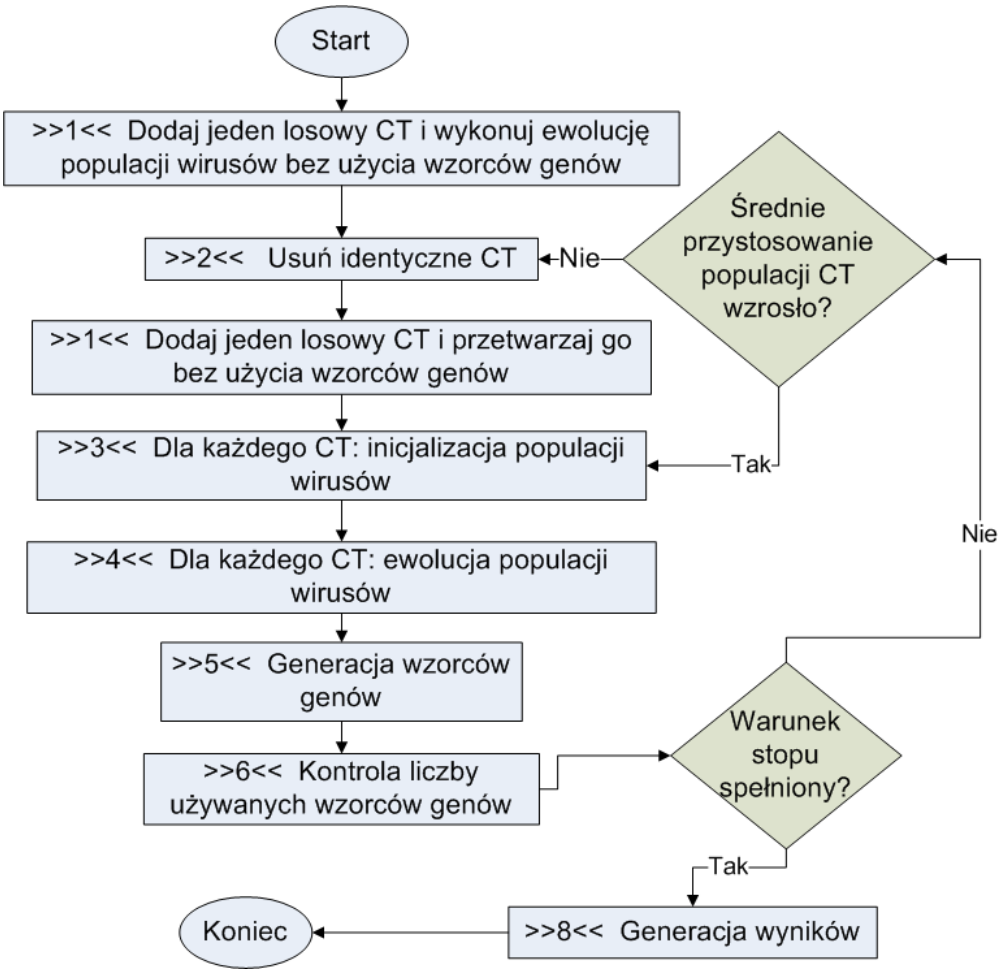
3.5.2.2 Wzorce genów – dlaczego są pożyteczne

Wzorce genów są dodatkową informacją o zależnościach pomiędzy genami, zebraną w trakcie przebiegu algorytmu. Istotnym jest, że wzorce genów zwiększają efektywność algorytmu. Nie jest jednak rozsądnym oczekiwać, że w którejkolwiek fazie algorytmu pula wzorców genów będzie zawierać wszystkie istniejące grupy zależnych od siebie genów. Głównym powodem takiej sytuacji jest fakt, że **nie jest uprawnionym** zamykanie rozumowania dotyczącego wzorców genów wyłącznie do konkatenacji funkcji zwodniczych, które posłużyły do testów dla algorytmu MuPPetS. Konkatenacje funkcji zwodniczych są doskonałym narzędziem testowym, ale niczym więcej. Problemy obliczeniowe spotykane w rzeczywistym świecie prawie nigdy nie są w pełni separowalne, tak więc zwykle „idealnym" wzorcem genów będzie wzorzec oznaczający wszystkie, lub prawie wszystkie geny. Także sieć powiązań pomiędzy poszczególnymi genami będzie zwykle tak skomplikowana, że w praktyce nigdy nie będziemy zdolni do pokrycia wszystkich mocnych bloków budujących (ang. *tight building blocks*). Należy również zauważyć, że sformułowania typu „geny, które są od siebie zależne bardziej niż od innych genów", czy „blok budujący" często, używane w niniejszej pracy i w literaturze [11, 15, 34, 35, 36, 39, 40, 57] są sformułowaniami, których precyzyjnie nie potrafi zdefiniować ani autor niniejszej pracy, ani inni badacze - autor niniejszej pracy nie spotkał się z miarą dla siły zależności/powiązań między genami.

Innym powodem, dla którego pula wzorców genów nie musi zawierać wzorców genów oznaczających wszystkie możliwe bloki budujące/grupy ściśle powiązanych genów (czymkolwiek te bloki budujące by nie były) są przyczyny praktyczne. Ogromna liczba wzorców genów byłaby raczej problemem niż korzyścią dla algorytmu. Ponadto można przyjąć, że dobry wzorzec genów to taki, który daje szansę na poprawę już istniejących CT. A więc wzorzec genów „dobry" w danym momencie (iteracji) algorytmu, wcale nie musi taki być kilka iteracji później. Należy również pamiętać, że jeśli wzorzec genów, który byłby pożyteczny w danym momencie nie istnieje jeszcze w puli wzorców genów, to zawsze istnieje szansa, że taki wzorzec, lub jego część zostanie znaleziona.

3.5.3 Fazy algorytmu MuPPetS

Każda iteracja algorytmu MuPPetS składa się z sześciu faz, które są powtarzane do czasu, aż algorytm spełni warunek stopu. Wszystkie fazy algorytmu zostały przedstawione na Rys. 16. Jako warunek stopu może zostać użyty limit czasowy, liczba iteracji lub inny. Poniżej znajduje się omówienie wszystkich faz algorytmu.



Rys. 16. Fazy algorytmu MuPPetS

Faza 1. CT jest tworzony w typowy dla algorytmów genetycznych, losowy sposób. Po wygenerowaniu CT generowana jest, przypisana do niego populacja wirusów. Każdy z wirusów jest tworzony w losowy sposób – każda pozycja oraz wartość genu wybierane są losowo z równym prawdopodobieństwem dla wszystkich możliwości. Długość genotypu wirusa jest określana losowo, z równym prawdopodobieństwem dla każdej możliwości i mieści się w zakresie od 1 do liczby genów niezbędnej do zakodowania pełnego rozwiązania. Po utworzeniu populacji wirusów, dla CT i odpowiadającej mu populacji wirusów przeprowadzana jest ewolucja populacji wirusów, która jest równoważna fazie juxtapositional algorytmu messy GA. Jeśli najlepszy znaleziony wirus posiada wyższą wartość funkcji przystosowania niż CT, to fenotyp wirusa zastępuje odpowiednie części genotypu/fenotypu CT (następuje „infekcja”). Te dwie operacje (inicjalizacja populacji wirusów i ewolucja

populacji wirusów) są wykonywane tak długo, dopóki wartość funkcji przystosowania dla CT ulega poprawie.

Faza 2. W tej fazie wszystkie CT w populacji CT są porównywane ze sobą nawzajem. Jeśli jakakolwiek para CT jest ze sobą identyczna, to jeden CT należący do takiej pary jest usuwany. Tak więc po tej fazie algorytmu MuPPetS wszystkie CT są parami różne.

Faza 3. W tej fazie następuje inicjalizacja populacji wirusów dla wszystkich CT. Najpierw każdy wirus jest inicjalizowany tak, jak miało to miejsce w fazie 1. Na przykład, jeśli losowo określona długość genotypu wirusa wynosi 2, to jego losowo wygenerowany genotyp może mieć następującą postać: [(3,1) (1,1)]. Następnie dla każdego wirusa wybierane są wzorzec genów i CT.

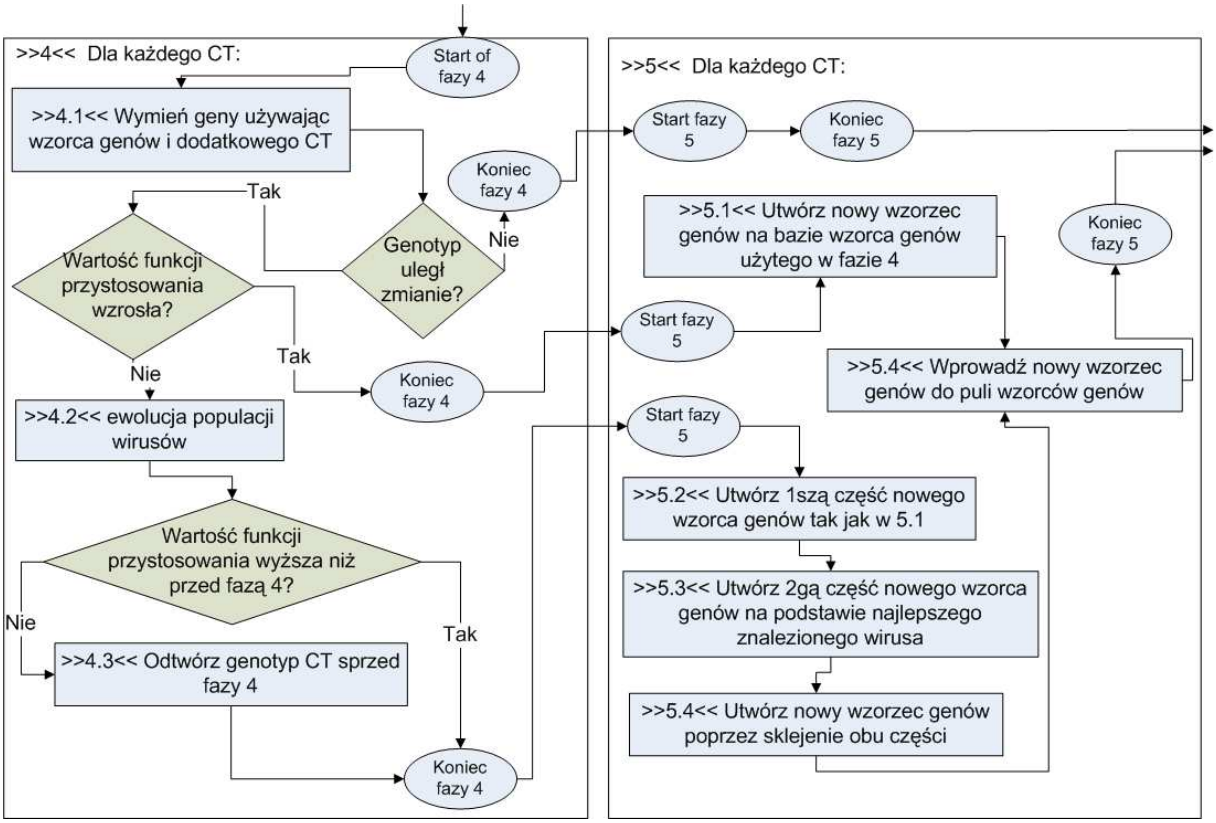
Wybór wzorca genów jest losowy z równym prawdopodobieństwem wyboru dowolnego wzorca genów, który znajduje się w puli wzorców genów. Po wybraniu wzorca genów wybierany jest losowo CT. Do wyboru CT używana jest metoda koła ruletki, prawdopodobieństwo wyboru poszczególnych CT jest proporcjonalne do wartości funkcji *f'*, której wartość jest wyliczana zgodnie ze wzorem:

$$f_i'(ct, pattern) = f_i \cdot d(ct, ct_i, pattern),$$

gdzie *ct* to CT, do którego przypisany jest dany wirus, f_i jest wartością funkcji przystosowania dla *i-tego* CT w populacji CT, a $d(ct, ct_i, pattern)$ oznacza liczbę różnych genów, oznaczonych przez wybrany dla inicjalizacji wirusa wzorzec genów, pomiędzy CT, do którego przypisany jest wirus i *i-tym* CT w populacji CT.

Wszystkie geny z CT wybranego do inicjalizacji wirusa, oznaczone przez wzorzec genów wybrany do inicjalizacji wirusa są dodawane do genotypu wirusa na bardziej znaczących pozycjach. Na przykład dla wzorca genów [(1) (1) (2)] i CT 01101, do genotypu wirusa zostanie dodany ciąg genów: [(1,0) (1,0) (2,1)]. W związku z tym, biorąc pod uwagę losowo wygenerowany fragment genotypu wirusa z przykładu podanego powyżej [(3,1) (1,1)], genotyp wirusa po inicjalizacji będzie miał następującą postać: [(3,1) (1,1), (1,0) (1,0) (2,1)].

Faza 4. Dla każdego CT (zwanego w niniejszym opisie *aktualnym*) wybierany jest losowo wzorzec genów i dodatkowy CT (zwany w niniejszym opisie *dodatkowym*). Selekcja wzorca genów i *dodatkowego* CT odbywa się w taki sam sposób jak w przypadku procedury inicjalizacji wirusa, opisanej w fazie 3. Geny z *dodatkowego* CT, oznaczone przez wzorzec genów wymieniają odpowiednie geny w *aktualnym* CT (ramka 4.1 na Rys. 17). Jeśli ta operacja spowoduje wzrost wartości funkcji przystosowania *aktualnego* CT, lub genotyp *aktualnego* CT nie ulegnie zmianie, to faza 4 kończy się. W przeciwnym razie przeprowadzana jest ewolucja populacji wirusów (faza *juxtapositional* algorytmu messy GA, ramka 4.2 Rys. 17), a geny z najlepszego znalezionego wirusa wymieniają odpowiednie geny w genotypie *aktualnego* CT (infekcja), jeśli prowadzi to do poprawy wartości funkcji przystosowania *aktualnego* CT. Jeżeli po zainfekowaniu *aktualnego* CT przez najlepszy znaleziony wirus, wartość funkcji przystosowania *aktualnego* CT jest niższa niż przed fazą 4, to *aktualnemu* CT przywracany jest genotyp sprzed fazy 4 (ramka 4.3 Rys. 17). Należy jednak zauważyć, że przywrócenie genotypu sprzed fazy 4 nie blokuje generacji wzorca genów bazującego na zmianach genotypu *aktualnego* CT, które zaszły w fazie 4.



Rys. 17. Fazy 4 i 5 algorytmu MuPPetS

Faza 5. Generacja wzorca genów jest przeprowadzana dla każdego CT, dla którego wartość funkcji przystosowania wzrosła po wykonaniu fazy 4, lub w przypadku którego doszło do infekcji w efekcie ewolucji populacji wirusów. Generacja wzorca genów bazuje na dwóch elementach – wzorcu genów, który został użyty na początku fazy 4 do wymiany genów pomiędzy *aktualnym* CT, a *dodatkowym* CT, oraz genotypie najlepszego znalezionego wirusa.

Na początku fazy 4 zostają losowo wybrane wzorzec genów i *dodatkowy* CT. Geny z *dodatkowego* CT oznaczone przez wzorzec genów wymieniają odpowiednie geny w *aktualnym* CT. Jeśli ta operacja nie doprowadzi do żadnych zmian w genotypie *aktualnego* CT, to faza 4 kończy się, a w fazie 5. nie jest generowany żaden nowy wzorzec genów.

Jeśli co najmniej jeden gen ulegnie zmianie, to wszystkie pozycje wskazujące geny, które nie uległy zmianie, są usuwane ze wzorca genów, użytego do wymiany genów pomiędzy *aktualnym* CT, a *dodatkowym* CT. Wynik tej operacji może być wynikiem działania (propozycją nowego wzorca) dla całej fazy 5, lub stanowić część wyniku działania fazy 5 (kratki 5.1 i 5.2 na Rys. 17). Na przykład: jeśli na początku fazy 4 genotyp *aktualnego* CT to 01011, wybrany wzorzec genów, oznaczający geny 3, 4, i 5 to [(4) (5) (3) (4) (5)], a *dodatkowy* CT to 11101, wtedy efektem operacji wstawienia genów z *dodatkowego* CT do *aktualnego* CT będzie: 01101. Należy zauważyć, że tylko 3ci i 4ty gen uległy zmianie, a więc wzorzec genów otrzymany na podstawie tej operacji będzie miał postać: [(4) (3) (4)].

Jeśli operacja wstawienia genów z *dodatkowego* CT do *aktualnego* CT spowoduje wzrost wartości funkcji przystosowania *aktualnego* CT wtedy faza 4 zakończy się bez ewolucji populacji wirusów (ramka 4.2 na Rys. 17), a wygenerowana propozycja nowego wzorca genów jest wstawiana do puli wzorców genów. Jeśli jednak ewolucja populacji wirusów odbędzie się, to na podstawie najlepszego znalezionego wirusa generowany jest wzorzec genów (ramka 5.3 na Rys. 17). Najpierw nieistotne geny są usuwane z genotypu wirusa – jeśli fenotyp najlepszego znalezionego wirusa zawiera geny o tych samych wartościach co *aktualny* CT, to wszystkie geny dotyczące takiej pozycji są usuwane z genotypu wirusa. Na

przykład: Przyjmijmy genotyp *aktualnego* CT to 01101, a genotyp najlepszego znalezionego wirusa [(1,0) (3,0) (1,1) (2,0) (3,1)], co daje fenotyp 101??. W fenotypie wirusa trzeci gen posiada tę samą wartość co trzeci gen w CT, a więc wszystkie geny odnoszące się do pozycji 3, są usuwane z genotypu wirusa. Po tej operacji genotyp wirusa będzie miał postać: [(1,0) (1,1) (2,0)]. Na końcu, z genotypu wirusa usuwane są wartości poszczególnych genów tworząc propozycję wzorca genów: [(1) (1) (2)] oznaczającego pozycje 1 i 2. Wzorce genów otrzymane na podstawie wirusa i wstawienia genów z *dodatkowego* CT do *aktualnego* CT ([(4) (3) (4)]) są ze sobą sklejane, dając ostateczną propozycję wzorca genów [(4) (3) (4) (1) (1) (2)], która jest wynikiem działania fazy 5.

Wzorzec genów, uzyskany w sposób opisany powyżej, jest następnie dodawany do puli wzorców genów pod warunkiem, że *d* losowo wybranych wzorców genów z puli oznacza inne zestawy genów niż nowy wzorzec (ramka 5.4 na Rys. 17). Na przykład: jeśli chcemy dodać do puli wzorców genów, wzorzec [(1) (1) (2)], przy *d*=3 i losowo wybranych z puli wzorcach genów [(1) (3) (2)], [(1) (2) (2)], [(3)], to nowy wzorzec genów nie zostanie dodany do puli wzorców genów, ponieważ drugi losowo wybrany z puli wzorzec genów oznacza dokładnie te same geny - 1 i 2. Porównanie nowego wzorca genów z *d* losowo wybranymi wzorcami genów z puli wzorców genów ma na celu utrzymanie różnorodności w puli wzorców genów. Jeśli nowy wzorzec genów nie został dodany do puli wzorców, to jest on zapamiętywany. W następnej iteracji algorytmu zapamiętany wzorzec jest sklejany z nową propozycją wzorca genów wygenerowaną przez ten sam CT. Na przykład: jeśli wzorzec [(1) (1) (2)] nie został dodany do puli wzorców genów, a w następnej iteracji algorytmu, dla tego samego CT, wygenerowano wzorzec [(5) (2)], to oba wzorce są sklejane tworząc wzorzec [(1) (1) (2) (5) (2)] i dla takiej postaci wzorca przeprowadzana jest kolejna próba dodania wzorca do puli wzorców genów.

Faza 6. Rozmiar puli wzorców genów jest określany przez użytkownika. Jeśli liczba wzorców genów w puli wzorców genów przekroczy dozwolony rozmiar, to niektóre ze wzorców będą musiały zostać usunięte z puli. Wzorce usuwane są losowo z równym prawdopodobieństwem usunięcia dla każdego wzorca, aż do momentu, kiedy liczba wzorców w puli nie przekracza ustalonego limitu.
Fakt, że nadliczbowe wzorce genów usuwane są z puli całkowicie losowo, bez udziału jakichkolwiek specjalizowanych operatorów odróżniających lepsze wzorce genów od gorszych, może zostać uznany za zaskakujący. W trakcie badań algorytmu MuPPetS podejmowano próby znalezienia metod pozwalających na określenie jakości wzorca. Wszystkie te próby zakończyły się jednak niepowodzeniem. Zdaniem autora niniejszej pracy przyczyną tego stanu rzeczy był następujący fakt: zakładając, że o własnościach funkcji przystosowania nie wiemy nic lub niewiele, nie posiadamy żadnych wskazówek aby decydować, które wzorce genów są w danym momencie lepsze od innych. Z tego powodu można jedynie liczyć na to, że lepsze wzorce genów będą liczniej reprezentowane w puli wzorców genów.

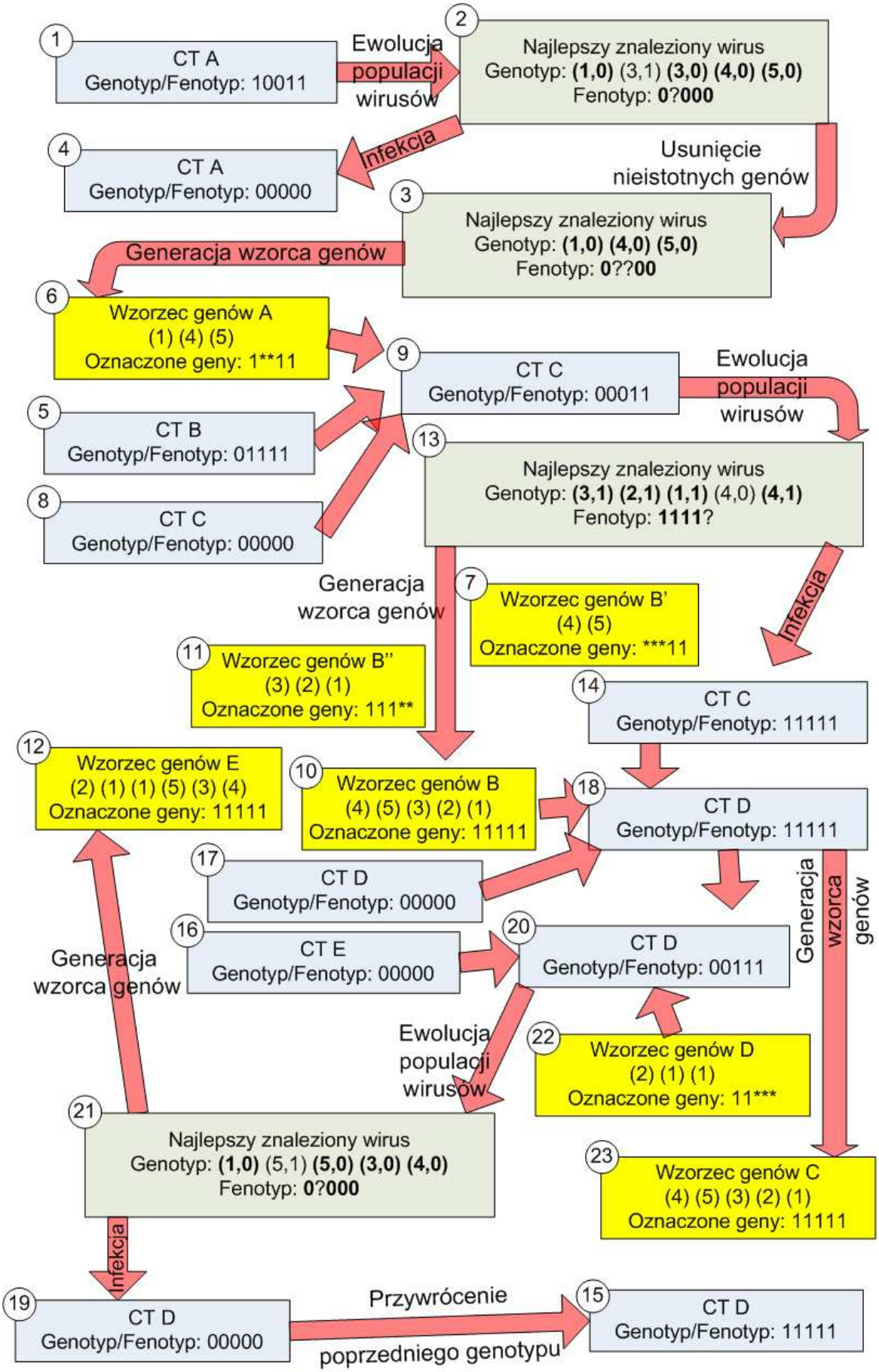
3.5.4 Algorytm MuPPetS – dlaczego i jak działa

Idea wzorców genów przedstawiona w niniejszej pracy i w [57], bazuje na poszukiwaniu i przechowywaniu informacji o grupach genów, które pod pewnymi warunkami (w kontekście tworzonym przez wszystkie lub część pozostałych genów fenotypu) są od siebie bardziej zależne niż od innych genów. Silne zależności pomiędzy genami (czego efektem jest znaczący wpływ na wartość funkcji przystosowania) powodują, że w procesie krzyżowania CT warto wymieniać je wszystkie razem. Należy zauważyć, że ewentualne braki bądź

niedoskonałości wynikłe z tak przeprowadzonego krzyżowania są zwykle naprawiane w efekcie ewolucji populacji wirusów (faza *juxtapositional* algorytmu mGA). Kluczową cechą algorytmu MuPPetS jest to, że dla stworzenia wzorców genów nie są poszukiwane wartości genów, ale jedynie ich pozycje. W świetle przeprowadzonych badań wydaje się, że poszukiwanie wzorców genów pozwala na lepsze przetwarzanie informacji przez populację CT, oraz przypisane im populacje wirusów, niż poszukiwanie schematów genów (ang. *gene templates*). Jednym z powodów przewagi wzorców nad schematami jest fakt, że grupa silnie powiązanych ze sobą genów, może przyjmować różne wartości w zależności od kontekstu tworzonego przez pozostałe geny – w takim przypadku byłby to zupełnie inny schemat, ale wciąż ten sam, lub prawie ten sam wzorzec genów.

Przykład działania wzorców genów i całego algorytmu MuPPetS został zamieszczony na Rys. 18. Zakładamy, że przykład odnosi się do 5-bitowego podproblemu, będącego składnikiem większej całości. Zakładamy również, że 00000 to lokalne, a 11111 globalne optimum rozpatrywanego podproblemu. Na początku, po inicjalizacji algorytmu, istnieje jedynie populacja losowo wygenerowanych CT. Dla jednego z nich (CT A – ramka 1 na Rys. 18) zostaje wygenerowany wirus uzupełniający genotyp CT do lokalnego optimum (00000) (ramka 2). Na podstawie tego wirusa generowany jest wzorzec genów oznaczający 3 spośród 5 genów podproblemu (ramki 3 i 6). W trakcie następnej iteracji algorytmu, dla CT C (ramka 8), który utknął w lokalnym optimum, uruchamiana jest faza 3 (inicjalizacja populacji wirusów). Jeśli w fazie 4 (ewolucja populacji wirusów) CT B (ramka 5) i wzorzec genów A (ramka 6) zostaną wybrane do wymiany genów z CT C, to zmodyfikowany genotyp CT C (ramka 9), będzie zawierał niektóre z „właściwych” genów (czyli takich, które mają wartość ‘1’ i są składnikiem globalnego optimum rozpatrywanego podproblemu). Taki genotyp CT C znacząco zwiększa szanse znalezienia wirusa, który pozwoliłby na znalezienie optymalnego rozwiązania dla podproblemu (11111). Jeśli tak się stanie (ramka 13), to wzorzec genów wygenerowany na podstawie takiego wirusa będzie wskazywać wszystkie lub prawie wszystkie geny podproblemu (wzorzec genów B’’ – ramka 11). Taki wzorzec genów jest następnie sklejany z wzorcem uzyskanym na podstawie operacji wymiany genów z początku fazy 4 z udziałem wzorca genów A i CT B (wzorzec genów B’ – ramka 7). Nowy wzorzec genów B (ramka 10) utworzony z wzorców B’ i B’’ jest następnie wstawiany do puli wzorców genów. Kiedy algorytm MuPPetS znajdzie wzorzec genów wskazujący wszystkie lub prawie wszystkie geny podproblemu i jeśli w populacji CT istnieje CT zawierający optymalne rozwiązanie dla podproblemu, to wymiana optymalnego rozwiązania dla podproblemu staje się dosyć prosta (sytuacja przedstawiona dla CT D – ramki 17 i 18, gdzie nie jest potrzebne uruchomienie ewolucji populacji wirusów) i spowoduje wzrost liczby wzorców genów (wzorzec genów C – ramka 23) wskazujących wszystkie, lub prawie wszystkie geny podproblemu (w tym przypadku geny 1,2,3,4 i 5).

Należy zauważyć, że wzorzec genów wskazujący część, lub nawet wszystkie geny podproblemu może zostać wygenerowany również wtedy, gdy znajdowane są rozwiązania leżące w obszarze optimum lokalnego. Taką sytuację obrazuje proces dalszego przetwarzania CT D (ramki 16, 29, 21, 22), gdzie wzorzec genów D (ramka 22) i CT E (ramka 16) zostają wybrane do operacji wymiany genów na początku fazy 4. W efekcie genotyp CT D powraca do optimum lokalnego (ramka 20). Pomimo tego zostaje wygenerowany wzorzec genów E (ramka 12), który wskazuje wszystkie geny podproblemu. W związku z faktem, że po wykonaniu fazy 4 genotyp CT D zmienił się z optimum globalnego do lokalnego jest prawdopodobne, że wartość funkcji przystosowania dla całego CT D spadła, a w związku z tym genotyp sprzed fazy 4 zostanie przywrócony (ramka 19).



Rys. 18. Przykład działania algorytmu MuPPetS

3.6 Klasyfikacja algorytmów fmGA, BOA i MuPPetS jako algorytmów typu „linkage learning”

Główną nowością, która cechuje algorytm MuPPetS są metody pozyskiwania i użycie wzorców genów. Wzorce genów mają za zadanie wskazywać i zapamiętywać, które grupy genów są ze sobą ściślej powiązane od innych. Fakt ten stawia algorytm MuPPetS wśród metod uczących się powiązań między genami (ang. *linkage learning*) [11, 39, 40]. Na podstawie [11] w tabeli Tabela 10 zaprezentowana jest klasyfikacja algorytmów fmGA, BOA i MuPPetS.

Tabela 10. Klasyfikacja algorytmów typu „linkage learning”

Cecha/algorytm	**BOA**	**fmGA**	**MuPPetS**
Odróżnianie “dobrych” i “złych” powiązań	Wielometryczne	Unimetryczne/ Wielometryczne	Unimetryczne
Reprezentacja powiązań	Wirtualna	Fizyczna	Obie
Sposób przechowywania powiązań	Centralny	Lokalny	Oba

Aby odróżnić „dobrze” (prawidłowo oznaczone powiązania pomiędzy genami) i „źle” oznaczone powiązania pomiędzy genami, algorytm może używać podejścia unimetrycznego (ang. *unimetric*), bazującego wyłącznie na wartości funkcji przystosowania, lub podejścia wielometrycznego (ang. *multimetric*) bazującego na wartości funkcji przystosowania i innych dodatkowych kryteriach. Algorytm fmGA wydaje się używać wyłącznie podejścia unimetrycznego. Jednakże w ramach procedury Kombinatorycznie Kompletnej Inicjalizacji (ang. *PCI - probabilistically complete initialization*, patrz: rozdz. 3.3.2), która zastąpiła w algorytmie fmGA fazę primordial z algrytmu mGA, dokonywane jest filtrowanie bloków budujących, w ramach którego faworyzowane są bloki o krótszej długości. Może to być uznane za rodzaj dodatkowej miary dla oceny jakości powiązań pomiędzy genami. W pracy [11] autorzy wskazują, że umieszcza to algorytm fmGA, gdzieś pomiędzy podejściem unimetrycznym, a wielometrycznym. W przypadku pozostałych dwóch algorytmów opisane powyżej wątpliwości klasyfikacyjne nie występują – bez wątpienia algorytm BOA używa podejścia wielometrycznego, a algorytm MuPPetS unimetrycznego.

Powiązania między genami (ang. *linkage*) mogą być reprezentowane w postaci ulokowania dwóch, lub więcej genów obok siebie w jednym chromosomie (reprezentacja fizyczna – ang. *physical linkage*), albo jako matryce, grafy, lub inne struktury danych opisujące zależności pomiędzy genami (reprezentacja wirtualna – ang. *virtual linkage*). Algorytm MuPPetS używa obu tych technik. Wzorce genów to typowa wirtualna metoda reprezentacji powiązań pomiędzy genami, natomiast wirusy są typową metodą reprezentacji fizycznej.

Informacja o powiązaniach pomiędzy genami może być przechowywana w centralnej bazie danych, lub być rozdystrybuowana w populacji. Znów algorytm MuPPetS używa obu metod – wzorce genów są przechowywane w jednej centralnej puli, natomiast użycie wirusów zapewnia przechowanie wiedzy o powiązaniach pomiędzy genami w całej populacji wirusów.

3.7 Wyniki eksperymentów

Wszystkie wyniki zaprezentowanych w niniejszym podrozdziale zostały również opublikowane w [57]. Głównym celem przeprowadzonych testów, było sprawdzenie

efektywności algorytmu MuPPetS przy rozwiązywaniu problemów o najwyższym stopniu trudności w porównaniu z algorytmami fmGA i BOA, które można zaliczyć do najskuteczniejszych, znanych obecnie, algorytmów rozwiązujących problemy stworzone z konkatenacji funkcji zwodniczych. Należy pamiętać, że algorytm MuPPetS został zaprojektowany do rozwiązywania problemów obliczeniowych występujących w świecie rzeczywistym. Jednak użycie funkcji zwodniczych, powszechnie znanego narzędzia testowego, pozwala na porównanie wszystkich trzech algorytmów w ich niezmienionej formie. Porównanie takie daje również istotne wskazówki odnośnie ogólnych cech charakteryzujących algorytm MuPPetS i pozostałe dwa algorytmy.

W tym miejscu należy również zauważyć, że dobór parametrów dla algorytmu fmGA, był poważną przeszkodą w przeprowadzeniu eksperymentów. Ostatecznie zdecydowano się na użycie parametrów zaproponowanych przez D. Goldberga w [36], jednak rozmiar populacji był wyliczany na podstawie wyników zwróconych przez algorytm MuPPetS w sposób, który faworyzował algorytm fmGA (patrz: rozdz. 3.7.2). W tym miejscu należy zauważyć, że wzór na wyliczenie rozmiaru populacji podany przez D. Goldberga w [36] nie jest użyteczny do zastosowań praktycznych. Wzór zakłada, że użytkownik posiada dużą wiedzę o rozwiązywanym zadaniu (np. liczba podfunkcji, z których zbudowany jest problem), a także nie uwzględnia faktu, że podfunkcje problemu mogą mieć zupełnie różny wpływ na ogólną wartość funkcji przystosowania.

3.7.1 Różnice pomiędzy wynikami uzyskanymi dla algorytmów BOA, fmGA i MuPPetS

Należy zauważyć, że dla wszystkich funkcji zwodniczych użytych do testów w niniejszej pracy (patrz: rozdz. 3.2), optimum lokalne (zbudowane wyłącznie z genów o wartości ‘0’) może być znalezione przez algorytm zachłanny w $n + 1$ krokach, gdzie n jest liczbą bitów potrzebną do zakodowania rozwiązania. Rozwiązanie zwrócone przez algorytm zachłanny jest nazywane „łatwym rozwiązaniem”. Optimum globalne (zbudowane wyłącznie z genów o wartości ‘1’) jest różne od „łatwego rozwiązania” na każdej pozycji. Z tego powodu w wielu pracach, gdzie jako funkcje testowe używane są funkcje zwodnicze, podaje się wyłącznie liczbę, lub odsetek ‘1’ w genotypie uzyskanego rozwiązania [34, 35, 36, 73]. W związku z faktem, że autor chciałby dostarczyć tak pełnej informacji o uzyskanych wynikach, jak to tylko możliwe, pozwalając tym samym na dokonywanie analiz i porównań nieuwzględnionych w niniejszej pracy, prezentowane wyniki będą zawierać nie tylko liczbę genów o wartości ‘1’ w genotypie rozwiązania wynikowego, ale również inne informacje. Należy podkreślić, że użyte funkcje testowe posiadają określone cechy. Na przykład, w tabeli Tabela 29 zaprezentowane są wyniki dla 150-bitowej funkcji typu „spike hard”, zbudowanej wyłącznie z 5-bitowych funkcji zwodniczych. Średnia ocena wyników zwróconych przez algorytmy fmGA, BOA i MuPPetS w 10 przebiegach to odpowiednio: 215,73, 219 i 218,96.Różnica pomiędzy fmGA i BOA to 1,14%, a różnica pomiędzy BOA i MuPPetS to zaledwie 0,01%. Zasadne jest więc pytanie, czy taka, śladowa, różnica rzeczywiście ma jakieś znaczenie. Żeby odpowiedzieć na to pytanie należy zauważyć, że wartość „łatwego rozwiązania”, dla tego przypadku to:

9 * wartość lokalnego optimum dla funkcji “5lh”
+
21* wartość lokalnego optimum dla funkcji “5hh”
=
9 * 0.99 + 21 * 9.88 = 216,39

Teraz jest oczywiste, że średnie rozwiązanie zaproponowane przez algorytm fmGA jest bardzo niskiej jakości, ponieważ jest gorsze nawet od „łatwego rozwiązania".

Innym pytaniem, które można postawić patrząc na wyniki dla wspomnianej 150-bitowej funkcji typu „spike hard" zbudowanej wyłącznie z 5-bitowych funkcji zwodniczych jest, czy różnice to nie szum statystyczny. Jak zostało zaznaczone wcześniej, głównym wskaźnikiem jakości rozwiązania dla konkatenacji funkcji zwodniczych jest nie wartość funkcji, ale liczba genów przyjmujących wartość '1'. W tabeli Tabela 28 średnia wartość dla odsetka genów o wartości '1' w rozwiązaniach proponowanych przez algorytmy fmGA, BOA i MuPPetS to, odpowiednio: 0.294, 1, 0.974. Wartości te wskazują, że wyniki proponowane przez algorytm fmGA były zupełnie inne i bardzo słabe w porównaniu z dwoma pozostałymi algorytmami, spośród których BOA było najlepsze, a MuPPetS zaproponowało wyniki nieznacznie gorsze.

Należy również zastanowić się, czy poszukiwanie rozwiązań, które dają bardzo niewielki procentowy wzrost wartości funkcji przystosowania jest zasadne. Aby odpowiedzieć na to pytanie należy zauważyć, że choć dla wybranego przykładu, różnica w wartości funkcji przystosowania pomiędzy rozwiązaniem optymalnym, a „łatwym rozwiązaniem" to zaledwie około 1,2%, to różnica ta nie wynika ze zmiany wartości kilku genów, ale wszystkich genów w genotypie! A więc nawet, jeśli z punktu widzenia wartości funkcji przystosowania rozwiązania są zbliżone, to z punktu widzenia genotypu są zupełnie różne. Skoro tak, to dla problemów spotykanych w rzeczywistym świecie algorytmy, które potrafią znaleźć optymalne, lub prawie optymalne rozwiązanie z punktu widzenia jakości genotypu, powinny być zdolne do przejścia wąskich gardeł (w przedstawionych w tym rozdziale testach za wąskie gardło należy uznać każdą składową funkcję zwodniczą, ponieważ funkcje te są trudne do rozwiązania) i znalezienia zupełnie nowych rozwiązań o znacząco lepszej jakości.

Powyższa opinia, że drobne udoskonalenia proponowanych przez algorytm rozwiązań, mogą prowadzić do przełomu, dzięki któremu populacja może zasiedlić nową (potencjalnie lepszą) niszę adaptacyjną znajduje potwierdzenie w przeprowadzonych badaniach. Powyższa sytuacja została zaobserwowana przez autora niniejszej pracy dla algorytmów rozwiązujących problemy projektowania przepływu w szkieletowych sieciach komputerowych [76, 77, 78].

3.7.2 Kodowanie, dostrajanie, procedura testowa

Wszystkie algorytmy (MuPPetS, fmGA i BOA) zostały zakodowane w języku C++. Wszystkie części wykonawcze algorytmów, dla których było to możliwe, były wspólne dla MuPPetS i fmGA (np. znaczna część fazy juxtapositional, wszystkie operacje na wirusach). Kod źródłowy algorytmu BOA został ściągnięty ze strony Laboratorium Algorytmów Genetycznych Stanu Illinois IlliGAL. Został on użyty w niezmienionym stanie z wyjątkiem dodania do algorytmu takich elementów jak ograniczenie czasowe i dodatkowych funkcji testowych do zbioru tych, które już były dostępne. Testy przeprowadzono na komputerze AMD Athlon 64 X2 Dual Core Processor 3800+, 2 GB RAM, z systemem operacyjnym Windows XP SP2. Dla każdej funkcji testowej, każda z badanych metod została wykonana 10 razy w pierwszej fazie eksperymentów (funkcje bez „ogona") i 5 razy w drugiej części eksperymentów (funkcje z „ogonem").

Algorytm fmGA w fazie primordial wykonuje pewną liczbę operacji, która jest z góry określona. Z tego powodu można oszacować ile razy w fazie primordial zostanie wyliczona wartość funkcji przystosowania. Aby umożliwić porównanie efektywności MuPPetS i fmGA, zdecydowano się najpierw wykonać procedurę dostrajania i testy dla algorytmu MuPPetS, a

dopiero później wykonać testy dla fmGA, przy czym ustawienia dla fmGA bazowały na liczbie wyliczeń funkcji przystosowania użytej przez MuPPetS. Taka procedura faworyzuje algorytm fmGA, ponieważ zastępuje warunek stopu bazujący na czasie (a więc na rzeczywistym zapotrzebowaniu na moc obliczeniową) warunkiem stopu bazującym na liczbie wyliczeń funkcji przystosowania (miarodajna ocena złożoności obliczeniowej - patrz punkt 3.7.3.1). Sytuacja taka faworyzuje algorytm fmGA, ponieważ w trakcie jego pracy wykonywanych jest wiele operacji nie mających związku z wyliczaniem wartości funkcji przystosowania (na przykład jeśli mechanizm „threshold" zdecyduje, że osobnik ma być kopiowany do następnej populacji zamiast zostać skrzyżowany).

Procedura dostrajania algorytmu MuPPetS została przeprowadzona wyłącznie dla pojedynczej populacji wirusów (bez użycia wzorców genów). Poszukiwane były takie ustawienia dla fazy ewolucji populacji wirusów, aby umożliwiała ona poprawianie CT, do którego populacja wirusów jest przypisana, przy minimalnej wymaganej mocy obliczeniowej. Pozostałe parametry zostały określone na podstawie doświadczenia autora, oraz wcześniejszych testów przeprowadzonych w toku prac nad algorytmem. Ostatecznie zdecydowano się na ustawienia zaprezentowane w tabeli

Tabela 11. Parametry dla algorytmu MuPPetS

Nazwa parametru	Wartość
Liczba wzorców w puli wzorców genów	200
Minimalna długość wzorca genów	3
Liczba pokoleń wirusów	5
Liczba wirusów przypisanych do CT	400
Współczynnik redukcji liczby wirusów w kolejnym pokoleniu wirusów	0,8
Prawdopodobieństwo cięcia	0,05
Prawdopodobieństwo sklejania	0,5
Prawdopodobieństwo mutacji	0,1
Prawdopodobieństwo usunięcia genu	0,1
Prawdopodobieństwo dodania genu	0,1

Ograniczenie czasowe było różne w zależności od funkcji testowej. Maksymalny czas przeznaczony na obliczenia podany jest w tabeli

Tabela 12. Ograniczenia czasowe użyte dla algorytmów MuPPetS i BOA

Funkcja testowa	Ograniczenie czasowe [s]
30-bitowa	200
50-bitowa	1200
150-bitowa (only 5) flat hard	4500
150-bitowa (only 5) spike strong and spike hard	3500
150-bitowa waszystkie pozostałe	2500
300-bitowa	7000

Tabela 13. Parametry dla algorytmu fmGA

Nazwa parametru	Wartość
Liczba osobników sprawdzanych przez mechanizm 'threshold'	10% populacji
Liczba generacji w fazie juxtapositional	20
Prawdopodobieństwo cięcia	0,03

Prawdopodobieństwo sklejania	1
Prawdopodobieństwo mutacji	0
Prawdopodobieństwo dodania genu	0
Prawdopodobieństwo usunięcia genu	0

Faza primordial algorytmu fmGA była uruchamiana dla parametru k=1,2,3, lub k=1,2,3,4,5, w zależności od funkcji testowej. Wartość l' była zmniejszana o 1 przy każdym uruchomieniu fazy primordial (patrz punkt: 3.3.2). Rozmiar populacji dla danej funkcji testowej, był tak określany, żeby oczekiwana liczba wyliczeń funkcji przystosowania (przy pominięciu mechanizmu „threshold") w fazie primordial była równa największej liczbie wyliczeń funkcji przystosowania dla algorytmu MuPPetS we wszystkich 10 próbach. Liczbę osobników w populacji wyliczano zgodnie ze wzorem:

$$pop_{size} = \frac{eval}{order_{\max} * (length - order_{\max}) + (order_{\max}^{\,2} - order_{\max})/2},$$

gdzie

pop_{size} to obliczany rozmiar populacji dla fmGA, $eval$ jest największą liczbą wyliczeń funkcji przystosowania dla MuPPetS we wszystkich 10 próbach dla danej funkcji testowej, $length$ to długość funkcji testowej, a $order_{\max}$ jest maksymalną długością funkcji zwodniczych, z których zbudowana jest funkcja testowa (43)

Dodatkowo jeśli wszystkie rozwiązania zaproponowane przez algorytm MuPPetS były optymalne, to rozmiar populacji dla algorytmu fmGA był podwajany. Rozmiar populacji dla algorytmu fmGA dla poszczególnych funkcji testowych został zaprezentowany w tabeli Tabela 30.

W przypadku algorytmu fmGA wydaje się pożądane, aby zmniejszać prawdopodobieństwo cięcia wraz ze wzrostem długości funkcji testowej. Intuicja podpowiada, że w przeciwnym przypadku algorytm fmGA może w fazie juxtapositional mieć problem ze skonstruowaniem pełnego rozwiązania z bloków budujących, które wyprodukował w fazie primordial. Przeprowadzone testy wykazały jednak, że niższe wartości prawdopodobieństwa cięcia dla funkcji 50- i 150-bitowych, powodują obniżenie efektywności algorytmu. W związku z tym prawdopodobieństwo cięcia pozostało na niezmienionym poziomie dla wszystkich długości funkcji testowych.

Algorytm messy GA można opisać jako fmGA z dodatkowymi wadami i w ogóle nie był on testowany. Należy spodziewać się, że wyniki zaproponowane przez mGA będą nielepsze niż te, które zostały zaproponowane przez fmGA. Wystarczającym wyjaśnieniem dla pominięcia mGA w testach powinien być również fakt, że liczba ciągów, która musi być wygenerowana w fazie primordial algorytmu mGA dla 300-bitwoego problemu i k=5 to około $5.6*10^{11}$ (w wynikach prezentowanych w niniejszej pracy i w [57], dla fazy primordial algorytmu fmGA potrzeba nie więcej niż $4*10^4$ ciągów, a algorytm MuPPetS używa nie więcej niż $5{,}4*10^7$ wyliczeń funkcji przystosowania dla wygenerowania rozwiązania). Z tych samych powodów, dla tej samej funkcji testowej z „ogonem", która jest problemem 600-bitowym, algorytm mGA potrzebowałby do przeprowadzenia fazy primordial około $1.9*10^{13}$ ciągów.

Wybór ustawień dla algorytmu BOA, podobnie jak dla fmGA, nie był zadaniem prostym. Pierwszym problemem był wybór wartości dla parametru *k*, ponieważ funkcje testowe zawierały sklejenia funkcji zwodniczych o różnej długości (niektóre zawierały funkcje zwodnicze o długości 5, a inne nie). Innym problemem był wybór rozmiaru populacji. Zdecydowano, że dla każdej grupy funkcji testowych (grupowanie odbywało się na podstawie długości funkcji) procedura dostrajania będzie odbywać się oddzielnie i zostanie przeprowadzona dla najtrudniejszej funkcji z całej grupy (a więc dla funkcji typu 'spike hard'). Taka procedura testowa wydaje się faworyzować algorytm BOA – w przypadku problemów, które spotykamy w świecie rzeczywistym zwykle nie wiemy wiele o ich własnościach, a więc nie można wskazać najtrudniejszego. Procedura dostrajania objęła rozmiary populacji liczące 5 000, 10 000, 20 000, 40 000, 80 000 i 120 000 osobników. W związku z faktem, że dla parametru *k*=2 BOA nie był w stanie znaleźć żadnego optymalnego rozwiązania dla funkcji 150-bitowej, zrezygnowano z używania tego ustawienia na korzyść *k*=4. Należy zauważyć, że fakt ten niewątpliwie faworyzuje algorytm BOA, ponieważ używa on idealnej wartości parametru k, w odniesieniu do rozwiązywanego problemu (wartość *k*=4 jest idealna dla problemu złożonego ze sklejenia funkcji zwodniczych, które nie mają długości większej niż 5). W przypadku rzeczywistego problemu takie perfekcyjne ustawienie parametru *k* byłoby najprawdopodobniej niemożliwe. Pozostałe parametry zostały ustawione tak jak w [73]. Pełne zestawienie parametrów dla algorytmu BOA podane jest w tabeli Tabela 14.

Tabela 14. Parametry dla algorytmu BOA

Nazwa parametru	Wartość
Population	80 000 (dla funkcji 150- i 300-bitowych) 10 000 (dla funkcji 50-bitowych) 5 000 (dla funkcji 30-bitowych)
k	4
τ (% populacji zastępowanej przez nowe osobniki)	50%
Prawdopodobieństwo mutacji	0,01
Miara dla porównywania jakości sieci bayesowskich	Metryka K2

W drugiej części eksperymentów, przeprowadzonych dla algorytmów MuPPetS i BOA, użyto takich samych ustawień i ograniczeń czasowych.

3.7.3 Wyniki i komentarze

Zaprezentowane w [57] i w niniejszej pracy wyniki dla algorytmu fmGA są w niektórych miejscach znacząco różne od wyników zaprezentowanych w [36]. Ten fakt może początkowo zaskakiwać, należy jednak zauważyć, że pomimo podobieństw istnieją znaczące różnice pomiędzy testami fmGA wykonanymi przez zespół D. Goldberga i tymi prezentowanymi w niniejszej pracy. Jedną z głównych różnic jest procedura inicjalizacji CT – Goldberg inicjalizuje samymi zerami, w niniejszej pracy proces ten jest całkowicie losowy. Jest prawdą, że CT zawierający w genotypie same '0' jest teoretycznie najgorszym punktem startu dla algorytmu genetycznego (ponieważ jest to optimum lokalne, dla którego genotyp jest różny od optimum globalnego na każdej pozycji). Taki „wyzerowany" CT jest jednak również bardzo wygodnym punktem startu dla algorytmu fmGA, ponieważ powoduje, że niemożliwe jest zdominowanie fazy primordial przez ciągi bitów, które należą do wielu różnych bloków

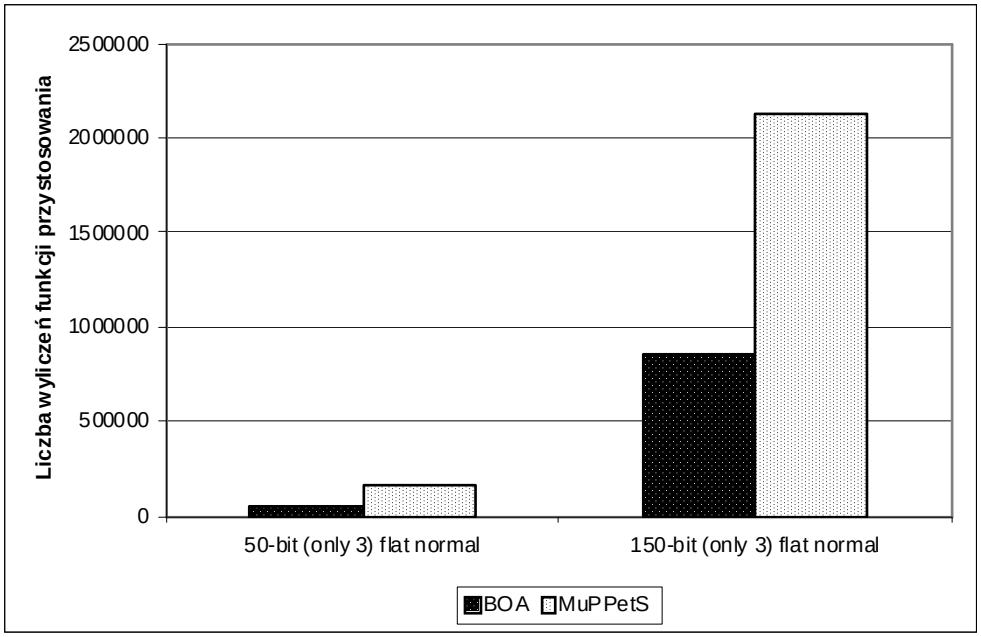
budujących. Na przykład jeśli funkcja testowa jest zbudowana z 3 funkcji zwodniczych o długości 3, ciąg bitów [(1,0) (4,0) (7,1)] będzie bardzo dobry dla CT o genotypie 100 100 000, a bardzo słaby dla „wyzerowanego" CT o genotypie 000 000 000. Drugą istotną różnicą są funkcje zwodnicze użyte do budowania funkcji testowych. Nie wiemy jakie dokładnie funkcje zwodnicze zostały użyte w pracy [36]. Wreszcie trzecią znaczącą różnicą są użyte w niniejszej pracy funkcje testowe, które są bardzo trudne nie tylko ze względu na użycie funkcji zwodniczych, ale również ze względu na:

- minimalne różnice pomiędzy wartościami lokalnego i globalnego optimum dla funkcji zwodniczej,
- użycie wielu funkcji zwodniczych o różnej długości i różnym wpływie na wartość funkcji testowej

Wymienione powyżej fakty sprawiają, że wszystkie funkcje testowe typu 'hard' i typu 'spiked', użyte w niniejszej pracy powinny stanowić bardzo trudny do rozwiązania problem dla każdego algorytmu.

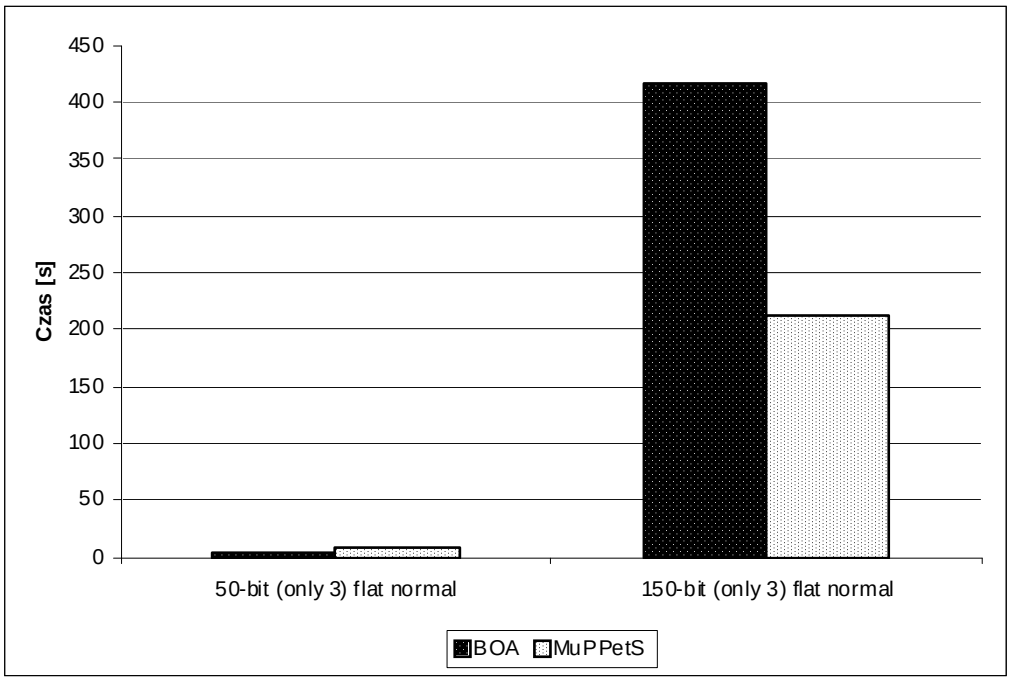
3.7.3.1 Złożoność obliczeniowa dla algorytmów BOA i MuPPetS

W wielu publikacjach dotyczących problematyki algorytmów ewolucyjnych jedyną miarą użytej przez algorytm mocy obliczeniowej jest liczba wyliczeń funkcji przystosowania. Jest to rozsądne dla algorytmów, które poza wyliczaniem wartości funkcji przystosowania nie wykonują innych, wymagających znacznej mocy obliczeniowej (a więc czasu) operacji. Przykładem takiego algorytmu jest klasyczny algorytm genetyczny (ang. *simple GA*). W niniejszej pracy za podstawową miarę złożoności obliczeniowej przyjęto czas obliczeń. Jest to uprawnione, ponieważ wszystkie testy przeprowadzono na tym samym komputerze. Dobrym przykładem, jak różne mogą być proporcje pomiędzy rzeczywiście użytym nakładem mocy obliczeniowej, a liczbą wyliczeń funkcji przystosowania jest porównanie wyników dla 50- i 150-bitowych funkcji typu 'flat normal' złożonych z funkcji zwodniczych o długości 3. Porównanie liczby wyliczeń funkcji przystosowania dla algorytmów BOA i MuPPetS jest przedstawione na Rys. 19. Jak widać dla osiągnięcia tego samego wyniku (oba algorytmy znalazły optymalne rozwiązanie we wszystkich 10 próbach) liczba użytych wyliczeń funkcji przystosowania jest znacznie niższa w przypadku BOA.



Rys. 19 Porównanie liczby wyliczeń funkcji przystosowania użytej przez algorytmy BOA i MuPPetS przy poszukiwaniu rozwiązania dla funkcji testowych ‘50-bit (only 3) flat normal’ i ‘150-bit (only 3) flat normal’

Na Rys. 20 przedstawiono porównanie czasu obliczeń użytego przez algorytmy BOA i MuPPetS przy poszukiwaniu rozwiązywania dla tych samych funkcji testowych (‘50-bit (only 3) flat normal’ i ‘150-bit (only 3) flat normal’). Jak widać pomimo, że dla funkcji 150-bitowej liczba wyliczeń funkcji przystosowania jest ponad dwa razy większa w przypadku algorytmu MuPPetS, to czas obliczeń użyty przez MuPPetS jest około dwa razy krótszy!



Rys. 20 Porównanie czasu obliczeń dla algorytmów BOA i MuPPetS przy poszukiwaniu rozwiązania dla funkcji testowych ‘50-bit (only 3) flat normal’ i ‘150-bit (only 3) flat normal’

Jak widać na podstawie zaprezentowanego powyżej przykładu, czas obliczeń używany przez algorytm (a więc rzeczywisty nakład obliczeniowy) rośnie wraz ze wzrostem długości kodowania problemu znacznie szybciej dla BOA niż dla MuPPetS. Dla takiej samej jak w [73] metody konstrukcji sieci bayesowskiej, złożoność obliczeniowa tej części algorytmu BOA wynosi: $O(k2^k n^2 N + kn^3)$, gdzie *n* to rozmiar problemu, *N* jest liczbą osobników w populacji, a *k* to maksymalna liczba poziomów w drzewie. Ponieważ dla problemu 50-bitowego *N = 10 000*, dla 150-bitowego *N = 80 000* i *N>>n*, czas poświęcony na budowanie sieci bayesowskiej przy każdej iteracji, powinien być około 72 razy większy dla problemu 150-bitowego, niż dla problemu 50-bitowego.

Średni czas potrzebny algorytmowi MuPPetS przy każdej iteracji to suma czasu potrzebnego na przeprowadzenie operacji na CT (jak na przykład generacja wzorców genów i likwidacja CT o identycznych genotypach) i czasu potrzebnego na ewolucję populacji wirusów:

$$M(N_{ct}, N_{vir}, N_{gen}) = N_{CT} * T_{CT} + N_{CT} * N_{vir} * N_{gen} * T_{vir},$$

gdzie N_{CT} jest średnią liczbą CT w trakcie przebiegu algorytmu, T_{CT} jest czasem potrzebnym na wykonanie operacji dla poszczególnych CT, N_{vir} jest średnią liczbą wirusów w trakcie ewolucji populacji wirusów (liczba wirusów maleje wraz z kolejnymi iteracjami), N_{gen} to liczba iteracji dla ewolucji populacji wirusów, T_{vir} to czas potrzebny na przeprowadzenie operacji na pojedynczym wirusie (wyliczenie wartości funkcji przystosowania, działanie operatorów cięcia i sklejania, etc.). (44)

Generacja wzorców genów jest operacją, której złożoność obliczeniowa nie jest zależna od rozmiaru problemu i czas potrzebny na jej przeprowadzanie jest podobny dla każdej iteracji. Usuwanie CT o tych samych genotypach zależy od liczby CT. Liczba porównań pomiędzy genotypami poszczególnych CT wynosi w najgorszym razie (jeśli wszystkie CT zostały zmodyfikowane w efekcie ewolucji populacji wirusów, a więc kontrolę genotypu nalezy przeprowadzić dla każdego CT) $\frac{N_{CT} * (N_{CT} - 1)}{2}$. T_{CT} jest zależne przede wszystkim od N_{CT} i $N_{CT} << N_{vir} * N_{gen}$, w związku z tym pierwsza część sumy $M(N_{ct}, N_{vir}, N_{gen})$ może zostać zignorowana, co daje:

$$M(N_{ct}, N_{vir}, N_{gen}) = N_{CT} * N_{vir} * N_{gen} * T_{vir} \quad (45)$$

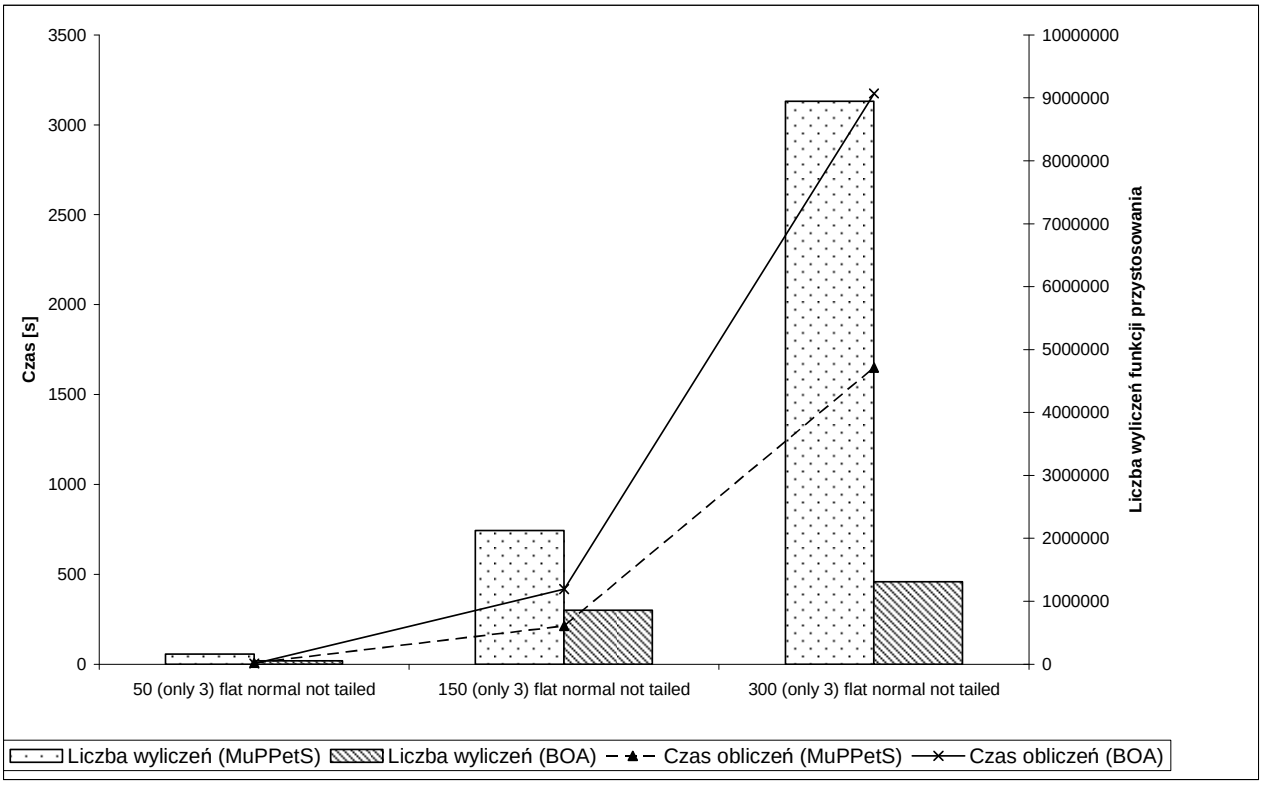
W związku z faktem, że T_{vir} jest stałe, złożoność obliczeniowa pojedynczej iteracji algorytmu MuPPetS wynosi *O(N)*, gdzie $N = N_{CT} * N_{vir} * N_{gen}$.

Należy jednak zauważyć, że *N* zależy od rozwiązywanego problemu w ogóle, a nie jedynie od rozmiaru rozwiązywanego problemu. Im trudniejszy jest problem (choć jest to pojęcie właściwie niedefiniowalne) tym bardziej znaczący będzie wzrost *N* w miarę wzrostu długości problemu. Dlatego w następnym podrozdziale zostanie przeprowadzone porównanie, w jaki sposób zapotrzebowanie na moc obliczeniową dla algorytmów BOA i MuPPetS zależy od typu funkcji testowej.

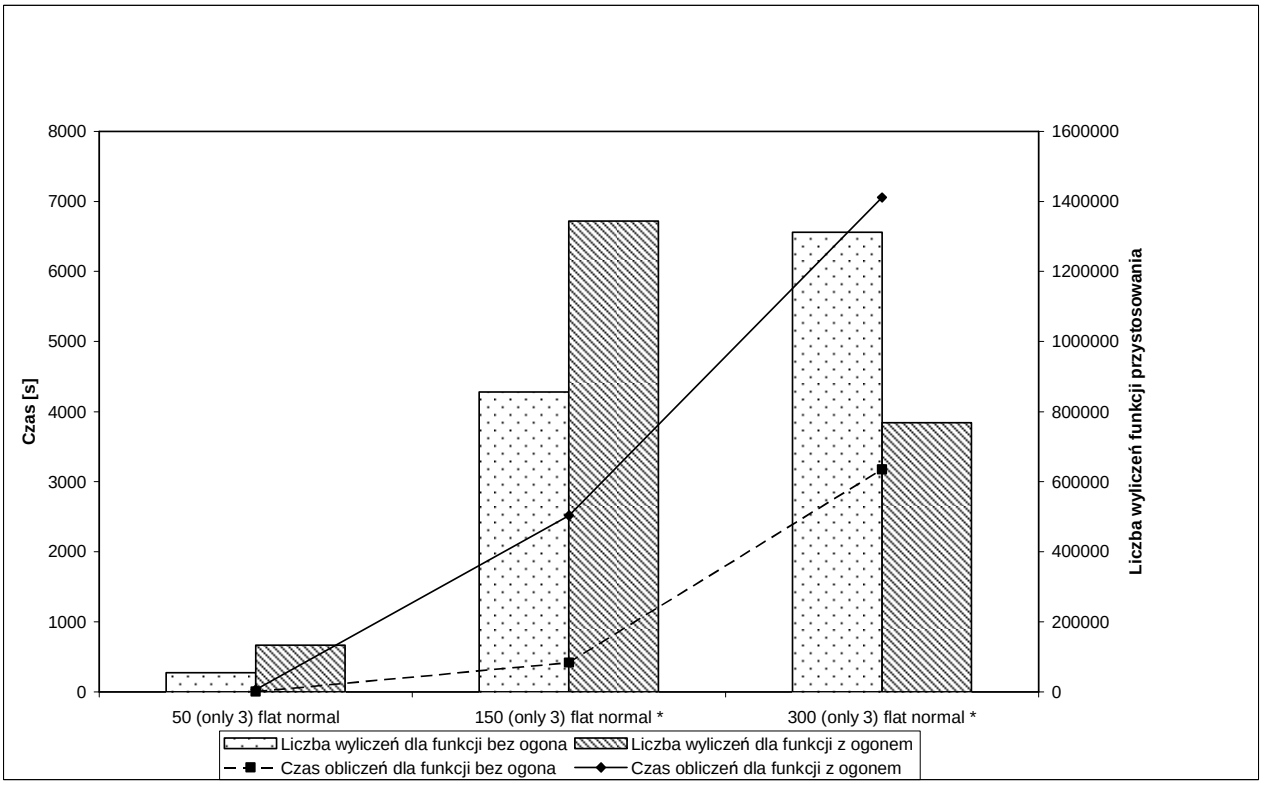
3.7.3.2 Złożoność obliczeniowa i liczba wyliczeń funkcji przystosowania.

Grupa funkcji typu ‘flat normal’ bez ogona jest dobra do porównań pomiędzy algorytmami MuPPetS i BOA, ponieważ dla wszystkich funkcji z tej grupy oba algorytmy zawsze znajdowały rozwiązania optymalne, a więc jakość wyników proponowanych przez oba algorytmy była identyczna. Na Rys. 21 zaprezentowano czas obliczeń i liczbę wyliczeń funkcji przystosowania dla obu algorytmów w zależności od długości funkcji testowej. Jak można zauważyć, algorytm MuPPetS jest około 2 razy szybszy od BOA, z drugiej jednak strony liczba wyliczeń funkcji przystosowania jest kilkakrotnie większa niż w przypadku BOA i ta różnica zwiększa się wraz ze wzrostem długości problemu. Powyższa zależność wskazuje na fakt, że im dłuższy jest czas potrzebny na wyliczenie wartości funkcji przystosowania, tym bardziej efektywny będzie algorytm BOA w porównaniu z MuPPetS. Z drugiej strony czas obliczeń, który jest niezbędny dla BOA do znalezienia rozwiązania rośnie znacząco wraz ze wzrostem długości problemu bez względu na to czy problem jest „trudny”, czy „łatwy” do rozwiązania. Taka sytuacja jest pokazana na *dla funkcji 150- i 300-bitowych z ogonem BOA nie było w stanie znaleźć rozwiązania optymalnego

Rys. 22. Czas użyty przez BOA jest kilka razy dłuższy dla problemu 150-bitowego, do którego został doklejony bardzo łatwy do rozwiązania „ogon” (konkatenacja funkcji o długości 1 – im więcej jedynek w genotypie, tym większa wartość funkcji; patrz rozdz. 3.2). Liczba wyliczeń funkcji przystosowania dla problemu 150-bitowego rośnie mniej niż dwa razy, podczas gdy czas obliczeń wydłuża się kilkukrotnie. Jeszcze ciekawsza sytuacja ma miejsce dla problemu o długości 300 bitów – czas obliczeń wzrasta dwa razy i osiąga maksymalny dozwolony limit 7000 sekund, przy jednoczesnym dwukrotnym spadku liczby wyliczeń funkcji przystosowania! Dodatkowy czas i część czasu, który wcześniej został poświęcony na wyliczanie wartości funkcji przystosowania został zużyty na budowanie sieci bayesowskiej. Należy również podkreślić, że BOA był bardzo daleki od znalezienia optymalnego rozwiązania, dla 300-bitowej funkcji ‘(only 3) flat normal’ liczba ‘1’ w genotypie była niższa niż 70%.



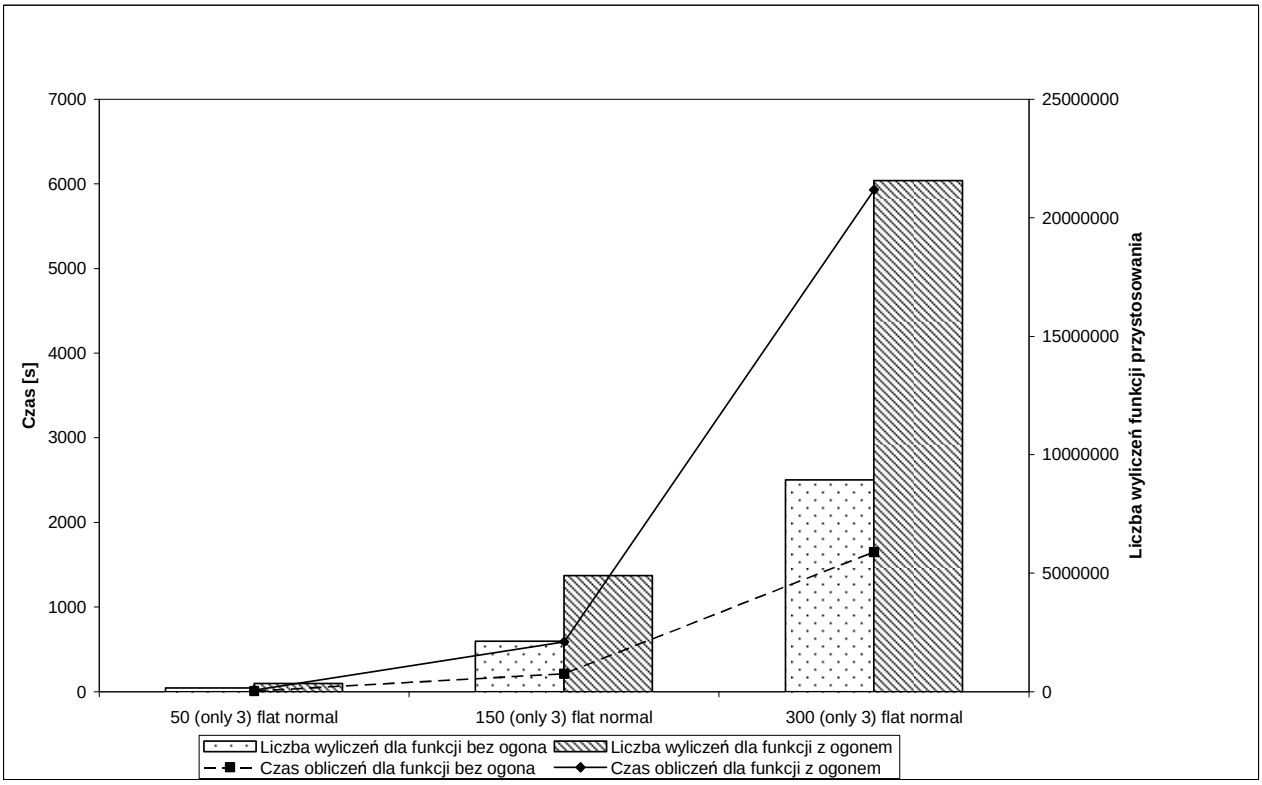
Rys. 21. Zależność pomiędzy czasem obliczeń i liczbą wyliczeń funkcji przystosowania, dla algorytmów MuPPetS i BOA dla grupy funkcji testowych ‘(only 3) flat normal’ bez ogona



*dla funkcji 150- i 300-bitowych z ogonem BOA nie było w stanie znaleźć rozwiązania optymalnego

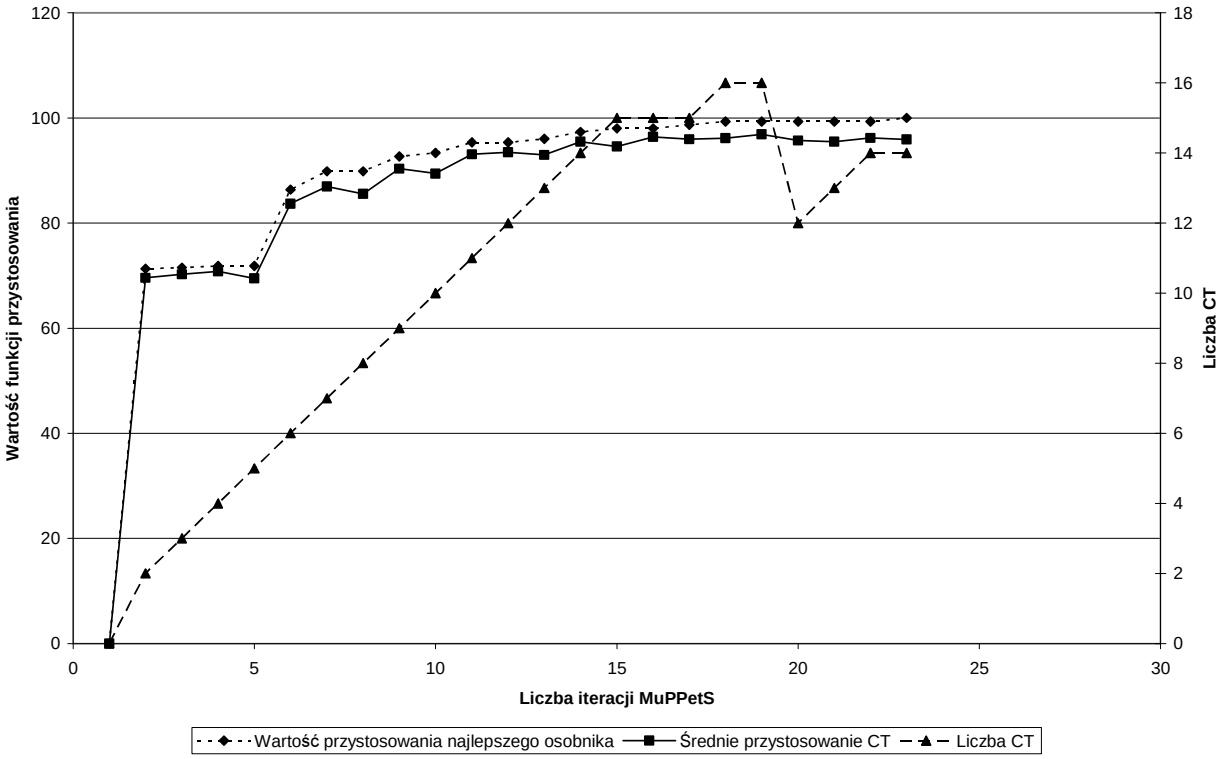
Rys. 22. Zależności pomiędzy czasem obliczeń i liczbą wyliczeń funkcji przystosowania dla algorytmu BOA

Rys. 23 prezentuje zależność złożoności obliczeniowej dla algorytmu MuPPetS i jest analogiczny do Rys. 22. Można zauważyć istotne różnice – wzrost czasu obliczeń jest proporcjonalny do liczby wyliczeń funkcji przystosowania. Co więcej algorytm MuPPetS znalazł optymalne rozwiązania dla funkcji z doklejonymi ogonami we wszystkich próbach. Na podstawie przedstawionych w niniejszej pracy wyników można przyjąć, że dodatkowy czas potrzebny na rozwiązanie funkcji z doklejonym „ogonem" został zużyty na znalezienie zależności pomiędzy odpowiednimi grupami genów (np. poprzez odnalezienie odpowiednich wzorców genów) oraz wymianę tych informacji pomiędzy koewoluującymi CT.

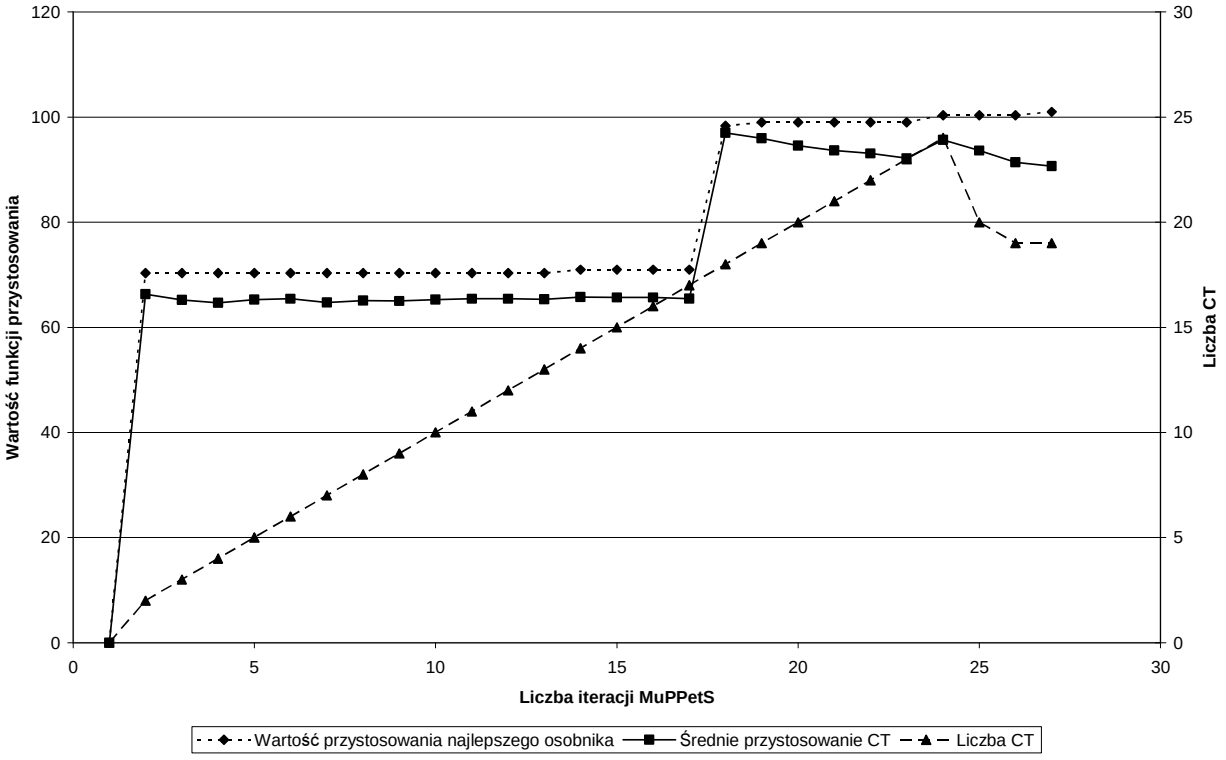


Rys. 23. Zależności pomiędzy czasem obliczeń i liczbą wyliczeń funkcji przystosowania dla algorytmu MuPPetS

Na Rys. 24 i Rys. 25 przedstawiony został przebieg algorytmu MuPPetS dla funkcji z i bez „ogona". Oba przebiegi są podobne, a główną różnicą jest liczba CT, która dla funkcji z „ogonem" jest około 1,5 raza większa. W obu przypadkach algorytm MuPPetS znalazł rozwiązanie optymalne.



Rys. 24. Jeden z przebiegów algorytmu MuPPetS dla funkcji testowej ‘300-bit (only 3) flat normal’



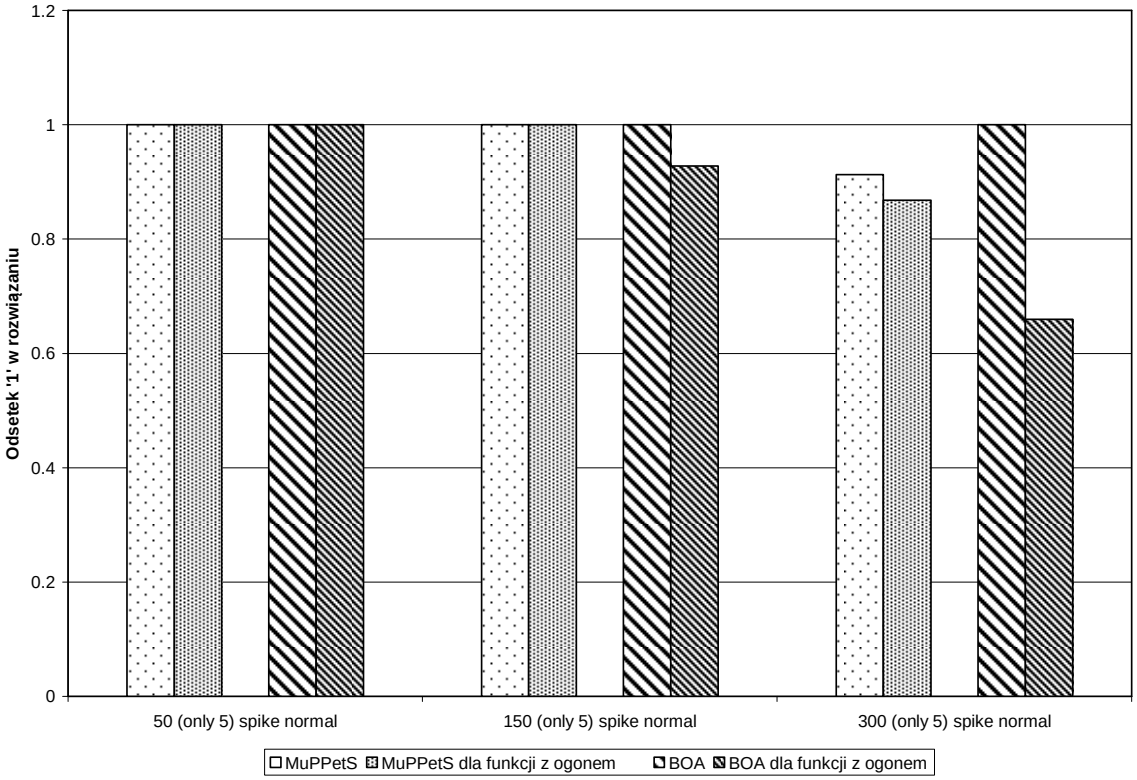
Rys. 25. Jeden z przebiegów algorytmu MuPPetS dla funkcji testowej ‘300-bit (only 3) flat normal’ z ogonem

Jest oczywiste, że niewielka liczba wyliczeń wartości funkcji przystosowania niezbędna dla otrzymania wyników dobrej jakości (jakkolwiek pojęcie „dobry" byłoby rozumiane – optymalny, satysfakcjonujący, etc.) jest zaletą algorytmu BOA. Jednakże silna zależność pomiędzy czasem obliczeń, a długością problemu jest istotną wadą. Należy pamiętać, że problemy obliczeniowe występujące w praktyce, zwykle wymagają długiego kodowania. Ponadto fragmenty, które są bardzo trudne do rozwiązania zwykle występują z fragmentami znacznie łatwiejszymi do rozwiązania[*] (patrz: rozdz. 2.4.6 cecha C). Dla takich problemów (jak na przykład problem projektowania przepływu w szkieletowych sieciach komputerowych [76, 77, 78]) algorytm BOA jest mało użyteczny, z drugiej strony MuPPetS wydaje się być bardzo obiecującym narzędziem.

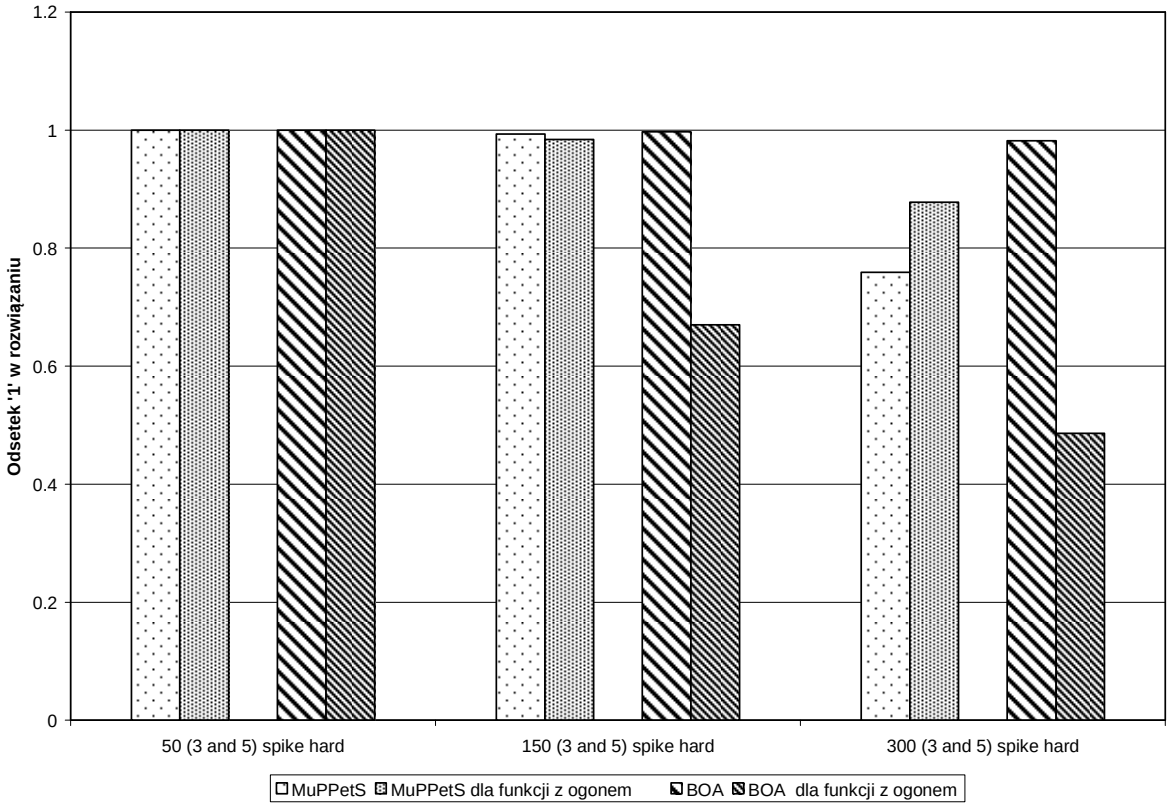
3.7.3.3 Jakość wyników zaproponowanych przez poszczególne algorytmy.

Na Rys. 26 i Rys. 27 zaprezentowano porównanie odsetka '1' (ang. *unitation*) w wynikach zwróconych przez algorytmy MuPPetS i BOA dla funkcji różnego typu i długości. Dla algorytmu MuPPetS odsetek '1' w rozwiązaniach jest podobny dla funkcji z „ogonem" i bez i nieznacznie spada wraz ze wzrostem długości problemu. Na Rys. 27 odsetek '1' nieznacznie wzrasta, ponieważ algorytm MuPPetS „wypełnił" '1' cały „ogon", co spowodowało wzrost ogólnego odsetka '1' w rozwiązaniu. Wyniki zwrócone przez BOA zachowują się w inny sposób – dla funkcji bez „ogona" spadek odsetka '1' wraz ze wzrostem długości problemu jest minimalny lub żaden. Jednak dla funkcji z „ogonem" wyniki są znacząco gorsze niż w przypadku MuPPetS. Dla funkcji z „ogonem" algorytm MuPPetS nie tylko zwraca rozwiązania lepszej jakości, ale również działa znacząco szybciej niż algorytm BOA, co zostało zaprezentowane na Rys. 28.

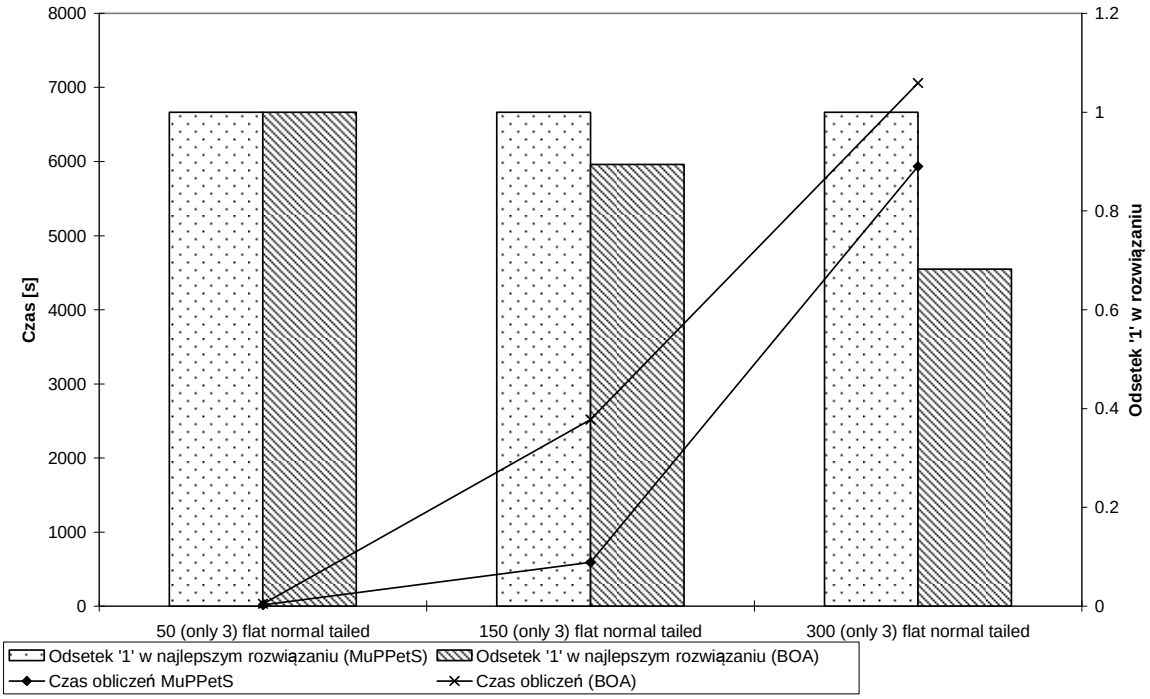
[*] nie oznacza to jednak, że te fragmenty są w pełni separowalne, jak w przypadku funkcji testowych użytych w niniejszej pracy



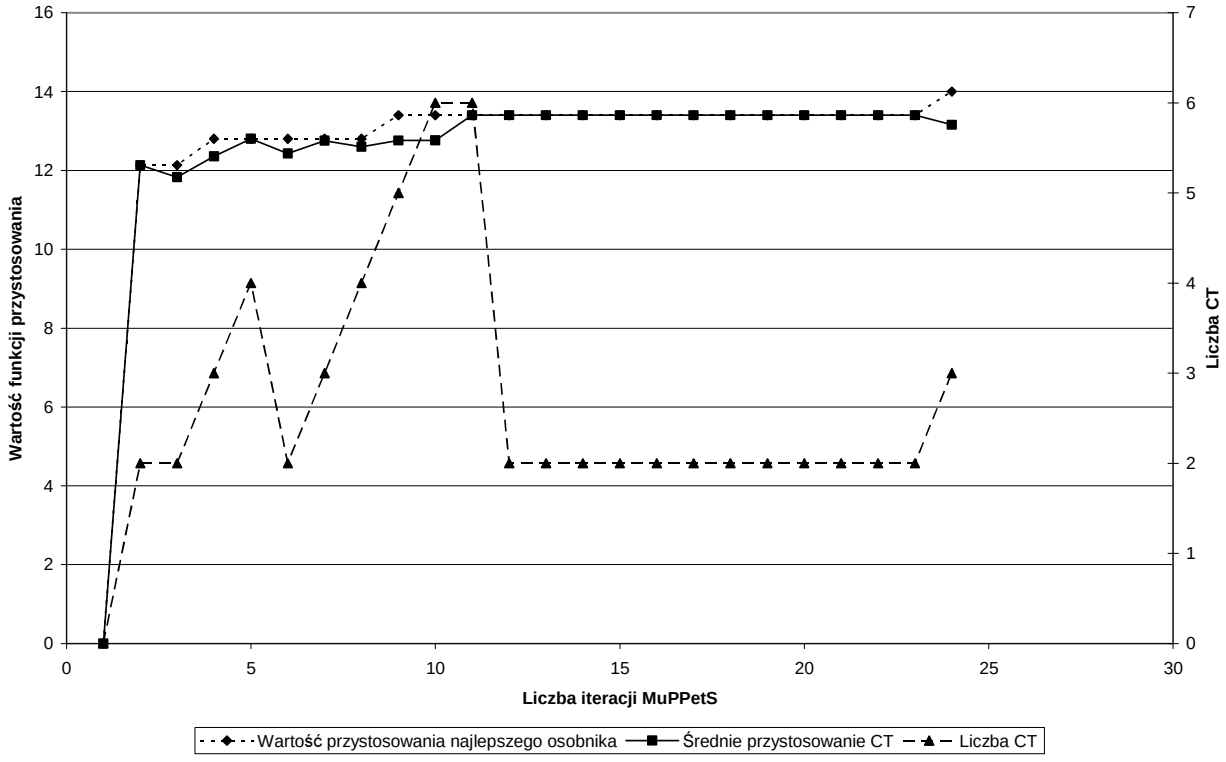
Rys. 26. Porównanie odsetka ‘1’ w rozwiązaniach algorytmów MuPPetS i BOA dla funkcji typu ‘spike normal’



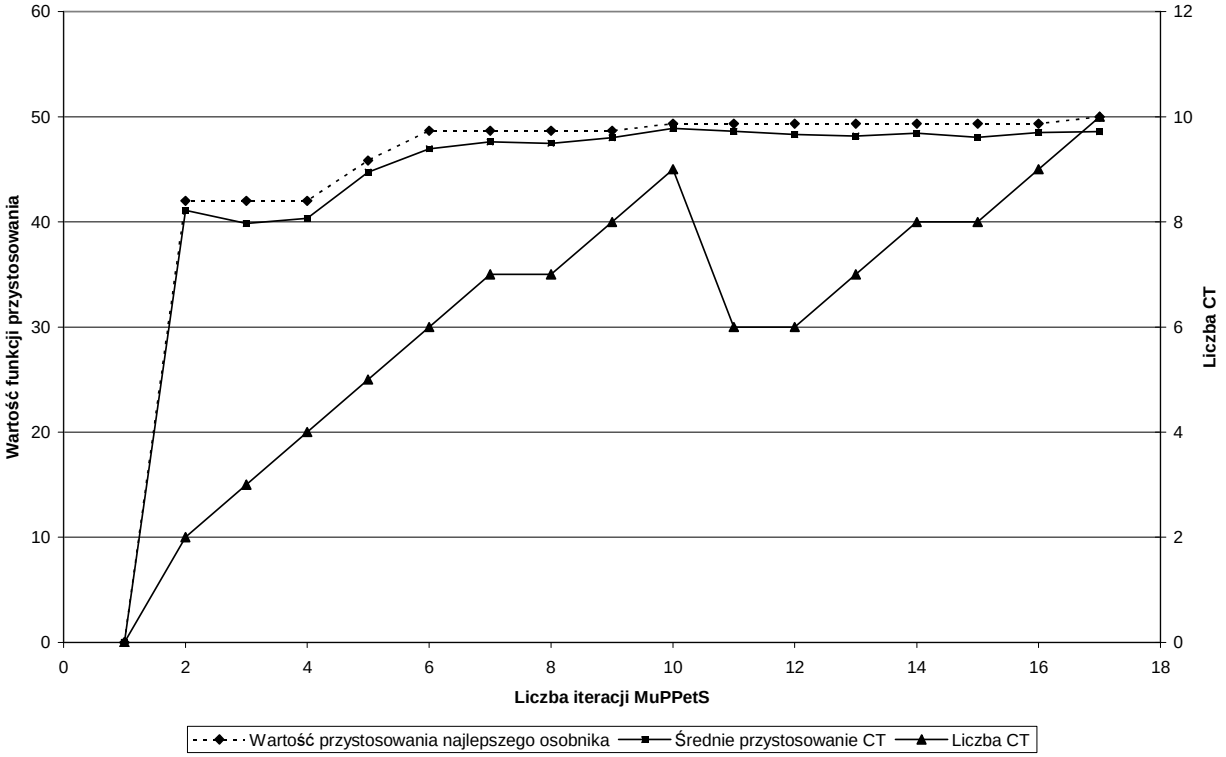
Rys. 27. Porównanie odsetka ‘1’ w rozwiązaniach algorytmów MuPPetS i BOA dla funkcji typu ‘spike hard’



Rys. 28. Porównanie czasu obliczeń i jakości wyników algorytmów MuPPetS i BOA dla funkcji z ogonem



Rys. 29. Jeden z przebiegów algorytmu MuPPetS dla funkcji testowej ‘50 (3 and 5) flat strong’



Rys. 30. Jeden z przebiegów algorytmu MuPPetS dla funkcji testowej '150 (only 3) flat strong'

3.7.3.4 Wyniki – podsumowanie

Wyniki eksperymentów przeprowadzonych na algorytmach MuPPetS, BOA i fmGA, omówionych w niniejszym rozdziale, zostały zaprezentowane w tabelach: Tabela 28, Tabela 29, Tabela 30 (Załącznik A - wyniki). Eksperymenty zostały podzielone na dwie części: eksperymenty na funkcjach bez „ogona" i eksperymenty na funkcjach z „ogonem". Pierwsza część eksperymentów pokazuje problemy algorytmu fmGA (Tabela 28) w uzyskaniu satysfakcjonującego wyniku, kiedy funkcje bazowe (patrz: punkt 3.2) mają różny wpływ na ogólną wartość funkcji przystosowania. Wszystkie funkcje typu „spike" były bardzo trudnymi zadaniami do rozwiązania dla fmGA. Nawet funkcje o długości 30 bitów były zbyt trudne, aby fmGA znalazł rozwiązanie optymalne we wszystkich przeprowadzonych próbach. Należy zauważyć, że nakład obliczeniowy używany przez fmGA (mierzony czasem obliczeń) był około 10 razy większy, niż ten użyty przez algorytm MuPPetS. Należy oczekiwać, że w drugiej części eksperymentów, gdzie do funkcji testowych dodano „ogon" algorytm fmGA radziłby sobie jeszcze gorzej. Z tego powodu zdecydowano się wykluczyć fmGA z drugiej części eksperymentów.

W pierwszej części eksperymentów, dla funkcji o długości 30 i 50 bitów algorytmy BOA i MUPPetS były w stanie znaleźć optymalne rozwiązanie we wszystkich próbach z wyjątkiem jednej, kiedy BOA nie znalazł optymalnego rozwiązania. Należy jednak zauważyć, że nakład obliczeniowy użyty przez BOA był znacznie niższy – od dwóch, aż do siedmiu (z jednym wyjątkiem dotyczącym przypadku, kiedy BOA nie znalazł optymalnego rozwiązania). Różnica w użytym nakładzie obliczeniowym zanika dla funkcji o długości 150 i 300 bitów –

czas obliczeń był mniej więcej taki sam dla obu algorytmów. Jednak z drugiej strony, dla niektórych problemów o długości 150 bitów BOA zwraca lepsze wyniki niż MuPPetS (MuPPetS nie zawsze był w stanie znaleźć rozwiązanie optymalne we wszystkich próbach). Ponadto, dla problemów o długości 300 bitów BOA zwraca znacząco lepsze wyniki, zwłaszcza dla funkcji testowych typu „hard".

W drugiej części eksperymentów do funkcji testowych dodany został „ogon". Dla krótszych funkcji (30- i 50-bitowych) oba algorytmy były w stanie znaleźć rozwiązania optymalne we wszystkich próbach, jednak czas obliczeń stał się mniej więcej równy. Dla funkcji 150-bitowych algorytm MuPPetS był lepszy dla 11 różnych funkcji, natomiast BOA był lepszy w jednym przypadku. Nakład obliczeniowy był znacząco niższy w przypadku MuPPetS – nawet do pięciu razy. Dla problemów 300-bitowych przewaga MuPPetS nad BOA staje się bezsprzeczna. Jakość wyników zwracanych przez BOA spada do poziomu, który można określić, jako nieakceptowalny – w niektórych przypadkach średnia wartość funkcji przystosowania wynosi mniej niż wartość rozwiązania, które można znaleźć w 601 krokach przy pomocy prostego algorytmu zachłannego. W przypadku algorytmu MuPPetS jego efektywność dla funkcji 300-bitowych wzrasta po doklejeniu „ogona". Wzrost efektywności MuPPetS jest zaskakujący, ale może być łatwo wyjaśniony – ogólna wartość odsetka '1' w rozwiązaniach wzrosła w porównaniu z funkcją bez „ogona", ponieważ MuPPetS łatwo i szybko wypełnia '1' cały „ogon". Fakt wzrostu efektywności MuPPetS po doklejeniu „ogona" pokazuje jeszcze dobitniej wady algorytmu BOA, w szczególności jego brak elastyczności.

Na podstawie przeprowadzonych badań można wskazać trzy zależności:

1. Im większa jest liczba genów niezbędna do zakodowania rozwiązania, tym szybszy staje się algorytm MuPPetS w porównaniu z BOA.
2. Im więcej cech wymienionych w rozdziale 2.4.6 posiada problem, tym bardziej efektywny staje się algorytm MuPPetS w porównaniu z BOA.
3. MuPPetS jest w stanie rozwiązywać problemy na podobnym poziomie efektywności, podczas gdy BOA jest wysoce zależne od parametrów problemu (dla funkcji 300-bitowych z „ogonem", BOA aż 6 razy na 10 zwróciło wyniki gorsze niż „rozwiązanie łatwe do znalezienia").

Powyższe obserwacje pozwalają wysunąć stwierdzenie, że algorytm MuPPetS ma szansę stać się efektywnym narzędziem do rozwiązywania praktycznych problemów. Cechują go dwa różne rodzaje sposobów przechowywania informacji o powiązaniach pomiędzy genami (ang. *linkage*) i jest łatwy do zrównoleglenia (np. wszystkie operacje na poszczególnych populacjach wirusów mogą być wykonywane równolegle). W tabeli Tabela 15 przedstawiono zestawienie wad i zalet poszczególnych algorytmów.

Tabela 15. Porównanie zalet i wad algorytmów BOA, MuPPetS i fmGA

Cecha	BOA	fmGA	MuPPetS
Efektyność dla funkcji krótkich (do 50 bitów) i typu 'flat'	**Najlepszy**	Najgorszy	Średni
Efektyność dla funkcji krótkich (do 50 bitów) i	**Najlepszy**	Najgorszy	Średni

typu ‘spike’			
Efektyność dla funkcji krótkich (do 50 bitów), typu ‘flat’ z „ogonem”	**Najlepszy / Średni**	Nie dotyczy	**Najlepszy / Średni**
Efektyność dla funkcji krótkich (do 50 bitów), typu ‘spike’ z „ogonem”	**Najlepszy / Średni**	Nie dotyczy	**Najlepszy / Średni**
Efektyność dla funkcji długich (powyżej 150 bitów), typu ‘flat’	**Najlepszy**	Najgorszy	Średni
Efektyność dla funkcji długich (powyżej 150 bitów), typu ‘spike’	**Najlepszy**	Najgorszy	Średni
Efektyność dla funkcji długich (powyżej 150 bitów), typu ‘flat’ z „ogonem”	Niska / niezadowalająca	Nie dotyczy	**Wysoka**
Efektyność dla funkcji długich (powyżej 150 bitów), typu ‘spike’ z „ogonem”	Niska / niezadowalająca	Nie dotyczy	**Wysoka**
Liczba wyliczeń funkcji kosztu	**Niska**	Bardzo wysoka	Wysoka
Nakład obliczeniowy inny niż wynikający z wyliczeń funkcji koszu	Bardzo wysoka	**Niska**	Średnia
Zależność efektywności od dostrojenia parametrów	Bardzo wysoka	Średnia	**Niska**
Elastyczność algorytmów (automatyczne strojenie)	Brak	Brak	**Średnia**
Możliwości pracy równoległej	Bardzo niskie	Niskie	**Wysokie**

3.7.3.5 Efekt kierunkowania uwagi

Algorytm MuPPetS posiada również interesującą i potencjalnie pożądaną cechę. Cecha ta jest możliwa do zaobserwowania między innymi na podstawie wyników dla funkcji typu ‘flat hard’ i ‘spike hard’. Pomimo, że funkcje typu ‘spike hard’ są generalnie trudniejsze do rozwiązania niż pozostałe funkcje, to w niektórych przypadkach algorytm MuPPetS znajduje lepsze rozwiązania dla funkcji typu ‘spike’ niż dla funkcji typu ‘flat’. Wyjaśnieniem tego zaskakującego zjawiska może być fakt, że dla funkcji typu ‘spike’ algorytm MuPPetS najpierw próbuje znaleźć jak najlepsze rozwiązanie dla tych funkcji bazowych, które mają największy wpływ na wartość funkcji przystosowania, a dopiero później zajmuje się pozostałymi składowymi problemu. Taki mechanizm kierunkowania uwagi może zostać osiągnięty dzięki wzorcom genów – jeśli pewne części funkcji testowej są w danym momencie przetwarzane częściej, to wygenerują one więcej wskazujących je wzorców genów. Im więcej wzorców genów wskazujących dane miejsca w genotypie znajduje się w puli wzorców genów, tym większy wysiłek obliczeniowy zostanie skierowany przez algorytm MuPPetS na analizę tych właśnie fragmentów genotypu itd. Ten samonapędzający się mechanizm załamie się dopiero wtedy, gdy dalsze ulepszenia danego fragmentu genotypu, w danym kontekście, nie będą już możliwe, lub będą bardzo trudne do znalezienia. W takiej

sytuacji „uwaga” algorytmu zostanie skierowana (znów za pośrednictwem wzorców genów) na inne fragmenty genotypu. W tym miejscu należy zauważyć, że mechanizmy kierunkowania uwagi są dobrze znanym zabiegiem w innych gałęziach sztucznej inteligencji [8][94].

Opisany powyżej efekt w niewielkim stopniu dotyczy funkcji typu ‘normal’. Przyczyną jest fakt, że ulepszenie i znalezienie globalnego optimum dla funkcji bazowej typu ‘hard’ jest zadaniem znacznie trudniejszym niż dla funkcji typu ‘normal’. W związku z tym, gdy ulepszenie dla funkcji typu ‘hard’ zostanie znalezione, to informacja o tym ulepszeniu jest rozgłaszana wśród populacji CT (za pośrednictwem wzorców genów) ze znacznie większą siłą niż w przypadku funkcji typu ‘normal’.

3.7.3.6 MuPPetS - autoadaptacja i globalna mutacja

Jednymi z najistotniejszych cech algorytmu MuPPetS (oczywiście poza użyciem wzorców genów) jest częściowa autoadaptacja i zdolność do automatycznej globalnej mutacji. Obie te cechy zostały uzyskane dzięki mechanizmom usuwania identycznych i dodawania nowych CT. Mechanizmy te pozwalają na przetrwanie tylko tym CT, które są co najmniej częściowo różne od pozostałych, a więc wprowadzają nową wiedzę do populacji CT. Po drugie, wszystkie nowe CT są początkowo przetwarzane oddzielnie (za pomocą kolejnych uruchomień ewolucji populacji wirusów), bez korzystania ze wzorców genów i genotypu innych CT, a więc jest możliwe, że wprowadzą one jakąś zupełnie nową wiedzę do populacji CT. Należy zauważyć, że dokładnie taki efekt jest celem działania operatora globalnej mutacji, mającego za zadanie „wybić” populację z aktualnie okupowanego optimum lokalnego i nie ma tu znaczenia, że operator globalnej mutacji w swojej klasycznej wersji funkcjonuje w zupełnie innych sposób.

Na Rys. 29 pokazano jeden z typowych przebiegów algorytmu MuPPetS dla funkcji 50-bitowej. Liczba CT rośnie w kolejnych iteracjach aż do 6 i gdy tak duża liczba CT nie jest już potrzebna, liczba CT spada aż do poziomu 2-3 CT, które wystarczają algorytmowi do znalezienia optymalnego rozwiązania w tym przypadku. Dla funkcji 150-bitowej liczba CT wzrasta do poziomu około 6-9 po okresie stałego wzrostu (Rys. 30).

3.8 MuPPetS - Podsumowanie

W niniejszym rozdziale zaproponowano nowy algorytm bazujący częściowo na idei nieporządnego algorytmu genetycznego (ang. *messy GA*) oraz na idei wzorców genów. Głównym celem zaprojektowania algorytmu MuPPetS było zaproponowanie metody, która może efektywnie rozwiązywać problemy charakteryzujące się cechami wymienionymi w rozdziale 2.4.6. Wyniki przeprowadzonych eksperymentów, uzyskane na bazie konkatenacji funkcji zwodniczych (ang. *decepteive functions*), pokazały wysoką efektywność, oraz elastyczność proponowanej metody. Algorytm MuPPetS jest w stanie rozwiązywać problemy zbudowane z funkcji zwodniczych, które są szczególnie trudnymi zadaniami dla algorytmów bazujących na idei algorytmu genetycznego. Ponadto, im większa była liczba genów niezbędnych do zakodowania problemu, tym wyższa była efektywność algorytmu MuPPetS w porównaniu z pozostałymi algorytmami. Dla funkcji testowych złożonych częściowo z elementów trudnych (funkcji zwodniczych) i częściowo z łatwych („ogon”) algorytm MuPPetS jest zdecydowanie najlepszy spośród testowanych algorytmów (w niektórych przypadkach algorytm BOA zwracał rozwiązania gorsze nawet od „łatwego” rozwiązania).

Innymi, istotnymi z praktycznego punktu widzenia, cechami MuPPetS są łatwość dostrajania jego parametrów i niewielka w porównaniu z BOA i fmGA zależność jakości rozwiązania od parametrów wejściowych algorytmu. Obie te cechy zostały uzyskane dzięki częściowej zdolności algorytmu do autodostrajania. Należy zauważyć, że MuPPetS był jedynym z trzech testowanych algorytmów, który zawsze używał tego samego zestawu parametrów bez względu na rozwiązywany problem (funkcje 300-bitowe, 50-bitowe, funkcje typu 'spike', 'flat', z „ogonem", „bez ogona", etc.). Z drugiej strony, algorytm BOA jest bardzo czuły na ustawienia swoich parametrów [13]. Na powyższych przesłankach można zatem sformułować wniosek, że wykorzystanie wzorców genów w sposób zaprezentowany w niniejszej pracy, wraz z koewoluującymi populacjami wirusów, może dać lepsze wyniki niż inne algorytmy.

Wyniki zaprezentowane w niniejszym rozdziale wskazują, że algorytm MuPPetS może zostać użyty dla dowolnego problemu, który można rozwiązać innymi algorytmami opartymi na idei algorytmów genetycznych. Jeśli rozmiar rozwiązywanego problemu staje się krótszy, a czas niezbędny do pojedynczego obliczenia wartości funkcji przystosowania rośnie, BOA wydaje się być lepszym narzędziem, choć również nieidealnym i obciążonym znaczącymi wadami (jak na przykład silna zależność jakości wyników od parametrów wejściowych algorytmu i rodzaju rozwiązywanego problemu). Dlatego do rozwiązywania trudnych problemów obliczeniowych, zwłaszcza wymagających długiego kodowania, algorytm MuPPetS wydaje się być dobrym wyborem.

Na obecnym etapie badań trudno jest wskazać jakiekolwiek istotne wady algorytmu MuPPetS, jeśli nie liczyć jego stosunkowo dużej skali skomplikowania w porównaniu ze standardowym algorytmem genetycznym. Jednak, zdaniem autora, istota operacji wykonywanych przez MuPPetS powinna być łatwa do zrozumienia. Wyniki pokazują również, że MuPPetS może być dobrą odpowiedzią na spadek efektywności algorytmu genetycznego (i szerzej - ewolucyjnego) spowodowany wzrostem długości kodowania problemu i trudnością samego problemu.

W następnym rozdziale zostanie zaprezentowany algorytm MuPPetS-FuN, który jest próbą wykorzystania idei zaprezentowanych w niniejszym rozdziale do rozwiązania trudnego problemu praktycznego – projektowania przepływu w szkieletowej sieci komputerowej.
Innymi kierunkami badań nad algorytmem MuPPetS powinny być również:

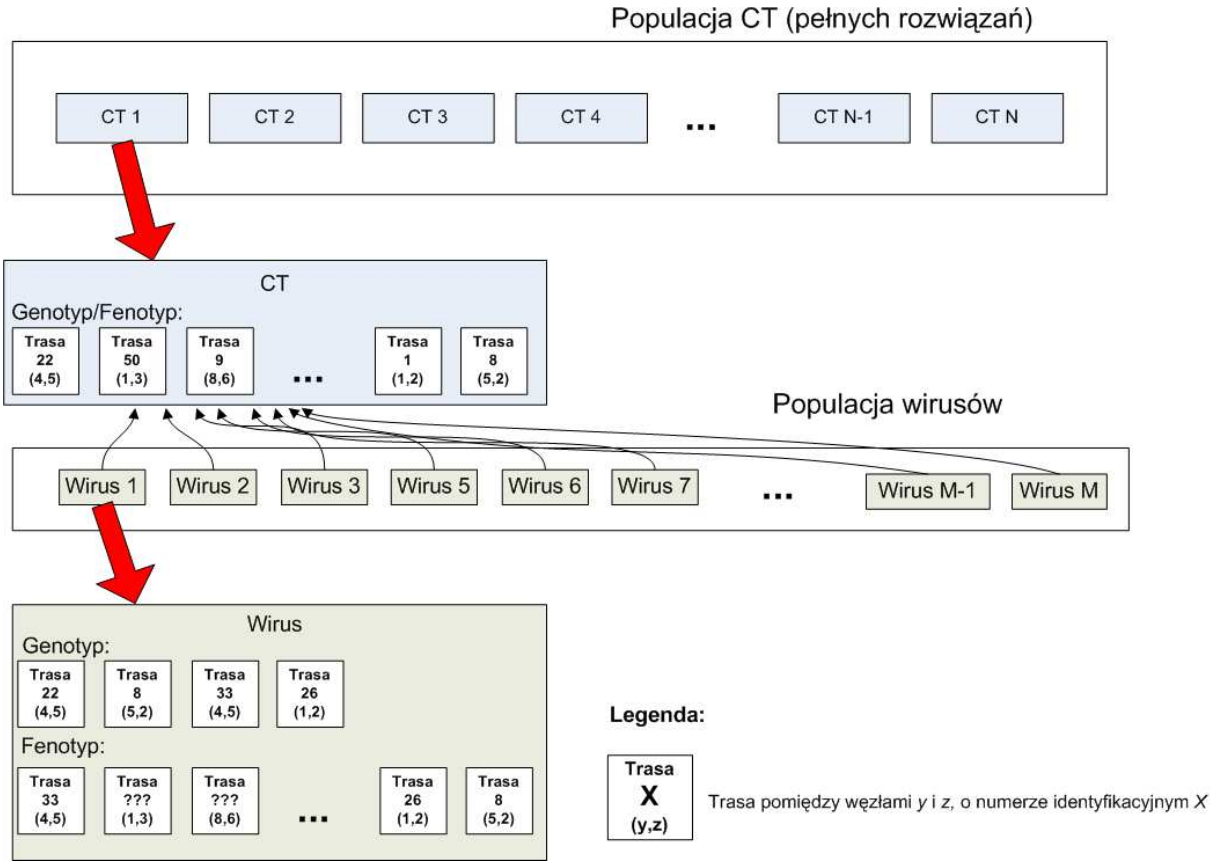
- Zaproponowanie bardziej zaawansowanych i „świadomych" metod selekcji wzorców genów (zarówno do dalszego przetwarzania, jak i usuwania z puli wzorców genów)
- Dalsze badania nad zaobserwowanym mechanizmem kierunkowania uwagi w celu jego użycia do zwiększenia efektywności algorytmu
- Stworzenie bardziej elastycznych metod dodawania i usuwania CT
- Rozwinięcie zdolności autoadaptacji algorytmu w taki sposób, aby prawdopodobieństwa użycia operatorów cięcia, sklejania, oraz mutacji były raczej ewoluowane przez sam algorytm niż ustawiane na stałe przez użytkownika

4. MuPPetS-FuN – propozycja nowego algorytmu do rozwiązywania problemu projektowania przepływu w szkieletowej sieci komputerowej bez rozgałęzień, opartego o schemat działania algorytmu MuPPetS oraz algorytm HEFAN

W niniejszym rozdziale przedstawiono algorytm MuPPetS-FuN (**Mu**lti **P**opulation **P**attern **S**earching Algorithm for **F**low Assigment in **N**on-bifurcated commodity flow), który jest propozycją nowego algorytmu rozwiązującego problem projektowania przepływu w szkieletowej sieci komputerowej zorientowanej połączeniowo. Algorytm MuPPetS-FuN używa szablonu MuPPetS przedstawionego w rozdziale 3, ponieważ ten model algorytmu opartego na idei algorytmu genetycznego pozwala oczekiwać lepszych wyników niż w przypadku klasycznego algorytmu genetycznego, nieporządnego algorytmu genetycznego (ang. *messy genetic algorithm*), czy algorytmu BOA. W algorytmie MuPPetS-FuN zostały również zastosowane elementy algorytmu HEFAN, takie jak kodowanie, użycie bazy tras, oraz użycie mechanizmów krzyżowania i oceny tras.

4.1 Budowa algorytmu MuPPetS-FuN

Algorytm MuPPetS-FuN korzysta w całości z szablonu pracy zaproponowanego przez algorytm MuPPetS. Szablon ten musiał być jednak uzupełniony o elementy, które są niezbędne, lub korzystne dla rozwiązywania problemu projektowania przepływu w szkieletowej sieci komputerowej. Te elementy zostały zaczerpnięte z algorytmu HEFAN. Struktura danych algorytmu MuPPetS-FuN została przedstawiona na Rys. 31.



Rys. 31 Struktura danych algorytmu MuPPetS-FuN

Jak widać na Rys. 31 w algorytmie MuPPetS-FuN zmianie uległ sposób kodowania genotypu – pojedynczy gen nie jest reprezentowany binarnie, ale przez identyfikator trasy. Zbiór dostępnych tras jest umieszczony w bazie tras, która została zaczerpnięta i funkcjonuje identycznie, jak w algorytmie HEFAN (patrz: punkt. 2.4.3). Fazy i przebieg algorytmu MuPPetS-FuN jest identyczny jak w przypadku algorytmu MuPPetS, (patrz: punkt. 3.5.3) z wyjątkiem różnic w sposobie dodawania i usuwania CT opisanych w następnym podrozdziale, oraz zmodyfikowanym przebiegiem fazy 1. Różnica w przebiegu fazy 1 polega na tym, że dla nowododanego CT populacja wirusów jest przetwarzana bez udziału wzorców genów nie dłużej niż określoną liczbę pokoleń (w algorytmie MuPPetS trwało to tak długo, aż ewolucja populacji wirusów nie przestała poprawiać funkcji przystosowania nowododanego CT). Operacje krzyżowania i mutacji na poziomie niskim z algorytmu HEFAN (patrz: rozdz. 2.4.5) wykonywane są w algorytmie MuPPetS-FuN w trakcie ewolucji populacji wirusów tak, jakby były to kolejne rodzaje operatorów mutacji (algorytm MuPPetS używa trzech operatorów mutacji: tradycyjnego, zmieniającego wartość genu, mutacji dodającej losowy gen do genotypu i usuwającej losowy gen z genotypu). Jednakże wirus, dla którego ma być przeprowadzona operacja krzyżowania, lub mutacji na poziomie niskim, jest przed tą operacją kopiowany, a operacja krzyżowania/mutacji na poziomie niskim odbywa się na kopii wirusa. Po jej wykonaniu kopia wirusa, na której przeprowadzono operację jest dodawana do populacji wirusów.

Wartość funkcji przystosowania dla każdego osobnika jest liczona tak jak w algorytmie HEFAN (patrz: punkt 2.4.4).

4.2 Wstępne testy algorytmu MuPPetS-FuN

Wszystkie testy algorytmu MuPPetS-FuN przeprowadzono przy użyciu zestawu sieci i wymagań dotyczących przepływu opisanych w punkcie 1.3. Wszystkie testy zostały przeprowadzone na komputerze AMD Athlon 64 X2 Dual Core Processor 3800+, 2 GB RAM, z systemem operacyjnym Windows XP SP2.

Procedurę dostrajania parametrów algorytmu MuPPetS-FuN, przeprowadzono przy użyciu sześciu, losowo wybranych sieci i wymagań dla przepływu. Wymogiem w procedurze losowania sieci było jednak, aby wylosowane eksperymenty różniły się pod względem topologii sieci i rodzaju eksperymentu (patrz: punkt 1.3). Dostrajanie przeprowadzone zostało tak, jak w przypadku strojenia algorytmu MuPPetS – parametry dobrano dla pojedynczego CT, dla którego wywoływano cyklicznie ewolucję populacji wirusów przez określony okres czas. W związku z powyższym pozyskiwanie i użycie wzorców genów, nie miało żadnego wpływu na przebieg i wynik procedury dostrajania algorytmu. Ostatecznie dla algorytmu MuPPetS-FuN wybrano konfigurację przedstawioną w tabeli Tabela 16.

Tabela 16. Parametry wywołania algorytmu MuPPetS-FuN

Nazwa parametru	Wartość
Liczba cykli klonera tras	2
Liczba najkrótszych tras pomiędzy każdą parą węzłów	4
Współczynnik kary za przekroczenie przepustowości łuku sieci	10
Rozmiar puli wzorców genów	500
Minimalna dopuszczalna długość wzorca	3
Liczba generacji wirusów	30
Rozmiar populacji wirusów	30
Współczynnik redukcji liczby wirusów w kolejnych populacjach	0,97
Prawdopodobieństwo "cięcia"	0,09
Prawdopodobieństwo "sklejania"	0,15
Prawdopodobieństwo mutacji	0,1
Prawdopodobieństwo usunięcia genu	0,1
Prawdopodobieństwo dodania genu	0,1
Prawdopodobieństwo krzyżowania na poziomie 'niskim'	0,5
Prawdopodobieństwo mutacji na poziomie 'niskim'	0,4
Maksymalna liczba wywołań fazy przetwarzania populajci wirusów dla nowododanego CT	10

Jakość wyników dla poszczególnych grup eksperymentów będzie porównywana na podstawie współczynnika jakości wyniku [78], obliczanego na podstawie wzoru:

$$RQ(R_c) = \begin{cases} 1 - \frac{R_c - R_B}{R_c} & dla \quad R_c > 0 \\ 0 & dla \quad R_c = 0 \end{cases},$$

gdzie
RQ to współczynnik jakości wyniku, R_c to ocena rozwiązania zwróconego przez dany algorytm, a R_B to ocena najlepszego rozwiązania spośród zwróconych przez wszystkie porównywane algorytmy

(46)

Należy zauważyć, że wartość współczynnika *RQ* dla najlepszego(najlepszych) rozwiązań to 0. Jednak wartość współczynnika *RQ* dla pozostałych rozwiązań zależy od wartości najlepszych. Np. jeśli jeden ze współzawodniczących algorytmów znajdzie takie rozwiązanie, że wartość funkcji LFL będzie wynosić 0 (najniższa możliwa wartość dla funkcji LFL), to ocena wszystkich pozostałych rozwiązań będzie zawsze równa 1. Jeśli jednak najlepszy znaleziony wynik będzie większy od 0, to wartość współczynników RQ dla pozostałych rozwiązań będzie zawsze mniejsza od 1.

Przeprowadzono dwa eksperymenty. W pierwszym eksperymencie porównano jakość wyników zwracanych przez algorytmy MuPPetS-FuN i HEFAN 2.2 przy ograniczeniu czasu obliczeń do 1200 sekund, a w drugim ograniczenie czasowe wynosiło 2400 sekund. W niniejszym rozdziale przedstawione zostaną różnego rodzaju statystyki i zestawienia jednak pełne wyniki dla wszystkich przeprowadzonych testów można znaleźć w Załącznik A - wyniki, w tabelach Tabela 31 i Tabela 32. Dla 180 różnych konfiguracji sieci i wymagań względem przepływu, średnie wartości współczynnika RQ dla obu algorytmów zaprezentowane są w tabeli Tabela 17. W tabeli Tabela 18 przedstawiono ile razy dany algorytm zwrócił lepszy wynik.

Tabela 17. Średnie wartości współczynnika RQ dla wyników zwróconych przez algorytmy MuPPetS-FuN i HEFAN 2.2 w zależności od czasu obliczeń

Czas obliczeń [s] / Algorytm	MuPPetS-FuN	HEFAN 2.2
1200	0,04	0,01
2400	0,13	0,15

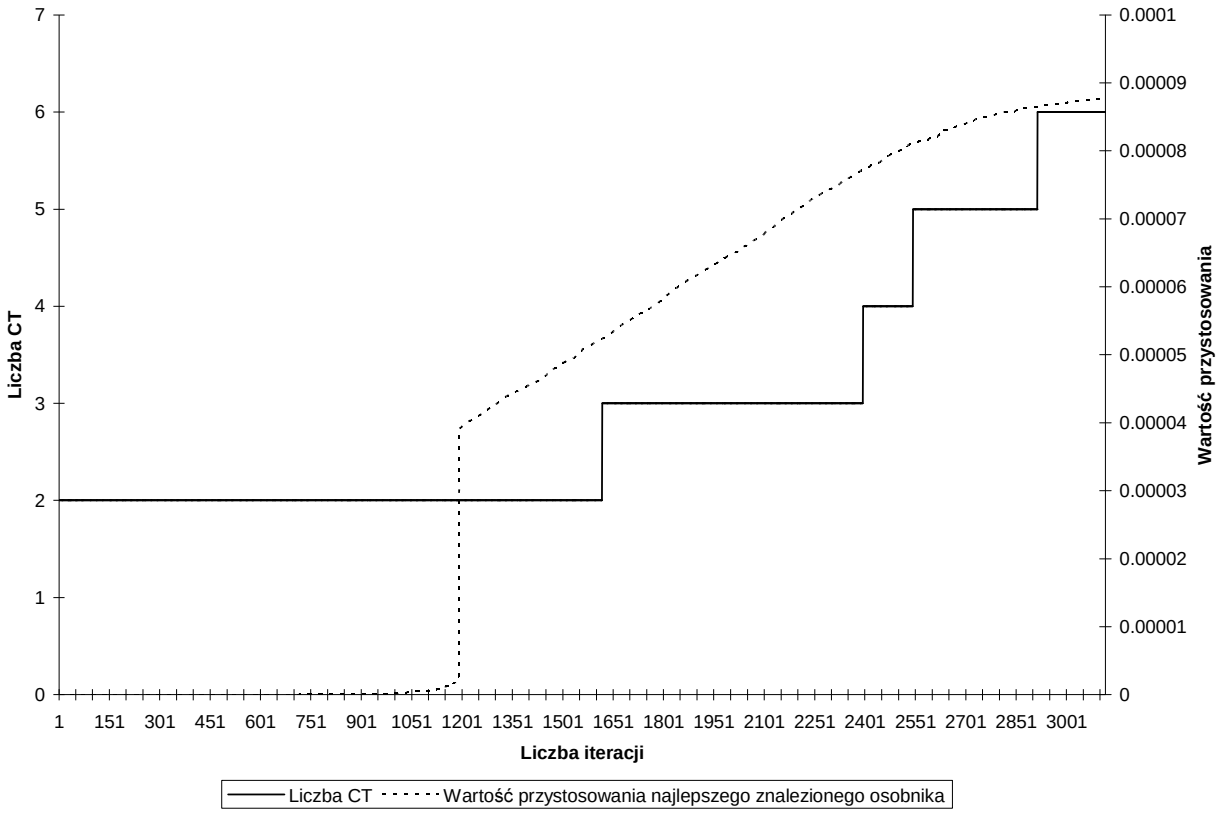
Tabela 18. Porównanie liczby zwycięstw algorytmów MuPPetS-FuN i HEFAN 2.2 w zależności od czasu obliczeń

Czas obliczeń [s] / Algorytm	MuPPetS-FuN	HEFAN 2.2	Remis
1200	61	48	71
2400	93	53	34

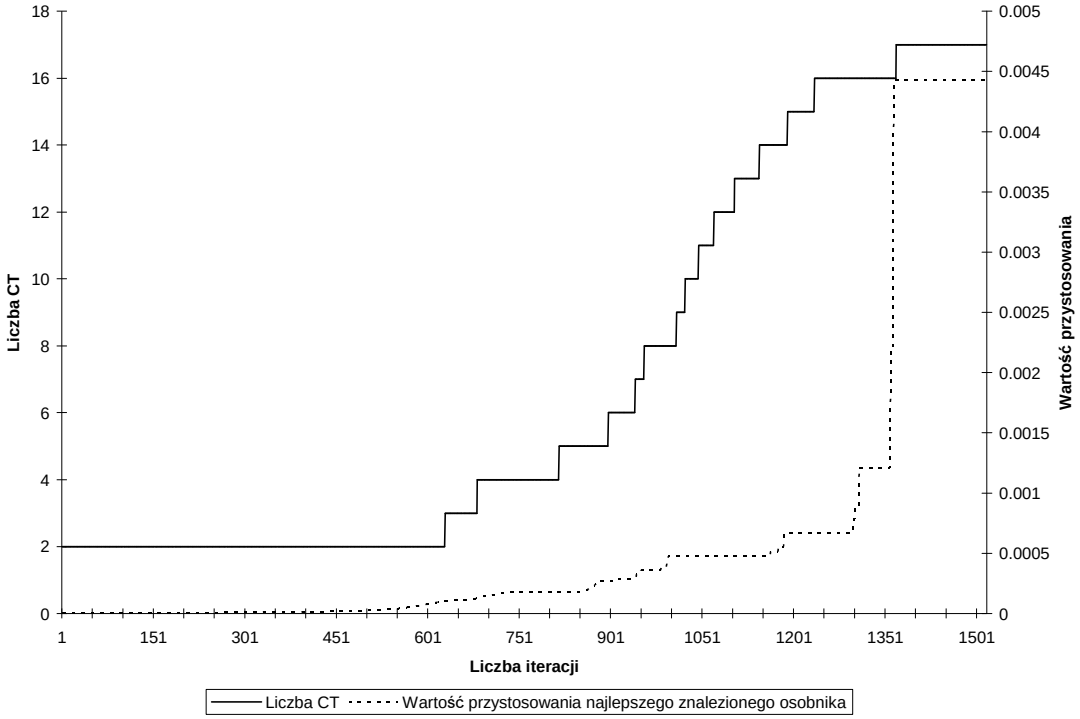
Należy zauważyć, że średnia jakość wyników, mierzona za pomocą współczynnika RQ była przy ograniczeniu czasu obliczeń do 1200 sekund niemal identyczna dla obu algorytmów. Niewielką przewagę osiągnął jednak algorytm HEFAN 2.2. Co jednak zaskakujące, algorytm MuPPetS-FuN zwrócił lepszy wynik aż w 61 przypadkach, podczas gdy HEFAN 2.2 był lepszy zaledwie 48 razy. Takie wyniki mogą świadczyć o tym, że algorytm MuPPetS-FuN

jest w stanie opuścić minimum lokalne, w którym utyka algorytm HEFAN 2.2. Jednocześnie jednak algorytm MuPPetS-FuN w wielu przypadkach nie ma dość czasu na wygenerowanie dobrego jakościowo wyniku, więc jeśli okazuje się słabszy, to różnica w jakości wyników pomiędzy oboma algorytmami jest znaczna na niekorzyść MuPPetS-FuN. Aby sprawdzić powyższą hipotezę przeprowadzono kolejny eksperyment, zwiększając czas obliczeń do 2400 sekund. W tym przypadku wartość współczynnika RQ dla obu algorytmów znacząco wzrosła, co oznacza, że różnice pomiędzy wartościami wyników były znacznie większe niż w pierwszym przypadku. Tym razem to algorytm MuPPetS-FuN okazał się nieznacznie lepszy pod względem średniej wartości współczynnika RQ. Wzrosła także przewaga w liczbie przypadków, w których algorytm MuPPetS-FuN okazał się lepszy od konkurenta: 93-53 na korzyść MuPPetS-FuN przy 34 remisach.

Pierwsze przeprowadzone testy wykazały, że potencjał algorytmu MuPPetS-FuN ukazuje się tym pełniej, im dłuższy jest czas przeprowadzanych obliczeń. Na tej podstawie można sformułować wniosek, że przewaga algorytmu MuPPetS-FuN nad algorytmem HEFAN 2.2 wynika z faktu, że lepiej radzi on sobie z opuszczaniem obszarów optimów lokalnych. Uzyskanych wyników nie można uznać jednak za satysfakcjonujące, ponieważ różnica w jakości zwracanych wyników, mierzona współczynnikiem RQ, jest niewielka. Dla określenia możliwych przyczyn tego faktu przeprowadzono analizę przebiegu algorytmu MuPPetS-FuN.



Rys. 32. Przebieg algorytmu MuPPetS-FuN dla przypadku, w którym algorytm MuPPetS-FuN okazał się lepszy od algorytmu HEFAN 2.2



Rys. 33. Przebieg algorytmu MuPPetS-FuN dla przypadku, w którym algorytm MuPPetS-FuN okazał się gorszy od algorytmu HEFAN 2.2

Na Rys. 32 i Rys. 33 przedstawiono przykładowe przebiegi algorytmu MuPPetS-FuN dla przypadku, w którym okazał on się lepszy od algorytmu HEFAN 2.2 (Rys. 32) i gorszy od algorytmu HEFAN 2.2 (Rys. 33). Na podstawie tych przebiegów można zauważyć, że algorytm MuPPetS-FuN wypada słabiej, gdy liczba CT w trakcie przebiegu algorytmu jest większa. Można więc wnioskować, że algorytm MuPPetS-FuN pracuje mniej efektywnie, gdy liczba używanych CT jest na tyle duża, że algorytm zbyt dużo czasu poświęca na sekwencyjne wywoływanie ewolucji populacji wirusów dla każdego CT. W celu potwierdzenia lub zaprzeczenia tej tezie przeprowadzono eksperyment, w którym algorytm MuPPetS-FuN używał wyłącznie jednego CT w całym swoim przebiegu. Tę wersję algorytmu nazwano MuPPetS-FuN Single. Porównanie efektywności algorytmu MuPPetS-FuN Single z algorytmem MuPPetS FuN zostało przedstawione w tabelach Tabela 19 i Tabela 20.

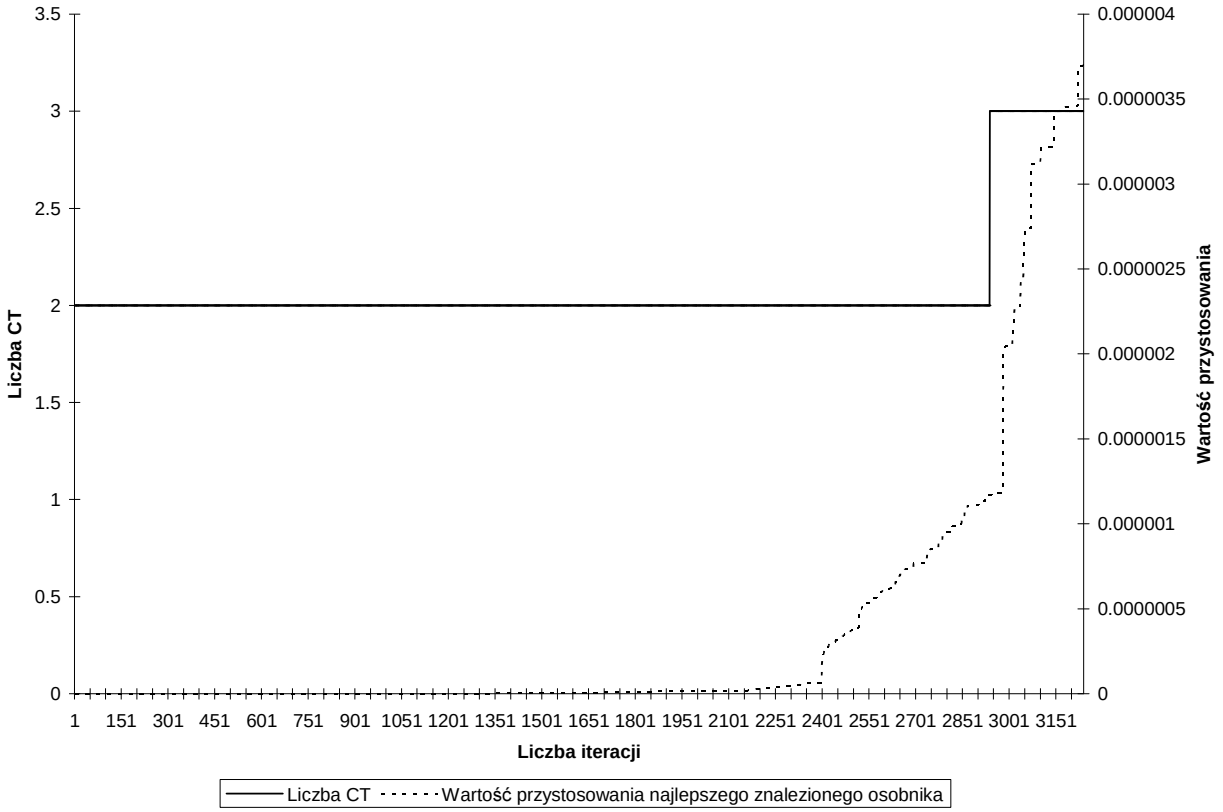
Tabela 19. Średnie wartości współczynnika RQ dla wyników zwróconych przez algorytmy MuPPetS-FuN i MuPPetS-FuN Single dla czasu obliczeń 2400 sekund

MuPPetS-FuN	MuPPetS-FuN Single
0,23711583	0,0027841

Tabela 20. Porównanie liczby zwycięstw algorytmów MuPPetS-FuN i MuPPetS-FuN Single dla czasu obliczeń 2400 sekund

MuPPetS-FuN	MuPPetS-FuN Single	Remis
14	132	34

Wyniki zaprezentowane w tabelach Tabela 19 i Tabela 20 pokazują niekwestionowaną przewagę algorytmu MuPPetS-FuN Single nad algorytmem MuPPetS-FuN. Tym bardziej zaskakujący jednak jest fakt, że algorytm MuPPetS-FuN zdołał 14 razy zwrócić wynik lepszy od konkurenta. Jeden z przebiegów algorytmu MuPPetS-FuN, dla przypadku, w którym okazał on się lepszy od algorytmu MuPPetS-FuN Single został przedstawiony na Rys. 34. Znamienne jest, że w przedstawionym przebiegu algorytm MuPPetS-FuN używał stosunkowo niewielkiej liczby CT. Można więc oczekiwać, że dla z góry ustalonej, niewielkiej liczby CT algorytm MuPPetS-FuN zwróci lepsze wyniki niż algorytm MuPPetS-FuN Single. W związku z tym wykonano eksperymenty dla algorytmu MuPPetS-FuN z ustaloną na stałe liczbą CT równą 2. Tę wersję algorytmu nazwano MuPPetS-FuN Double. Porównanie efektywności algorytmu MuPPetS-FuN Single z algorytmem MuPPetS-FuN Double zostało przedstawione w tabelach Tabela 21 i Tabela 22.



Rys. 34. Przebieg algorytmu MuPPetS-FuN dla przypadku, w którym algorytm MuPPetS-FuN okazał się lepszy od algorytmu MuPPetS-FuN Single

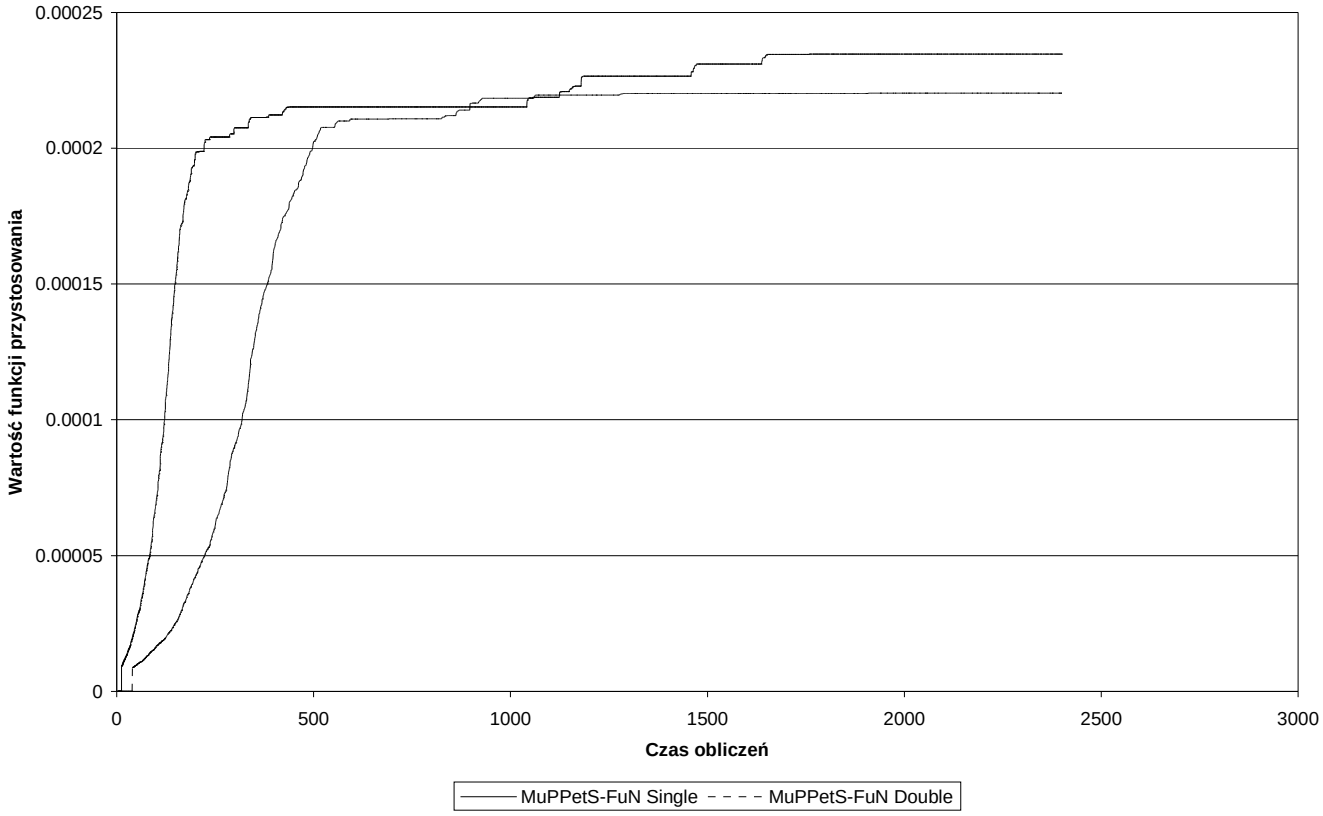
Tabela 21. Średnie wartości współczynnika RQ dla wyników zwróconych przez algorytmy MuPPetS-FuN Single i MuPPetS-FuN Double dla czasu obliczeń 2400 sekund

MuPPetS-FuN Single	MuPPetS-FuN Double
0,03084319	0,12372732

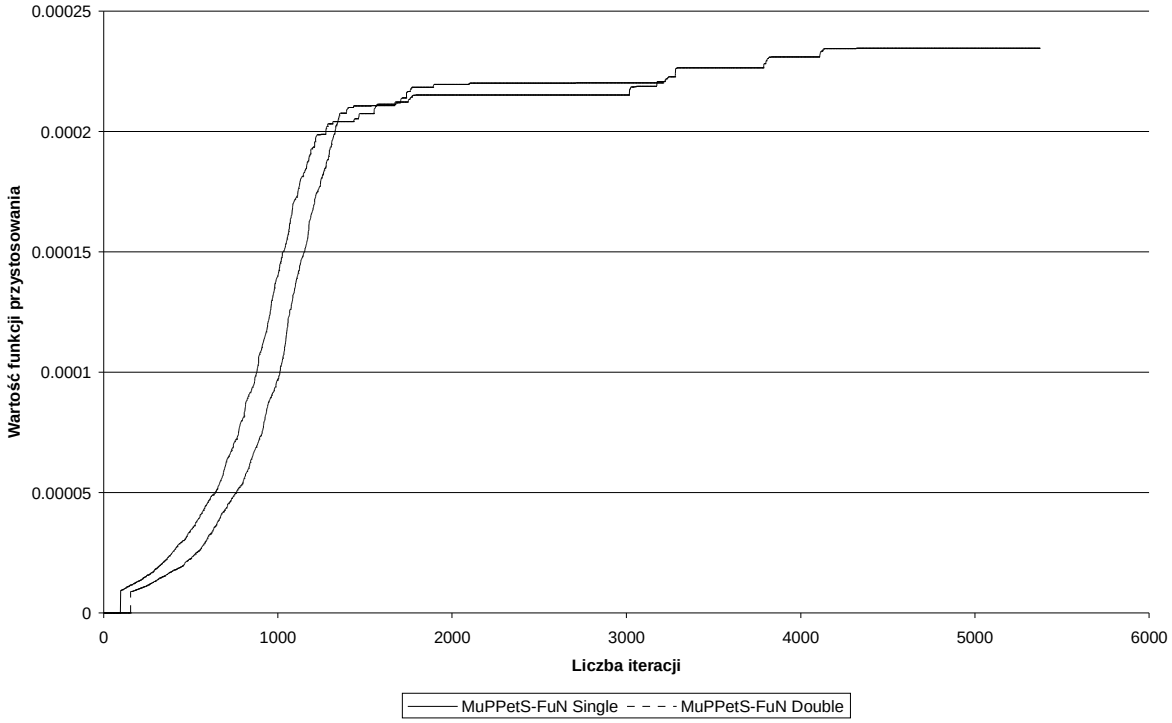
Tabela 22. Porównanie liczby zwycięstw algorytmów MuPPetS-FuN Single i MuPPetS-FuN Double MuPPetS-FuN Single i MuPPetS-FuN Double dla czasu obliczeń 2400 sekund

MuPPetS-FuN Single	MuPPetS-FuN Double	Remis
103	32	45

Porównanie jakości wyników zwracanych przez algorytmy MuPPetS-FuN Single i MuPPetS-FuN Double wyraźnie wskazuje, że algorytm MuPPetS-FuN Single jest znacznie bardziej efektywny. Należy jednak zauważyć, że algorytm MuPPetS-FuN Double zwrócił wyniki o znacznie wyższej jakości niż MuPPetS-FuN w swojej podstawowej wersji.



Rys. 35. Porównanie przebiegów algorytmów MuPPetS-FuN Single i MuPPetS-FuN Double w zależności od czasu



Rys. 36. Porównanie przebiegów algorytmów MuPPetS-FuN Single i MuPPetS-FuN Double w zależności od liczby iteracji dla czasu 2400 sekund

Na Rys. 35 i Rys. 36 przedstawiono porównanie przebiegów algorytmów MuPPetS-FuN Single i MuPPetS-FuN Double w zależności od czasu i liczby iteracji. Na ich przykładzie jest widoczne, że MuPPetS-FuN Double przegrywa w konkurencji z MuPPetS-FuN Single z powodu znacznie mniejszej liczby iteracji. Na podstawie powyższych wniosków, zaproponowano kolejną wersję algorytmu MuPPetS-FuN, o nazwie MuPPetS-FuN Active.

4.3 MuPPetS-FuN Active

Algorytm MuPPetS-FuN Active używa tylko jednego CT, chyba że poprawienie wartości jego przystosowania za pomocą ewolucji populacji wirusów okazało się niemożliwe. W takiej sytuacji liczba CT w populacji CT jest zwiększana o 1. Jeśli wartość przystosowania najlepszego CT w populacji wzrośnie, to usuwane są wszystkie CT poza najlepszym. Jeśli wartość przystosowania wszystkich CT w populacji pozostanie niezmieniona dodawany jest nowy CT.

Porównanie efektywności algorytmu MuPPetS-FuN Single z algorytmem MuPPetS-FuN Active zostało przedstawione w tabelach Tabela 23, Tabela 24.

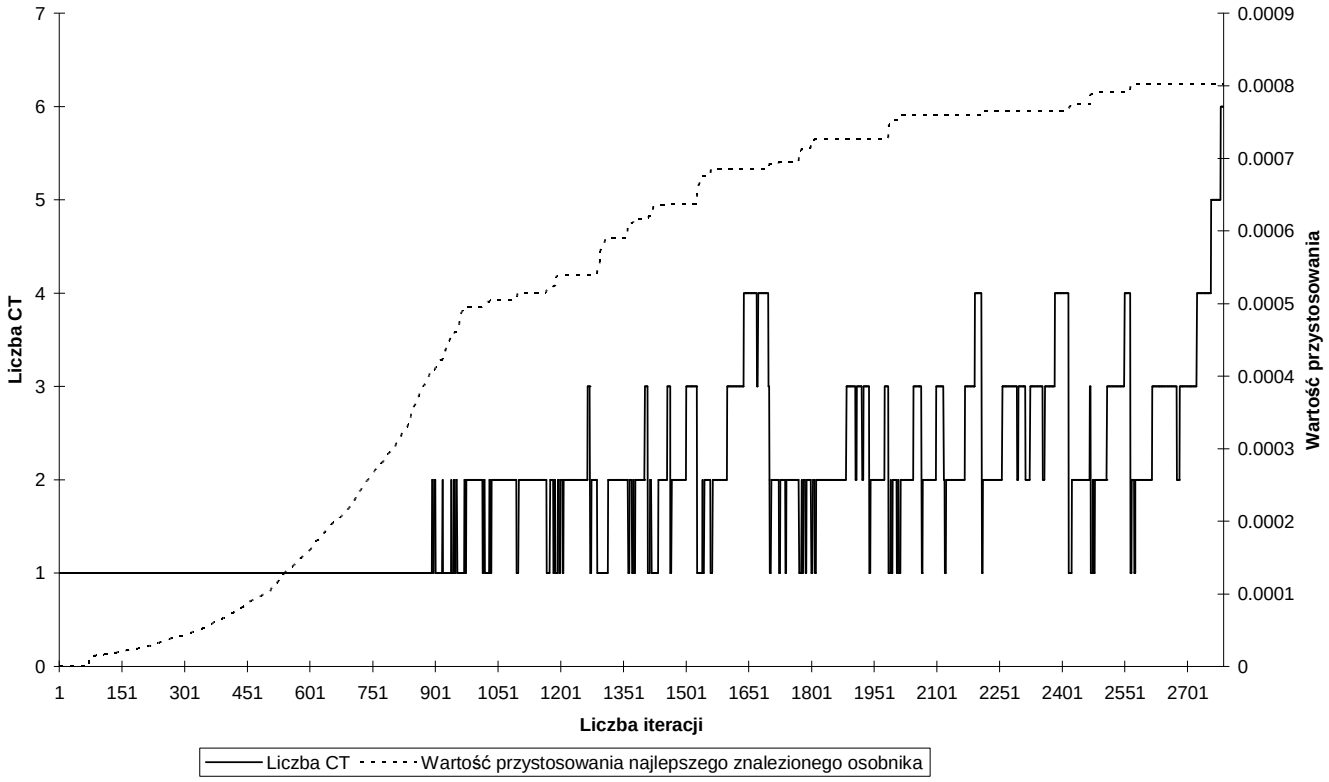
Tabela 23. Średnie wartości współczynnika RQ dla wyników zwróconych przez algorytmy MuPPetS-FuN Single i MuPPetS-FuN Active dla czasu obliczeń 2400 sekund

MuPPetS-FuN Single	MuPPetS-FuN Active
0,05832049	0,04729452

Tabela 24. Porównanie liczby zwycięstw algorytmów MuPPetS-FuN Single i MuPPetS-FuN Double MuPPetS-FuN Single i MuPPetS-FuN Active dla czasu obliczeń 2400 sekund

MuPPetS-FuN Single	MuPPetS-FuN Active	Remis
76	58	46

W związku z faktem, że wartość współczynnika RQ, dla algorytmu MuPPetS-FuN Active jest niższa niż dla algorytmu MuPPetS-FuN Single, został on uznany za lepszy i wybrany do porównań z innymi algorytmami. Uwagę zwraca jednak fakt, że liczba zwycięstw ma większą wartość dla algorytmu MuPPetS-FuN Single.



Rys. 37. Przykład przebiegu algorytmu MuPPetS-FuN Active

Na Rys. 37 został przedstawiony przykładowy przebieg algorytmu MuPPetS-FuN Active. Jak można zauważyć, w początkowej fazie szybkiego wzrostu wartości funkcji przystosowania, przez około 900 iteracji, algorytm używa wyłącznie jednego CT. Następnie algorytm wchodzi w fazę szybkich oscylacji liczby CT pomiędzy wartościami 1 i 2 (od 900 do 1000 iteracji), przy czym szybki wzrost wartości funkcji przystosowania zostaje utrzymany. W kolejnej fazie (od 1000 do około 1200 iteracji) przebiegu algorytmu liczba CT jest raczej stabilna i wynosi 2, wzrost wartości funkcji przystosowania wyraźnie wyhamowuje, jednak wciąż zachowuje stosunkowo płynny charakter. W ostatniej fazie liczba CT znacząco rośnie – do 3-4, a nawet 6 do CT w ostatnich iteracjach. Również wykres zmian wartości funkcji przystosowania najlepszego znalezionego osobnika zmienia swój charakter na wyraźnie schodkowy – po okresach przestoju następuje nagły wzrost i kolejny przestój.

Na podstawie analizy przebiegu algorytmu MuPPetS-FuN Active można dojść do wniosku, ze zastosowana w nim strategia dodawania i usuwania CT w trakcie przebiegu

algorytmu jest najlepsza ze wszystkich zaprezentowanych w niniejszej pracy. Z jednej strony nie powoduje niepotrzebnego zużycia mocy obliczeniowej w początkowej fazie przebiegu algorytmu, kiedy ulepszanie najlepszego znalezionego rozwiązania wydaje się być stosunkowo łatwe. Z drugiej natomiast, wprowadzając do populacji nowe CT potrafi doprowadzić do przełomu i opuszczenia zajmowanego obszaru optimum lokalnego.

5. Porównanie efektywności algorytmu MuPPetS-FuN Active z innymi algorytmami

W niniejszym rozdziale przedstawiono porównanie efektywności algorytmu MuPPetS-FuN Active oraz algorytmów HEFAN 2.2 i LRH. Część z wyników zaprezentowanych wyników została już opublikowana w pracy [78]. Wszystkie eksperymenty dla algorytmów MuPPetS-FuN Active, oraz HEFAN 2.2 zostały przeprowadzone na komputerze AMD Athlon 64 X2 Dual Core Processor 3800+, 2 GB RAM, z systemem operacyjnym Windows XP SP2. Wyniki zostały pogrupowane w zależności od rodzaju sieci i rodzaju eksperymentu (patrz rozdział: 1.3). Eksperymenty zostały przeprowadzone dla 180 różnych konfiguracji sieci i zapotrzebowań na przepływ. Pełne wyniki dla wszystkich algorytmów są zmieszczone w Załącznik A - wyniki, w tabeli Tabela 32.

5.1 Porównanie efektywności algorytmu MuPPetS-FuN Active i algorytmu LRH

W tabeli Tabela 25 przedstawiono porównanie wyników zwróconych przez algorytmy MuPPetS-FuN Active i LRH.

Tabela 25. Porównanie wyników zwróconych przez algorytmy LRH i MuPPetS-FuN Active

	Współczynnik RQ		Porównanie liczby zwycięstw		
Rodzaj eksperymentu	LRH	MuPPetS-FuN Active	LRH	remis	MuPPetS-FuN Active
Tylko sieci typu NET 104	0.04892992	**0.04540753**	10	5	**15**
Tylko sieci typu NET 114	0.15500783	**0.11101282**	8	7	**15**
Tylko sieci typu NET 128	0.1254151	**0.10749656**	10	9	**11**
Tylko sieci typu NET 144	0.17742332	**0.03123931**	6	7	**17**
Tylko sieci typu NET 162	**0.06856755**	0.1078275	**13**	9	8
Tylko sieci typu NET 120 (sieci typu "grid")	**0**	0.33176013	**25**	5	0
Eksperyment A	0.13069451	**0.08871113**	10	20	**30**
Eksperyment A bez sieci NET 120	0.15683341	**0.02122313**	2	18	**30**
Eksperyment B	**0.08709086**	0.11015359	20	**21**	19
Eksperyment B bez sieci NET 120	0.10450904	**0.0713619**	13	18	**19**
Eksperyment C	**0.06988649**	0.1685072	**42**	1	17
Eksperyment C bez sieci NET 120	**0.08386378**	0.14920519	**32**	1	17

Algorytm MuPPetS-FuN Active był lepszy dla wszystkich rodzajów sieci za wyjątkiem sieci typu NET 162 i sieci typu „grid”. W przypadku sieci typu NET 162 przewaga algorytmu LRH nad algorytmem MuPPetS-FuN Active jest wyraźna, jednak nie różni się znacząco od przewagi, którą dla pozostałych grup sieci osiąga MuPPetS-FuN Active. Jednak dla sieci typu „grid” przewagę algorytmu LRH należy uznać za miażdżącą – w żadnym z 30 przypadków algorytm MuPPetS-FuN Active nie był lepszy niż LRH, a remis wystąpił zaledwie 5 razy. Taka znacząca różnica dla tej, szczególnej, grupy sieci jest trudna do wytłumaczenia.

W tabeli Tabela 25 zaprezentowano również porównanie efektywności poszczególnych algorytmów z podziałem na poszczególne rodzaje eksperymentów (A, B i C – patrz punkt 1.3). W związku z zaskakującą różnicą w efektywności obu algorytmów dla sieci typu „grid”, statystyki zostały przedstawione dla wszystkich przypadków wchodzących w skład danego eksperymentu i dla wszystkich przypadków z wyłączeniem sieci typu „grid”. W przypadku eksperymentów typ A przewagę algorytmu MuPPetS-FuN Active nad LRH należy ocenić jako bardzo wyraźną. W przypadku eksperymentu B żaden algorytm nie uzyskał wyraźnej przewagi. Wreszcie w eksperymencie C algorytm LRH ma nad algorytmem MuPPetS-FuN Active przewagę porównywalną do tej, którą osiągnął algorytm MuPPetS-FuN Active w eksperymencie A. Sytuacja znacząco zmienia się, kiedy z wyników eksperymentów wyeliminowana zostanie grupa sieci typu „grid”. Dla eksperymentu A algorytm MuPPetS-FuN Active jest lepszy aż w 60% przypadków, remis jest osiągany w 36% przypadków, a sytuacja w której lepszy jest algorytm LRH to zaledwie 4% przypadków. Również dla eksperymentów z grupy B przewaga algorytmu MuPPetS-FuN Active staje się wyraźna. Wyjątkiem pozostaje grupa eksperymentów typu C – algorytm LRH wciąż pozostaje lepszy, choć jego przewaga nie jest już tak wyraźna, a algorytm MuPPetS-FuN Active jest lepszy aż w 34% przypadków.

Powyższe wyniki prowadzą do następujących obserwacji:

- Algorytm MuPPetS-FuN Active jest efektywniejszy od algorytmu LRH dla sieci o nieregularnej siatce
- Algorytm LRH jest zdecydowanie efektywniejszy od algorytmu MuPPetS-FuN Active dla sieci typu „grid”
- Algorytm MuPPetS-FuN Active jest efektywniejszy od LRH w eksperymentach, które wymagają zestawienia mniejszej liczby połączeń (eksperymenty typu A)
- W eksperymentach, które wymagają zestawienia większej liczby połączeń (eksperymenty typu B i C) żaden z algorytmów nie osiąga szczególnej przewagi

5.2 Porównanie efektywności algorytmu MuPPetS-FuN Active i algorytmu HEFAN 2.2

W tabeli Tabela 26 przedstawiono porównanie wyników zwróconych przez algorytmy MuPPetS-FuN Active i HEFAN 2.2.

Tabela 26. Porównanie wyników zwróconych przez algorytmy HEFAN 2.2 i MuPPetS-FuN Active

	Współczynnik RQ		Porównanie liczby zwycięstw		
Rodzaj eksperymentu	HEFAN 2.2	MuPPetS-FuN Active	HEFAN 2.2	remis	MuPPetS-FuN Active
Tylko sieci typu NET 104	0.11494016	**0**	0	0	**30**
Tylko sieci typu NET 114	0.16179091	**0.03498377**	3	11	**16**
Tylko sieci typu NET 128	0.2898007	**0.07327293**	3	9	**18**
Tylko sieci typu NET 144	0.14381444	**0.03375545**	6	11	**13**
Tylko sieci typu NET 162	0.1848758	**0.04436165**	6	11	**13**
Tylko sieci typu NET 120 (sieci typu "grid")	0.52311991	**0**	0	3	**27**
Eksperyment A	0.08028913	**0.04547144**	9	24	**27**
Eksperyment B	0.16653179	**0.04231651**	10	18	**32**
Eksperyment C	0.46235004	**0.00539895**	2	3	**55**

Na podstawie przeprowadzonych badań można stwierdzić, że algorytm MuPPetS-FuN Active jest lepszy od algorytmu HEFAN 2.2 dla każdej klasy zadań. Wydaje się być interesujące, że największą przewagę nad algorytmem HEFAN 2.2, algorytm MuPPetS-FuN Active osiąga w tych klasach zadań, w których jest słabszy od algorytmu LRH, a więc dla eksperymentów typu C i sieci typu „grid”. Może to oznaczać, że te klasy zadań są trudne do rozwiązania dla algorytmów bazujących na idei algorytmu ewolucyjnego (dlatego najefektywniejszym narzędziem do ich rozwiązywania pozostał algorytm LRH), jednak dzięki użytym w algorytmie MuPPetS-FuN Active mechanizmom, ta słabość została znacząco ograniczona. Należy jednak pamiętać, że przewaga algorytmu MuPPetS-FuN nad HEFAN 2.2 jest bezdyskusyjna bez względu na klasę zadań, a więc wzrost efektywności w porównaniu z algorytmem HEAFAN 2.2, który był punktem wyjścia do badań, nastąpił na każdym polu.

6. Podsumowanie

Głównym celem niniejszej rozprawy było zaproponowanie nowej metody obliczeniowej wykorzystującej podejście ewolucyjne, efektywniejszej od nieporządnego algorytmu genetycznego i innych metod ewolucyjnych, a także skuteczniejszej w zastosowaniach praktycznych i o mniejszej czułości na ustawienia parametrów sterujących. Wszystkie te cechy są bardzo ważne z punktu widzenia możliwości stosowania AG do trudnych zadań praktycznych.

Przeprowadzone badania eksperymentalne pozwalają stwierdzić, że tak postawiony cel został osiągnięty.

Algorytm MuPPetS okazał się bardzo skuteczny w porównaniu z innymi metodami rozwiązującymi problemy zbudowane na bazie funkcji zwodniczych (rozdział 3). W porównaniu z metodą fmGA, która również używa kodowania nieporządnego, MuPPetS jest lepszy w każdym testowanym przypadku. Porównanie z metodą BOA również wypada korzystnie dla MuPPetS. Dla problemów bez ‘ogona’ (dodatkowy element problemu, który jest łatwy do rozwiązania) obie metody zwracają wyniki o porównywalnej jakości, choć BOA wypada dla tej grupy problemów nieco lepiej. Jednak w przypadku problemów z ‘ogonem’ metoda MuPPetS daje znacząco lepsze wyniki. Należy zauważyć, że problemy praktyczne zazwyczaj zawierają składniki analogiczne do ‘ogona’.

Należy również podkreślić elastyczność zaproponowanej metody – jak pokazują badania przeprowadzone w rozdziałach 3 i 4 MuPPetS posiada zdolność do wydostawania się z obszarów lokalnego optimum. Dzięki temu metoda przy wydłużeniu czasu obliczeń jest w stanie poprawiać proponowane wyniki. Istotną zaletą jest również prosta procedura dostrajania metody. Kolejną zaletą jest stosunkowo niski nakład obliczeniowy wykorzystywany przez MuPPetS w porównaniu z liczbą wyliczeń funkcji oceny przystosowania.

Wymienione wcześniej zalety metody MuPPetS udało się osiągnąć dzięki dwóm zasadniczym elementom, które legły u podstaw konstrukcji metody:

- Zaproponowaniu i użyciu wzorców genów, które pozwalają na zwiększenie efektywności wymiany informacji pomiędzy kompletnymi rozwiązaniami
- Zaproponowaniu wirusów - nowego rodzaju osobników, które używają kodowania nieporządnego i służą do przeglądania podprzestrzeni rozwiązań, dla już istniejących kompletnych propozycji

Warto zauważyć, że zaproponowany w pracy nowy rodzaj osobników, nazwanych wirusami, jest tylko pozornie podobny do osobników zakodowanych nieporządnie. Zasadnicze różnice wskazano w tabeli Tabela 27 Jak widać kodowanie i związane z operatory są jedyną cechą wspólną osobników zakodowanych nieporządnie i wirusów. Różni je cel użycia, sposób inicjalizacji, oraz użycie operatorów mutacji, które w nieporządnym algorytmie genetycznym przeszkadzałyby i pogarszałyby wyniki, a w przypadku MuPPetS są ważnymi elementami metody. Innymi słowy można powiedzieć, że metody MuPPetS, mGA i fmGA używają tego samego narzędzia (kodowanie nieporządne), ale do zupełnie innych zastosowań.

Tabela 27. Różnice pomiędzy osobnikami zakodowanymi nieporządnie (ang. *messy individual*), a wirusami

	Messy Individual (mGA, fmGA)	Wirus (MuPPetS)
Kodowanie	Nieporządne	Nieporządne
Cel	Utworzenie kompletnych rozwiązań	Przegląd podprzestrzeni
Inicjalizacja	PCI/Primordial (bliskie przeglądowi zupełnemu)	Losowa, niski koszt obliczeniowy, użycie wzorców genów
Mutacja	Brak	Istotny element metody

Należy zauważyć, że metoda MuPPetS-FuN Active, przeznaczona do proponowania rozwiązań dla problemu praktycznego (projektowanie przepływu bez rozgałęzień w szkieletowych sieciach komputerowych zorientowanych połączeniowo), wykonana z użyciem szablonu pracy zaproponowanym w MuPPetS okazała się skuteczniejsza od dotychczas znanych metod. W porównaniu z metodą HEFAN 2.2 używającą szablonu pracy typowego dla algorytmu genetycznego, metoda MuPPetS-FuN Active okazała się znacząco lepsza dla wszystkich podtypów sieci, na których przeprowadzono badania. W porównaniu z algorytmem LRH, MuPPetS-Fun Active był skuteczniejszy dla 7 z 12 różnych podtypów sieci, słabszy tylko dla 4 podtypów, a dla jednego podtypu sieci obie metody zwróciły wyniki o podobnej jakości.

Głównymi elementami nowatorskimi pracy są:

1. Propozycja i wykorzystanie wzorców genów w metodzie MuPPetS. Dzięki temu rozwiązaniu możliwa jest efektywna wymiana informacji pomiędzy koewoluującymi populacjami wirusów, poszczególnymi CT, a także wprowadzenie mechanizmów przypominających kierunkowanie uwagi. Wszystkie te elementy znacząco poprawiły jakość proponowanych przez metodę wyników.
2. Koncepcja i sposób wykorzystania wirusów (rozdz. 3.5.2.1) – „nieporządnie" zakodowanych osobników (ang. *messy coded*), które wykorzystywane są do wprowadzania drobnych (w kontekście długości całego genotypu) poprawek w genotypie. Jest to podejście przeciwne do prezentowanego w algorytmach mGA (ang. *messy GA*) i fmGA (ang. *fast messy GA*), gdzie „nieporządnie" zakodowane osobniki są wykorzystywane jako bloki budujące kompletne, lub niemal kompletne rozwiązania większych problemów. Takie podejście było jednym z elementów, który pozwolił na rezygnację ze stosowania fazy Primordial, bądź PCI będących jednymi z głównych mankamentów algorytmów mGA i fmGA
3. Koewolucja wielu populacji wirusów przypisanych do poszczególnych CT. Koewolucja wielu populacji wirusów pozwoliła na redukcję prawdopodobieństwa utykania metody w obszarach optimów lokalnych.
4. Płynna regulacja liczby koewoluujących CT w trakcie przebiegu algorytmu. Dzięki temu mechanizmowi możliwe jest lepsze wykorzystanie zasobów przeznaczonych na działanie algorytmu. Na podstawie przykładowych przebiegów algorytmu MuPPetS

można zauważyć, że jeśli algorytm utyka w obszarze optimum lokalnego, z którego trudno się wydostać, liczba koewoluujących CT (i populacji wirusów) zwiększa się, a po wyjściu z obszaru optimum lokalnego zmniejsza.

Należy zauważyć, że wymienione powyżej nowatorskie elementy pracy nie miałyby wielkiego znaczenia w oderwaniu od siebie. Zdaniem autora wszystkie należy traktować jako składniki większej całości.

W zastosowaniu praktycznym - zaproponowana nowa metoda projektowania przepływu, bazująca na metodzie MuPPetS, jest wyraźnie efektywniejsza, niż algorytm HEFAN również bazujący na idei ewolucji. Zaproponowana metoda MuPPetS-FuN Active, dla sieci o nieregularnej siatce, jest również efektywniejsza niż algorytm LRH.

Opisywana metoda posiada szereg pożądanych cech, takich jak:

- Niska zależność efektywności algorytmu od parametrów wejściowych
- Bardzo wysoka efektywność w porównaniu z innymi metodami ewolucyjnymi
- Algorytm może zostać łatwo dostosowany do pracy równoległej
- Spadek efektywności związany z koniecznością użycia stosunkowo długich ciągów genów został znacząco ograniczony

Należy również zauważyć, że przeprowadzone w niniejszej pracy badania wskazują, że szablon pracy proponowany przez algorytm MuPPetS, może być podstawą do opracowania efektywniejszych niż obecnie metod rozwiązujących inne, niż problem projektowania przepływu, trudne problemy obliczeniowe występujące w praktyce. Wniosek ten potwierdza również efektywne użycie pojedynczych populacji wirusów w pracach [79, 80].

Zdaniem autora, w toku dalszych prac nad rozwojem idei proponowanych przez algorytm MuPPetS badania powinny przebiegać w następujących kierunkach:

- Analiza czy strategia dodawania i usuwania CT zaproponowana w algorytmie MuPPetS-FuN Active poprawi efektywność algorytmu MuPPetS również dla problemów badanych w pracy [57]
- Badania nad zaproponowaniem alternatywnych metod generacji wzorców genów, poprawiających efektywność algorytmu
- Użycie idei algorytmu MuPPetS w rozwiązywaniu innych problemów praktycznych

Należy pamiętać, że w trakcie dalszych badań mogą pojawić się nowe obiecujące potencjalne zwiększenie efektywności metody obszary. Badania prowadzone w niniejszej pracy są kontynuowane i obecnie koncentrują się na strategii zwiększania i zmniejszania liczby koewoluujących CT. Nie zmienia to jednak faktu, że zarówno sam algorytm MuPPetS, jak i bazujący na nim algorytm MuPPetS-FuN Active rozwiązujący problem praktyczny dowiodły swojej wysokiej skuteczności w porównaniu z innymi dostępnymi metodami.

W tym miejscu pragnę podziękować wszystkim osobom, które wspierały mnie w trakcie bardzo długiej drogi pomiędzy rozpoczęciem, a zakończeniem prac nad niniejszą rozprawą. Przede wszystkim podziękowania chciałbym złożyć Pani prof. dr hab. inż. Halinie Kwaśnickiej, bez której życzliwości, cierpliwości i pomocy ta praca nigdy by nie powstała, swojej Żonie i Rodzicom za wszystko to, co w życiu najlepsze, oraz prof. nadzw. dr hab. Krzysztofowi Walkowiakowi za wieloletnią i owocną współpracę. Szczególne podziękowania kieruję również do prof. dr hab. Leszka Paradowskiego i dr hab. Anny Bireckiej, za bezwzględny spokój i medyczny profesjonalizm, dzięki którym mogę funkcjonować bez obaw o moje zdrowie.

Michał Przewoźniczek

Bibliografia

1. J. Anderson, B. Doshi, S. Dravida, P. Harshavardhana, „Fast Restoration of ATM Networks" in *IEEE J. Select. Areas Commun.*, No. 1, Vol. 12, 1994, pp. 128-138

2. A. Assad, "Multicommodity network flows – a survey", *Networks,* Vol. 8, 1978, s.37-91

3. E. Ayanoglu, R. Gitlin, „Broadband Network Restoration", in *IEEE Communications Magazine*, No. 6, Vol. 34, 1996, pp. 110-119

4. D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, J. McManus, „"Requirements for traffic engineering over MPLS", *RFC 2702*, 1999

5. B. Bagula, M. Botha, A. Krzesiński, "Online traffic engineering: the Least Interference Optimization Algorithm", *Proc. Of IEEE International Conference on Communications ICC*, Paris 2004, s. 1232-1236

6. A. Banerjee, J. Drake, J. Lang, B. Turner, D. Awduche, L. Berger, K. Kompella, Y. Rekhter, „Generalized Multiprotocol Label Switching: An Overview of Signaling Enhancements and Recovery Techniques" in *IEEE Communications Magazine*, No. 7, Vol. 39, 2001, s. 144-151

7. N. A. Barricelli, "Symbiogenetic evolution processes realized by artificial methods", *Methods*, s.143–182, 1957.

8. Y. Bo, S. Xiaohong, W. Yadong, "Back propagation based on selective attention for fast convergence of training neural network", *Proc. 5th World Congress on Intelligent Control and Automation*, vol. 3, 2004, s. 2009-2013.

9. J. Burns, T. Ott, A. Krzesiński, K. Muller, „Path selection and bandwidth allocation in MPLS networks", *Performance Evaluation*, Vol. 52, 2003, s. 133-152

10. T. Chen, T. Oh, „Reliable services in MPLS", *IEEE Communications Magazine*, No. 12, Vol. 37, 1999, s.58-62

11. Y. Chen, T. Yu, K. Sastry, D. E. Goldberg, „A Survey of Linkage Learning Techniques in Genetic and Evolutionary Algorithms", IlliGAL Report No. 2007014, Illinois Genetic Algorithms Laboratory, 2007

12. P. Chołda, A. Jaszczyk, „Procedury wznawiania pracy w sieciach MPLS" in *Przegląd Telekomunikacyjny*, Vol. 2-3, 2004, s.289-294

13. E. S. Correa, Shapiro J. L. "Model Complexity vs. Performance in the Bayesian Optimization Algorithm" in *Proc of 9th International Conference on Parallel Problem Solving from Nature PPSN IX*", *Lecture Notes in Computer Science*, Vol. 4193, Springer Verlag, 2006, pp. 998-1007

14. L. Davies, “Handbook of Genetic Algorithm”, Van Nostrand Reinhold, Nowy Jork, 1996

15. K. Deb D. E. Goldberg, “Sufficient Conditions for Deceptive and Easy Binary Functions” in *Annals of Mathematics and Artificial Intelligence*, Vol. 10, 1992, pp. 385-408

16. R. Dias, E. Camponogara, J.M. Farnes, R. Willrich, A. Campestrini, „Implementing traffic engineering in MPLS-based IP networks with Lagragean Relaxation” *International Conference on Computers and Communication ISCC*, 2003, Vol. 1, Antalya, 2003, s. 373-378

17. R. Doverspike, J. Yates, „Challenges for MPLS in Optical Network Restoration”, *IEEE Communication Magazine*, No. 2, Vol. 39, 2001, pp. 89-96

18. A. Dunn, W. Grover, M. MacGregor, “Comparison of k-Shortest Path and Maximum Flow Routing for Network Facility Restoration” in *IEEE Journal on Selected Areas In Communication*, No. 1, Vol.12, 1994, s. 88-99

19. R. Elbaum, M. Sidi, „Topological Design of Local Area Networks Using Genetic Algorithms”, *IEEE/ATM Transactions On Networking*, Vol. 5, 1996, s. 766-778

20. M. Ericsson, M. Rescende, P. Pardalos „A Genetic Algorithm for the weight setting problemin OSPF Routing”, *J. of Combinatorial Optimization*, Vol. 6, 2002, s. 299-333

21. W. Findeisen, J. Szymanowski, A. Wierzbiski, „Teoria i metody obliczeniowe optymalizacji”, PWN, Warszawa, 1980

22. D. B. Fogel “Evolutionary Computation. Toward a New Philosophy of Machine Intelligence”, IEEE Press, Piscataway, NJ, 1995

23. L.R. Ford, D.R. Fulkerson, „Przepływy w sieciach”, PWN, Warszawa 1969

24. L. Fratta, M. Gerla, L. Kleinrock, „The Flow Deviation Method: An Approach To Store and Forward Communication Network Design”, *Networks*, Vol. 3, 1973, s.97-33

25. A. S. Fraser, “Simulation of genetic systems by automatic digital computers”, *Australian Journal of Biological Science*, Vol. 10, s. 484-499, 1957

26. A. S. Fraser, "Simulation of genetic systems by automatic digital computers: 5- linkage, dominance and epistasis,", *Biometrical Genetics*, New York, s. 70-83, 1960

27. A. S. Fraser, “Simulation of genetic systems”, *Journal of Theoretical Biology*, Vol. 2, No. 329, s. 346, 1962.

28. L.M. Gambardella, M. Origo, “Solving symetric and asymetric TSPs by ant colonies”, *International Conference on Evolutionary Computation*, 1996, s. 622-627

29. B. Gavish, S. Huntler, „An Algorithm for Optimal Route Selection in SNA Networks”, *IEEE Trans. Commun.*, Vol. COM-31, No. 10, 1983, s. 1154-1160

30. B. Gavish, I. Neuman, „A System for Routing and Capacity Assignment in Computer Communication Networks” *IEEE Trans. Commun.*, Vol. 37, No. 4, 1989, s. 360-366

31. B. Gavish, I. Neuman, „Routing in a Network with Unreliable Components” *IEEE Trans. Commun.*, Vol. 40, No. 7, 1992, s. 1248-1258

32. A. Gershit, A. Shulman, „Architecture for Restorable Call Allocation and Fast VP Restoration in Mesh ATM Networks” in *IEEE Trans. Commun.*, No. 3, Vol. 47, 1999, pp. 397-403

33. A. Girard, B. Sanso, “Multicommodity Flow Models, Failure Propagation and Reliable Loss Network Design”, *IEEE/ACM Trans. Networking*, Vol. 6, No. 1, 1998, s. 82-93

34. D. E. Goldberg, B. Korb, K. Deb, “Messy genetic algorithms: Motivation, analysis, and first results”, *Complex Systems*, Vol. 3, 1989, pp. 493-530

35. D. E. Goldberg, K. Deb, B. Korb, “Don't worry, be messy”, *Prcs. 4th International Conference on Genetic Algorithms*, San Diego, 1991, pp. 24-30.

36. D. E. Goldberg, K. Deb, H. Kargupta, G. Harik, “Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms”, *Prcs. 5th International Conference on Genetic Algorithms*, San Mateo, 1993, pp. 56-64.

37. W. Grover, „Mesh-based Survivable Networks: Options and Strategies for Optical, MPLS, SONET and ATM Networking”, Prentice Hall PTR, Upper Saddle River, New Jersey, 2004

38. M. Held, R. Karp, „The travelling salesman problem and minimum spanning trees”, *Operation Research*, Vol. 18, No. 6, s. 1138-1162

39. G. R. Harik, “Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms”, Doctoral dissertation, University of Michigan, Ann Arbor, MI, 1997

40. G. R. Harik, “Linkage learning via probabilistic modeling in the ECGA”, IlliGAL Report No. 99010, Urbana, IL, 1999

41. J.H. Holland, “Adaptation in natural and artificial systems”, The MIT Press, 1975

42. K. Holmberg, D. Yuan, „A Langraegan Approach to Network Design Problems”, *International Transactions in Operational Research*, Vol.5, No.6, 1998, 529-539

43. K. Holmberg, D. Yuan, „A Langraegan Heuristic Based Branch-and-Bound Approach for the Capacitated Network Design Problem”, *Operations Research*, Vol.48, 2000, 461-481

44. B. Jeager, D. Tipper, „Priorized traffic restoration in connection oriented QoS based networks” in *Computer Communications*, Vol. 26, Issue 18, 2003, pp.2025-2036

45. D.S. Johnson, L.A. McGeoch, “The traveling salesman problem: A case study in local optimization”, *Local Search in Combinatorial Optimization*, John Wiley & Sons, New York, 1997, pp.215-310

46. A. Juttner, B. Szviatovszki, I. Mecs, Z. Rajko, „Lagrange Relaxation Based Method for the QoS Routing Problem", IEEE INFOCOM, 2001

47. H. Kargupta, "SEARCH, polynomial complexity, and the fast messy genetic algorithm", Technical report 95008. Urbana, IL: University of Illinois at Urbana-Champaign, 1995

48. A. Kasprzak, „Algorytmy równoczesnej optymalizacji przepływów, przepustowości kanałów i struktur topologicznych sieci teleinformatycznych", Wydawnictwo Politechniki Wrocławskiej, Wrocław 1989

49. A. Kasprzak, „Rozległe sieci komputerowe z komutacja pakietów", Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 1999

50. A. Kasprzak, „Projektowanie struktur rozległych sieci komputerowych", Oficyna Wydawnicza Politechniki Wrocławskiej, 2001

51. R. Kawamura, H. Ohta, „Architectures for ATM Network Survivability and Their Field Deployment" in *IEEE Communication Magazine*, No. 8, Vol. 37, 1999, pp. 88-94

52. R. Kawamura, Sato K., I. Tokizawa, „Self-healing ATM Networks Based on Virtual Path Concept" in *IEEE J. Select. Areas Community*, No. 1, Vol. 12, 1994, pp. 120-127

53. R. Kawamura, I. Tokizawa, „Self-healing Virtual Path Architecture in ATM Networks" in *IEEE Communications Magazine*, No. 9, Vol. 33, 1995, pp. 72-79

54. J.L. Kennington, „A Survey of Linear Cost Multicommodity Networks Flows", *Operations Research*, Vol. 26, 1978, pp. 209-236

55. L. Kleinrock, "Communication Nets: Stochastic Message Flow and Delay", McGraw-Hill, New York, 1964

56. J.L. Kulikowski, „Zarys teorii grafów", PWN, Warszawa, 1986

57. H. Kwaśnicka, Przewoźniczek M., „MuPPetS – a new evolutionary method based on the idea of messy GA", *IEEE Transactions on evolutionary computation*, No. 5, Vol. 15, 2011, pp. 715-734

58. H. Kwaśnicka, „Ewolucyjne projektowanie sieci neuronowych" *Oficyna Wydawnicza PWr*, Wrocław, 2007

59. H. Kwaśnicka, M. Paradowski, "Efficiency aspects of neural network architecture evolution using direct and indirect encoding", *Adaptive and natural computing algorithms. Proceedings of the international conference*. pp. 405-408, 2005

60. H. Kwaśnicka, "Genetic and Evolutionary Algorithms - an Overview", Wydawnictwo Politechniki Wrocławskiej, 1998

61. J. Lawrance, „Designing Multiprotocol Label Switching Networks", *IEEE Communications Magazine*, No. 5, Vol. 39, 2001, pp. 134-142

62. Z.J. Lee, “A hybrid algorithm applied to travelling salesman problem”, *IEEE Internetional Conference on Networking*, Vol. 1, 2004, pp. 237-242

63. J. Marzo, E. Calle, C. Scoglio, T. Anjah, „QoS online routing and MPLS multilevel protection: a survey” in *IEEE Communications Magazine*, No. 10, Vol.41, 2003, pp. 126-132

64. T.A.S. Masutti, L.N. deCastro, “A self organizing neural network using ideas from the immune system to solve the travelling salesman problem”, *Information Sciences*, 2009, pp. 1454-1468

65. Z. Michalewicz, “Genetic Algorithms + Data Structures = Evolution Programs”, 3rd edn. Springer-Verlag, Berlin Heidelberg New York, 1996

66. W. Molisz, „Przeżywalność sieci teleinformatycznych i telekomunikacyjnych”, Monografie 31, Wydawnictwo Politechniki Gdańskiej, 2002

67. K. Murakami, H. Kim, “Virtual Path Routing for Survivable ATM Networks”, *IEEE/ACM Trans. Networking*, No. 2, Vol. 4, 1996, pp.22-39

68. K. Murakami, H. Kim, „Optical Capacity and Flow Assignment for Self-Healing ATM Networks Based on Line and End-toEnd Restoration”, *IEEE/ACM Trans. Networking*, No. 2, Vol. 6, 1998, pp. 207-221

69. P. Nilsson, M. Pióro, Z. Dziong, „Link protection within an Existing Backbone Network”, *Proc. of Internetional Network Optimization Conference INOC*, 2003

70. E. Oki, K. Shiomoto, D. Shimazaki, N. Yamanaka, W. Imajaku, Y. Takigawa, „Dynamic multilayer routing schemes in GMPLS-based IP+optical networks”, *IEEE Communications Magazine*, No. 1, Vol. 43, 2005, pp. 108-114

71. J. Pearl, “Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference”, 1988 San Mateo, CA: Morgan Kaufman Publishers

72. M. Pelikan, D. Goldberg, E. Cantu-Paz, „Linkage Problem, Distribution Estimation, and Bayesian Networks”, IlliGAL Report No 98013, 1998

73. M. Pelikan, D. Goldberg, E. Cantu-Paz, “BOA: The Bayesian Optimization Algorithm”, IlliGAL Report No 99003, 1999

74. Piechowiak M., Zwierzykowski P., „Algorytmy typu ‘multicast’ w sieciach pakietowych”, Poznańskie Warsztaty Telekomunikacyjne, 2004, s.1-6

75. M. Pióro, D. Medhi, „Routing, Flow, and Capacity Design in Communication and Computer Networks”, Morgan Kaufman Publisher, 2004

76. M. Przewoźniczek, K. Walkowiak, “Evolutionary Algorithm for Congestion Problem in Connection-Oriented Networks”, *Lecture Notes in Computer Science*, Vol. 3483, Springer Verlag, 2005, pp. 802-811.

77. M. Przewoźniczek, K. Walkowiak, “Quasi-hierarchical Evolution Algorithm for Flow Assignment in Survivable Connection-Oriented Networks”, *International Journal of Applied Mathematics and Computer Science*, No. 4 Vol. 16, 2006, pp. 101-116.

78. M. Przewoźniczek, K. Walkowiak, “Quasi-hierarchical Evolutionary Algorithm for Flow Optimization in Survivable MPLS Networks”, *Lecture Notes in Computer Science*, Vol. 4707, Springer Verlag, 2007, pp. 330-342.

79. M. Przewoźniczek, K. Walkowiak, “Modeling and optimization of survivable P2P multicasting”, *Computer Communications*, Vol. 34, Issue 8, Elsevier, 2011, s. 921-1042

80. M. Przewoźniczek, K. Walkowiak, M. Woźniak, “Optimizing distributed computing systems for k-nearest neighbors classifiers - evolutionary approach”, *Logic Journal of the IGPL*, Vol. 19, Issue 2 , 2011, pp. 357-372

81. B. Puype, A. Groebbens, S. De Maesschlack, D. Colle, I. Lievens, M. Pickavet, P. Demeester, „Benefits of GMPLS for multilayer recovery”, *IEEE Communications Magazine*, No. 7, Vol. 43, 2005, pp. 51-59

82. K. Pytel., G. Kluka, „Wykorzystanie logiki rozmytej do wspomagania ewolucji osobników w algorytmach genetycznych”, *Studia z Automatyki i Informatyki*, tom 27, Poznańskie Towarzystwo Przyjaciół Nauk, Poznań, 2002

83. N. Radcliffe, P. Surry, “Co-operation through Hierarchical Competition in Genetic Data Mining”, *Technical Report EPCC-TR94-09*, Edinburgh Parallel Computing Center, 1994

84. G. Retvari, T. Cinkler, J. Biro, „A novel Langragian-relaxation to the minimum cost multicommodity flow problem and its application to OSPF traffic engineering”, *11th IEEE Symposium on Computers and Communications ISCC*, Alexandria, 2004, s. 957-962

85. A. Riedl, „A Versatile Genetic Algorithm for Network Planning” *Proceedings of EUNICE'98*, 1998, s. 97-103

86. A. Riedl, „A Hybrid Genetic Algorithm for Routing Optimization in IP Networks Utilizing Bandwidth and Delay Metrics”, *Proceedings of IPOM*, 2002

87. M. Rocha, P. Sousa, M. Rio, P. Cortez “QoS Constrained Internet Routing with Evolutionary Algorithms”, *Proc. 2006 IEEE Congress on Evolutionary Computation*, Vancouver, 2006, pp. 2720-2727

88. E. Rosen, A. Viswanathan, R. Callon, „Multiprotocol label switching architecture”, RFC 3031, 2001

89. D. Rutkowska, M. Piliński, L. Rutkowski, „Sieci neuronowe, algorytmy genetyczne i systemy rozmyte”, Wydawnictwo Naukowe PWN, Warszawa, 1999.

90. S. M. Sait, M. H. Sqalli, M. A. Mohiuddin, “Engineering Evolutionary Algorithm to Solve Multi-objective OSPF Weight Setting Problem”, *Lecture Notes in Computer Science*, Vol. 4304, Springer Verlag, 2006, pp. 950-955.

91. S. Sengupta, et al.: „Switched optical backbone for cost-effective scalable core IP networks" in *IEEE Commun. Mag*. No. 6, Vol. 41, 2003, pp. 60–70

92. V. Sharma, F. Hellstrand (ed.), „Framework for MPLS-based Racovery", RFC 3469, 2003

93. K. Shinozawa, T. Uchiyama, K Shimohara, "An approach for solving dynamic TSPs using neural networks", *Neural Networks, IEEE International Joint Conference*, Vol. 3, 1991, pp. 2450-2454

94. J. Schmidhuber, "Simple Algorithmic Principles of Discovery, Subjective Beauty, Selective Attention, Curiosity & Creativity", *Proc. 10th Intl. Conf. on Discovery Science*, LNAI, Vol. 4755, Springer, 2007, s. 26-38.

95. W. Szeto, R. Boutaba, Y. Iraqi, „Dynamic online routing algorithm for MPLS traffic engineering" in *Lecture Notes in Computer Science*, Vol. 2345, 2002, pp.936-946

96. A. Tannenbaum, „Sieci komputerowe", Wydawnictwo Helion, Gliwice 2004

97. A. Tannenbaum, „Structured Computer Organization, 5th Edition", Prentice Hall, 2005

98. T. Van Landegem, P. Van Kwikelberge, H. Van Derstraeten, „A Self-Healing ATM Network Based on Multilink Principles" in *IEEE J. Select. Areas Commun.*,Vol. 12, 1994, pp.139-148

99. J. Vasseur, M. Pickavet, P. Demeester, „Network recovery: Protection and Restoration of Optical , SONET-SDH, IP and MPLS", Morgan Kaufmann Publishers, San Francisco, 2004

100. K. Walkowiak, „Assignment of virtual paths in survivable ATM networks for local-destination rerouting" in *Proc. of 3rd Internetional Workshop on Design of Reliable Communication Networks DRCN*, 2001, pp. 273-280

101. K. Walkowiak, „Genetic approach to virtual paths assignment in survivable ATM networks" in *Proc. of 7th Internetional Conference on Soft Computing MENDEL*, 2001, pp. 13-18

102. K. Walkowiak, „Niezawodność w sieciach komputerowych zorientowanych połączeniowo" in *Telekomunikacja Cyfrowa – Technologie i Usługi*, Vol. 4, 2001-2002, pp. 38-48

103. K. Walkowiak, „A New Approach to Survivability of Connection Oriented Networks" in *Lecture Notes in Computer Science*, Vol. 2657, 2003, pp. 501-510

104. K. Walkowiak, „The Branch and Bound Algorithm for Joint Optimization of Primary and Backup Routes in Survivable Networks" in *Archiwum Informatyki Teoretycznej i Stosowanej*, No. 4, Vol. 15, 2003, pp. 273-296

105. K. Walkowiak, „Heuristic algorithms for assignment of non-bifurcated multicommodity flows" in *Proc. of Advanced simulation of ASIS*, Acta MOSIS, No. 93, Sv. Hostyn, 2003, pp. 243-248

106. K. Walkowiak, „A New Method of Primary Routes Selection for Lacal Restoration" in *Lecture Notes in Computer Science*, Vol.3042, 2004, pp. 1024-1035

107. K. Walkowiak, „Analysis of a New Function for Local Restoration of Survivable Connection-Oriented Networks" in *Proc. of Advanced simulation of ASIS*, Acta MOSIS, No. 98, Sv. Hostyn, 2004, pp. 80-85

108. K. Walkowiak, „A Branch and Bound Algorithm for Primary Routes Assignment in Survivable Connection Oriented Networks" in *Computational Optimization and Applications*, No. 2, Vol. 27, 2004, pp. 149-171

109. K. Walkowiak, „A Unified Approach to Survivability of Connection-Oriented Networks" in *Lecture Notes in Computer Science*, Vol.3733, 2005, pp. 3-12

110. K. Walkowiak, „The Hop-Limit Approach for Flow Assignment in Survivable Connection Oriented Networks" in *New Trends in Computer Networks*, Vol.1, Imperial College Press, 2005

111. K. Walkowiak, „Analysis of cut ineaqualities for optimization of non-bifurcated multicommodity flows in survivable networks" in *Proc. of 4th Polish-German Teletraffic Symposium*, Wrocław, 2006, pp. 213-218

112. K. Walkowiak, „Algorytmy wyznaczania przepływów typu unicast i anycast w przeżywalnych sieciach zorientowanych połączeniowo", Oficyna Wydawnicza Politechniki Wrocławskiej, 2007

113. C.M. White, G.G. Yen, "A hybrid evolutionary algorithm for travelling salesman problem", *Congress on Evolutionary Computation*, CEC2004, Vol. 2, 2004, pp.1473-1478

114. J. Woźniak, K. Nowicki, „Sieci LAN, MAN, i WAN – protokoły komunikacyjne", Wydawnictwo Fundacji Postępu Telekomunikacji, Kraków, 1998

115. Zhang F., Zheng X., Zhang H., Guo Y., „A kind of topology aggregation algorithm in hierarchical wavelength-routed optical networks" in *Photonic Network Communications*, No. 2, Vol. 9, no. 2, 2005, pp. 167-180

Załącznik A - wyniki

Tabela 28. Funkcje testowe bez „ogona” – wyniki, część główna

Funkcja testowa	Odsetek ‘1’ w genotypie			Optymalne rozwiązania			Liczba wyliczeń funkcji oceny przystosowania			Czas [s]		
	fmGA	BOA	MuPPetS	fmGA	BOA	MuPPetS	fmGA	BOA	MuPPetS	fmGA	BOA	MuPPetS
30 (only 3) flat normal	1	1	1	10	10	10	150335.7	17750	37757.3	11.364	0.519	1.275
30 (only 3) flat hard	1	1	1	10	10	10	389658.1	23250	121293.1	37.932	0.724	3.999
30 (only 3) spike normal	0.88	1	1	2	10	10	141551.6	22250	38351.3	10.206	0.682	1.151
30 (only 3) spike hard	0.84	1	1	1	10	10	849911.5	27000	141107.8	100.052	0.83	4.714
50 (only 3) flat normal	0.94	1	1	4	10	10	504528.9	54500	161171	41.502	3.726	7.399
50 (only 3) flat hard	0.899	1	1	2	10	10	1990739.7	67000	449417.2	202.175	4.695	22.072
50 (only 3) spike normal	0.868	1	1	0	10	10	1042173.5	74500	226657.4	92.842	5.171	10.355
50 (only 3) spike hard	0.742	1	1	0	10	10	1384774.4	90500	404497.1	128.966	6.171	19.809
50 (3 and 5) flat normal	0.92	1	1	5	10	10	1390148.1	52500	362934.2	119.062	3.414	15.387
50 (3 and 5) flat hard	0.882	1	1	4	10	10	2593310.8	78000	814418.4	253.905	5.192	35.966
50 (3 and 5) spike normal	0.84	1	1	0	10	10	1730171.5	73000	449822.2	143.244	4.87	18.838
50 (3 and 5) spike hard	0.704	1	1	0	10	10	1947346.1	90500	628779.7	165.731	5.913	27.626
50 (only 5) flat normal	1	1	1	10	10	10	885486	50000	340269.1	66.028	3.086	13.803
50 (only 5) flat hard	0.81	1	1	1	10	10	4104340.1	67500	779242.7	397.107	4.44	33.191
50 (only 5) spike normal	0.778	1	1	0	10	10	3160274.1	79000	902534.7	325.997	4.909	35.299
50 (only 5) spike hard	0.578	0.99	1	0	9	10	2085433.2	93500	681439.6	170.877	125.253	28.284
150 (only 3) flat normal	0.715	1	1	0	10	10	4397625	856000	2123900	837.709	417.357	213.154
150 (only 3) flat hard	0.528	1	1	0	10	10	19338985	1280000	8587258	4077.136	600.331	843.793
150 (only 3) spike normal	0.698	1	1	0	10	10	11420991	1228000	4436650	2416.16	555.673	416.3
150 (only 3) spike hard	0.494	1	1	0	10	10	30183790	1564000	9612991	6895.034	717.427	927.623
150 (3 and 5) flat normal	0.787	1	1	0	10	10	7825034.5	860000	3150959	1452.175	416.594	298.82
150 (3 and 5) flat hard	0.582	1	0.994	0	10	8	23572137	1348000	21098146	4467.099	624.396	1793.914
150 (3 and 5) spike normal	0.785	1	0.983	0	10	7	21200151	1224000	18232037	4047.358	555.027	1639.754
150 (3 and 5) spike hard	0.56	0.997	0.993	0	9	9	20467430	1584000	18348130	3860.465	723.649	1701.216
150 (only 5) flat normal	0.679	1	1	0	10	10	11400530	748000	4449383	2021.113	358.364	363.698
150 (only 5) flat hard	0.371	1	0.967	0	10	6	40184625	1196000	35597072	7699.98	578.157	3093.244
150 (only 5) spike normal	0.645	1	1	0	10	10	18736019	1148000	9648771	3443.204	506.847	792.018
150 (only 5) spike hard	0.294	1	0.974	0	10	6	32276843	1700000	27833477	6146.791	757.033	2372.733
300 (only 3) flat normal	0.639	1	1	0	10	10	18900499	1312000	8944733	6489.349	3174.91	1650.467
300 (only 3) flat hard	0.582	1	0.955	0	10	2	28342039	1976000	37707593	9889.356	4448.88	6699.607
300 (only 3) spike normal	0.61	1	0.999	0	10	9	29087096	1840000	24337184	10715.96	3925.92	4134.421
300 (only 3) spike hard	0.45	1	0.942	0	10	1	28196896	2464000	38491490	10328.38	5214.48	6856.888
300 (3 and 5) flat normal	0.623	1	0.995	0	10	8	29957019	1272000	31850829	10500.27	3092.1	5179.734
300 (3 and 5) flat hard	0.508	1	0.817	0	10	0	30253639	2056000	39676140	9939.503	4708.18	6994.153
300 (3 and 5) spike normal	0.596	1	0.95	0	10	0	31445418	1864000	44951478	11404.07	4214.23	6997.711
300 (3 and 5) spike hard	0.383	0.982	0.759	0	2	0	30766227	2648000	43339802	10845.01	6085.99	6995.551
300 (only 5) flat normal	0.518	1	0.998	0	10	9	33573519	1152000	38221038	11147.55	2789.68	6038.308
300 (only 5) flat hard	0.494	1	0.645	0	10	0	28900573	1896000	53111076	21490.27	4495.34	6997.538
300 (only 5) spike normal	0.481	1	0.913	0	10	0	34954247	1696000	48738294	11718.64	3693.16	6996.678
300 (only 5) spike hard	0.378	0.964	0.627	0	2	0	33376623	2704000	46323216	11358.39	5896.81	6998.39

Tabela 29. Funkcje testowe bez „ogona” – wyniki, uzupełnienie

Funkcja testowa	Wartość					Liczba osobników w populacji fmGA	Liczba wyliczeń funkcji oceny przystosownia		
	Max	„rozwiązanie łatwe do znalezienia”	fmGA	BOA	MuPPetS		fmGA	BOA	MuPPetS
30 (only 3) flat normal	10	3.33	10	10	10	1688.9286	150335.7	17750	37757.3
30 (only 3) flat hard	10	9.8	10	10	10	4979.4286	389658.1	23250	121293.1
30 (only 3) spike normal	73	24.32	72.2	73	73	1327.7143	141551.6	22250	38351.3
30 (only 3) spike hard	73	71.1	72.933	73	73	7670.2857	849911.5	27000	141107.8
50 (only 3) flat normal	17	5.61	16.333	17	17	3346.4167	504528.9	54500	161171
50 (only 3) flat hard	17	16.66	16.967	17	17	13198.639	1990739.7	67000	449417.2
50 (only 3) spike normal	125	41.61	122.35	125	125	6910.9861	1042173.5	74500	226657.4
50 (only 3) spike hard	125	123.46	124.65	125	125	9183.375	1384774.4	90500	404497.1
50 (3 and 5) flat normal	14	4.9	13.52	14	14	6223.2766	1390148.1	52500	362934.2
50 (3 and 5) flat hard	14	13.76	13.982	14	14	11372.936	2593310.8	78000	814418.4
50 (3 and 5) spike normal	104	36.91	101.72	104	104	7153.4043	1730171.5	73000	449822.2
50 (3 and 5) spike hard	104	102.7	103.65	104	104	8048.0511	1947346.1	90500	628779.7
50 (only 5) flat normal	10	4	10	10	10	4526.2468	885486	50000	340269.1
50 (only 5) flat hard	10	**9.9**	**9.9765**	10	10	17290.834	4104340.1	67500	779242.7
50 (only 5) spike normal	73	29.2	71.58	73	73	13062.46	3160274.1	79000	902534.7
50 (only 5) spike hard	73	**72.13**	**72.116**	72.999	73	8620.1021	2085433.2	93500	681439.6
150 (only 3) flat normal	50	16.5	39.05	50	50	12987.266	4397625	856000	2123899.6
150 (only 3) flat hard	50	**49**	**44.122**	50	50	57102.743	19338985	1280000	8587257.5
150 (only 3) spike normal	365	121.5	302.95	365	365	33725.613	11420991	1228000	4436649.9
150 (only 3) spike hard	365	**357.7**	**332.56**	365	365	89124.36	30183790	1564000	9612991.4
150 (3 and 5) flat normal	46	15.6	39.427	46	46	14020.65	7825034.5	860000	3150959.2
150 (3 and 5) flat hard	46	45.14	45.311	46	45.9975	42234.854	23572137	1348000	21098146
150 (3 and 5) spike normal	334	114	306.3	334	333.7	37984.672	21200151	1224000	18232037
150 (3 and 5) spike hard	334	327.66	331.73	334	333.986	36675.367	20467430	1584000	18348130
150 (only 5) flat normal	30	12	24.24	30	30	20427.709	11400530	748000	4449383.4
150 (only 5) flat hard	30	29.7	29.742	30	29.9877	72001.834	40184625	1196000	35597072
150 (only 5) spike normal	219	87.6	179.39	219	219	33570.827	18736019	1148000	9648771.1
150 (only 5) spike hard	219	**216.39**	**215.73**	219	218.957	57831.068	32276843	1700000	27833477
300 (only 3) flat normal	100	33	61.017	100	100	30330.535	18900499	1312000	8944733.3
300 (only 3) flat hard	100	**98**	**75.733**	100	99.9118	45482.443	28342039	1976000	37707593
300 (only 3) spike normal	730	243	532.93	730	729.933	46679.293	29087096	1840000	24337184
300 (only 3) spike hard	730	**715.4**	**682.35**	730	729.057	45251.651	28196896	2464000	38491490
300 (3 and 5) flat normal	80	28.5	53.16	80	79.82	29022.599	29957019	1272000	31850829
300 (3 and 5) flat hard	80	**78.7**	**68.615**	80	79.8344	29309.892	30253639	2056000	39676140
300 (3 and 5) spike normal	584	209.1	455.43	584	582.2	30464.558	31445418	1864000	44951478
300 (3 and 5) spike hard	584	**574.09**	**571.63**	583.99	582.523	29807.182	30766227	2648000	43339802
300 (only 5) flat normal	60	24	37.61	60	59.94	32524.607	33573519	1152000	38221038
300 (only 5) flat hard	60	**59.4**	**35.625**	60	59.737	37747.859	28900573	1896000	53111076
300 (only 5) spike normal	438	175.2	284	438	434.88	33862.896	34954247	1696000	48738294
300 (only 5) spike hard	438	**432.78**	**404.28**	437.97	436.268	32334.444	33376623	2704000	46323216

Tabela 30. Funkcje testowe z „ogonem” – wyniki

Funkcja testowa	Odsetek ‘1’ w genotypie		Optymalne rozwiązania		Wartość				Liczba wyliczeń funkcji oceny przystosownia		Czas [s]	
	BOA	MuPPetS	BOA	MuP PetS	Max	„rozwiązanie łatwe do znalezienia”	BOA	MuPPetS	BOA	MuPPetS	BOA	MuPPetS
30 (only 3) flat normal	1	1	5	5	11	4.33	11	11	49000	102460.4	4.402	3.08
30 (only 3) flat hard	1	1	5	5	11	10.8	11	11	47500	463114.6	4.37	16.224
30 (only 3) spike normal	1	1	5	5	74	25.32	74	74	72000	244103.6	11.004	7.766
30 (only 3) spike hard	1	1	5	5	74	72.1	74	74	65000	344933.2	9.948	11.66
50 (only 3) flat normal	1	1	5	5	18	6.61	18	18	133000	352802.4	30.64	16.362
50 (only 3) flat hard	1	1	5	5	18	17.66	18	18	129000	1371836.8	30.98	67.086
50 (only 3) spike normal	1	1	5	5	126	42.61	126	126	164000	589042.6	37.83	24.492
50 (only 3) spike hard	1	1	5	5	126	124.46	126	126	166000	983204.2	40.294	43.678
50 (3 and 5) flat normal	1	1	5	5	15	5.9	15	15	137000	820170.8	30.066	33.054
50 (3 and 5) flat hard	1	1	5	5	15	14.76	15	15	141000	2012133.2	31.502	87.24
50 (3 and 5) spike normal	1	1	5	5	105	37.91	105	105	166000	1243649.8	34.566	46.562
50 (3 and 5) spike hard	1	1	5	5	105	103.7	105	105	169000	1344005.4	37.972	53.298
50 (only 5) flat normal	1	1	5	5	11	5	11	11	129000	1061220	27.562	39.616
50 (only 5) flat hard	1	1	5	5	11	10.9	11	11	135000	1282278.2	30.198	50.996
50 (only 5) spike normal	1	1	5	5	74	30.2	74	74	162000	1698338	34.4	57.026
50 (only 5) spike hard	1	1	5	5	74	73.13	74	74	177000	1795582.6	39.202	67.496
150 (only 3) flat normal	0.894	1	0	5	51	17.5	50.789	51	1344000	4902070.8	2515.21	590.174
150 (only 3) flat hard	0.818	0.998	0	4	51	50	50.635	50.9961	1248000	19816992	2550	2079.48
150 (only 3) spike normal	0.766	1	0	5	366	122.5	365.4	366	1200000	6975469.2	2514.98	766.952
150 (only 3) spike hard	0.666	1	0	5	366	358.7	364.95	366	1136000	15814130	2495.29	1685.016
150 (3 and 5) flat normal	0.892	0.996	0	4	47	16.6	46.785	46.88	1368000	13996672	2525.75	1352.122
150 (3 and 5) flat hard	0.794	0.988	0	3	47	46.14	46.625	46.9887	1264000	19221960	2528.79	1897.008
150 (3 and 5) spike normal	0.784	1	0	5	335	115	334.56	335	1256000	17296154	2534.51	1743.688
150 (3 and 5) spike hard	0.67	0.984	0	2	335	328.66	334.05	334.905	1200000	21746311	2558.97	2245.358
150 (only 5) flat normal	0.924	1	0	5	31	13	30.845	31	1384000	11239137	2519.38	1084.278
150 (only 5) flat hard	1	0.988	5	3	31	30.7	31	30.9901	2040000	35417948	4110.86	3337.762
150 (only 5) spike normal	0.928	1	0	5	220	88.6	219.85	220	1824000	23331755	3531.33	2040.974
150 (only 5) spike hard	0.784	0.942	0	2	220	217.39	219.66	219.736	1656000	35191777	3529.66	3322.154
300 (only 3) flat normal	0.682	1	0	5	101	34	83.467	101	768000	21568593	7057.47	5930.996
300 (only 3) flat hard	0.522	0.962	0	0	101	**99**	**92.833**	100.859	760000	31555458	7276.56	6995.488
300 (only 3) spike normal	0.642	1	0	5	731	244	639.49	730.867	744000	22943225	7244.03	6088.928
300 (only 3) spike hard	0.528	0.948	0	0	731	**716.4**	**699.66**	729.533	784000	31236588	7251.93	7000.414
300 (3 and 5) flat normal	0.662	0.96	0	0	81	29.5	64.931	78.1193	768000	28530072	7105.69	7000.488
300 (3 and 5) flat hard	0.488	0.886	0	0	81	**79.7**	**72.96**	80.7994	744000	33752947	7139.55	6998.844
300 (3 and 5) spike normal	0.632	0.932	0	0	585	210.1	499.11	572.619	728000	28749814	7131.2	6998.66
300 (3 and 5) spike hard	0.486	0.878	0	0	585	**575.09**	**547.74**	583.396	752000	33543357	7021.96	6999.4
300 (only 5) flat normal	0.68	0.908	0	0	61	25	51.075	54.398	760000	29687280	6991.34	6999.34
300 (only 5) flat hard	0.494	0.796	0	0	61	**60.4**	**54.538**	60.6988	768000	34388857	7150.03	7012.47
300 (only 5) spike normal	0.66	0.868	0	0	439	176.2	400.82	408.88	784000	30179230	7185.22	7003.214
300 (only 5) spike hard	0.474	0.812	0	0	439	**433.78**	**410.29**	437.274	784000	37397702	7234.51	6996.844

Tabela 31. Wyniki obliczeń dla funkcji LFL dla algorytmów MuPPetS-Fun i HEFAN 2.2 dla czasu obliczeń 1200 sekund

eksperyment\algorytm	MuPPetS-Fun	HEFAN 2.2
104b00	1584	1728
104b01	1204	1204
104b02	458	457
104b03	3219	3219
104b04	2657	2657
104b05	4906	4906
104b06	4083	4084
104b07	5976	5976
104b08	9823	9937
104b09	15618	15736
104d00	11400	11726
104d01	27466	27544
104d02	5113	5180
104d03	10901	11148
104d04	6040	6134
104d05	2873	2979
104d06	3492	3492
104d07	706	716
104d08	3476	3469
104d09	7109	7350
114b00	0	0
114b01	0	0
114b02	0	0
114b03	2598	2570
114b04	0	0
114b05	1106	1096
114b06	0	0
114b07	4563	4509
114b08	10038	9994
114b09	1312	1308
114d00	1661	1715
114d01	2047	2135
114d02	2329	2472
114d03	7899	8078
114d04	4372	4632
114d05	1160	1136
114d06	4761	4952
114d07	10493	10988
114d08	0	0
114d09	13133	13482
128b00	0	0
128b01	0	0
128b02	0	0
128b03	0	0
128b04	0	0
128b05	0	0
128b06	0	0

128b07	783	367
128b08	9563	9384
128b09	8463	8407
128d00	11873	12383
128d01	0	0
128d02	5618	5705
128d03	1506	1436
128d04	2967	3107
128d05	4483	4870
128d06	1309	1376
128d07	4943	5871
128d08	1381	1782
128d09	11854	12148
144b00	0	0
144b01	0	0
144b02	0	0
144b03	1836	1554
144b04	190	0
144b05	7227	7063
144b06	0	0
144b07	4881	4126
144b08	17702	17569
144b09	5806	5278
144d00	14553	14785
144d01	738	702
144d02	135	17
144d03	0	0
144d04	6416	6504
144d05	2326	2524
144d06	9429	9747
144d07	8450	8616
144d08	216	106
144d09	10466	10425
162b00	0	0
162b01	0	0
162b02	0	0
162b03	0	0
162b04	2144	1685
162b05	12663	11262
162b06	6572	6146
162b07	1669	1468
162b08	1555	1436
162b09	2233	2232
162d00	221	223
162d01	56	56
162d02	124	124
162d03	268	0
162d04	861	853
162d05	9826	10138
162d06	15427	15537
162d07	5947	5637

162d08	6365	6579
162d09	3460	3538
104050	3400	3400
104051	4680	4680
104052	5764	5764
104053	7000	7000
104054	8220	8220
104055	10095	10205
104056	13488	13856
104057	16380	17187
104058	19762	20374
104059	23190	23797
114052	0	0
114053	0	0
114054	0	0
114055	0	0
114056	2208	2208
114057	4188	4188
114058	8548	8548
114059	11506	11506
114060	14520	14520
114061	19298	19298
128078	0	0
128079	0	0
128080	0	0
128081	0	0
128082	506	0
128083	5183	4684
128084	8832	9084
128085	15905	13310
128086	25862	24132
128087	31908	31560
144079	0	0
144080	0	0
144081	0	0
144082	0	0
144083	3024	2609
144084	6372	5532
144085	11090	9815
144086	15420	14130
144087	17955	17955
144088	26712	27592
162093	0	0
162094	0	0
162095	0	0
162096	0	0
162097	0	0
162098	4118	4118
162099	7104	6609
162100	9100	9100
162101	15934	14419

162102	20490	17448
g120b00	0	0
g120b01	0	0
g120b02	0	0
g120b03	0	0
g120b04	2692	2653
g120b05	880	895
g120b06	21575	21500
g120b07	30968	31121
g120b08	48647	48613
g120b09	41941	42048
g120d00	23704	23725
g120d01	24430	24571
g120d02	2690	2741
g120d03	20202	20348
g120d04	7269	7381
g120d05	15067	15314
g120d06	7107	7300
g120d07	4259	4264
g120d08	3335	3639
g120d09	9961	10006
g120074	0	0
g120075	0	0
g120076	0	0
g120077	0	0
g120078	0	0
g120079	3478	3592
g120080	9600	9680
g120081	20379	20217
g120082	31452	31288
g120083	41079	41245

Tabela 32 Wyniki obliczeń dla funkcji LFL dla algorytmów MuPPetS-Fun, MuPPetS-Fun Active, HEFAN 2.2, LRH dla czasu obliczeń 2400 sekund

Eksperyment\algorytm	MuPPetS-FuN	HEFAN 2.2	MuPPetS-FuN Single	MuPPetS-FuN Double	MuPPetS-FuN Active
104b00	1853	1861	1862	1677	1753
104b01	1478	1247	1392	1350	1246
104b02	669	561	636	499	553
104b03	3503	3386	3287	3287	3287
104b04	2769	2876	2770	2747	2657
104b05	5213	5830	5026	5223	5244
104b06	4251	4778	4262	4540	4321
104b07	6004	6650	6064	6064	6064
104b08	10250	12004	10317	10493	10228
104b09	16287	18717	16113	16388	16128
104d00	12582	15643	12040	12420	11975
104d01	30151	33622	28817	29579	29178
104d02	613127	1087310	171824	163256	733624
104d03	11401	13172	11118	11476	11125
104d04	6879	8228	6435	6509	6490
104d05	3248	3476	2995	3028	3003
104d06	3751	4241	3553	3673	3616
104d07	1445	1364	732	711	760
104d08	4247	5064	3576	3862	3683
104d09	7378	8143	7265	7469	7262
114b00	0	0	0	0	0
114b01	0	0	0	0	0
114b02	0	0	0	0	0
114b03	3168	3340	2867	2855	2828
114b04	364	40	130	154	194
114b05	1206	1182	1274	1234	1210
114b06	335	99	54	0	129
114b07	4908	4886	4534	4742	4710
114b08	10600	11069	10060	10372	10272
114b09	1620	1656	1704	1404	1314
114d00	2613	3713	1990	2295	2103
114d01	2850	3423	2077	2389	2145
114d02	3139	4207	2349	2723	2394
114d03	11961	265350	9216	10981	9723
114d04	6306	8079	5646	5997	5489
114d05	1859	1951	1713	1685	1586
114d06	7682	9922	5886	6702	6052
114d07	14533	15846	11502	12097	11762
114d08	588	0	0	0	0
114d09	324784	907770	273879	202610	54725
128b00	0	0	0	0	0
128b01	0	0	0	0	0
128b02	0	0	0	0	0
128b03	0	0	0	0	0
128b04	636	2524	0	0	0
128b05	792	0	0	85	0

128b06	122	0	0	0	0
128b07	5540	3216	861	1550	1228
128b08	13677	23595	12222	14156	12853
128b09	11112	19145	9863	11823	11203
128d00	24770	28378	15839	22639	16584
128d01	1296	1920	183	924	407
128d02	12692	16182	7660	9550	8028
128d03	4398	7677	1914	3414	2586
128d04	6274	9949	4029	4901	4338
128d05	10427	22861	7285	10281	8311
128d06	3146	7643	1531	2942	1797
128d07	8569	13851	6226	8187	6791
128d08	4656	5196	2369	2944	2520
128d09	17132	28275	16082	17449	16542
144b00	0	0	0	0	0
144b01	0	0	0	0	0
144b02	0	0	0	0	0
144b03	2846	1902	1959	2321	2125
144b04	1028	103	603	969	250
144b05	7766	7568	7664	7682	7383
144b06	0	0	0	0	0
144b07	7602	5687	4817	6284	6254
144b08	19572	20073	18152	18943	18005
144b09	7155	5993	5533	5868	5667
144d00	18586	26356	16448	17556	16299
144d01	839	1013	739	894	857
144d02	3589	7512	1937	2692	2423
144d03	548	0	0	0	0
144d04	10627	22704	7133	8670	8211
144d05	4242	6052	2725	2904	2736
144d06	11909	19608	11498	12139	11660
144d07	11298	15204	9285	10513	9581
144d08	2115	2872	542	1206	824
144d09	10787	13651	10605	10739	10555
162b00	0	0	0	0	0
162b01	0	0	0	0	0
162b02	0	0	0	0	0
162b03	0	0	0	0	0
162b04	5078	2757	3288	5141	3204
162b05	14013	11732	13264	13408	13544
162b06	8961	9122	7837	7204	8185
162b07	7484	2251	3891	8022	2228
162b08	5082	3913	3683	3240	5024
162b09	5109	2827	3601	4528	3585
162d00	506	297	485	390	410
162d01	56	56	56	56	56
162d02	819	256	303	501	269
162d03	3375	5732	830	1542	722
162d04	5678	10202	2323	4354	3021
162d05	14936	24608	12093	13969	13031
162d06	270449	86956521	279677	35830	19215

162d07	11152	25868	6767	9900	7341
162d08	12085	7336756	8377	11484	9483
162d09	419430	69051	5170	63086	28922
104050	3600	3600	4200	3900	3500
104051	5088	5088	5088	4782	4986
104052	5972	6180	5972	5972	5868
104053	7106	7212	7318	7212	7000
104054	8352	8718	8328	8352	8232
104055	10370	11550	11275	10780	10670
104056	13688	14864	13432	13656	13264
104057	16983	17553	16950	16779	16722
104058	20600	20866	20284	20110	20052
104059	25390	26520	23848	24514	24404
114052	0	0	0	0	0
114053	0	0	0	0	0
114054	0	0	0	0	0
114055	0	0	0	0	0
114056	2208	2208	2208	2432	2208
114057	4416	4416	4188	4188	4416
114058	8548	8780	8548	8780	8548
114059	11742	11742	11506	11506	11742
114060	16320	15720	15000	14760	14760
114061	19732	19336	19336	19336	19298
128078	0	0	0	0	0
128079	0	0	0	0	0
128080	3840	0	960	2160	480
128081	846	0	0	0	0
128082	5244	0	630	156	1122
128083	12323	7093	5185	6428	5765
128084	13512	8496	10104	12108	10596
128085	25675	16770	13685	21205	16450
128086	40240	37816	31046	35924	34214
128087	46908	44298	36426	37989	38604
144079	0	0	0	0	0
144080	0	0	0	0	0
144081	0	0	0	0	0
144082	156	0	156	0	0
144083	3190	3024	3024	3024	3439
144084	7044	5532	5616	6120	5532
144085	11955	9815	10665	11090	10665
144086	16280	14560	14560	15850	14990
144087	18129	17085	18390	17955	17085
144088	33576	33840	35248	32256	33224
162093	0	0	0	0	0
162094	0	0	0	0	0
162095	0	0	0	0	0
162096	4320	0	0	960	0
162097	1336	0	366	1336	0
162098	6078	4118	4608	6568	4608
162099	10074	6609	8589	7599	8094
162100	13700	10100	12600	12600	9200

162101	19520	15429	16439	18055	15429
162102	21390	17958	20388	19488	18162
g120b00	305	279	0	0	0
g120b01	344	0	0	0	0
g120b02	160	1209	0	210	0
g120b03	870	3458	442	698	224
g120b04	7558	16148	6374	6573	6161
g120b05	8866	20235	4465	6291	4737
g120b06	29410	49611	27056	26659	26625
g120b07	44800	66233	36743	42964	38024
g120b08	72385	110192	61145	72021	63400
g120b09	60571	79058	49474	55850	50715
g120d00	35576	47908	28248	35975	28789
g120d01	43478	7283320	33147	39734	33493
g120d02	5755	9248	4224	5025	4169
g120d03	27615	41272	24207	26417	25067
g120d04	10875	15265	9817	10380	9762
g120d05	23095	32404	18550	21784	19003
g120d06	10097	13252	8597	9652	8767
g120d07	12575	1355564	5419	316652	6126
g120d08	8026	14449	4455	6429	5135
g120d09	4745268	17985611	1536875	2781079	797354
g120074	0	0	0	0	0
g120075	225	0	0	0	0
g120076	2092	2236	1936	1776	564
g120077	7975	7588	5124	7434	3276
g120078	8604	9486	4374	6246	1956
g120079	11936	13522	9809	10599	7342
g120080	22560	35600	23200	22720	21360
g120081	28923	33279	24012	24963	24135
g120082	42260	43116	34178	35882	33522
g120083	50112	55924	43889	46045	45136

Michał Przewoźniczek - dotychczasowe publikacje

Czasopisma z listy filadelfijskiej:

H. Kwaśnicka, Przewoźniczek M., „MuPPetS – a new evolutionary method based on the idea of messy GA", *IEEE Transactions on evolutionary computation*, No. 5, Vol. 15, 2011, pp. 715-734

M. Przewoźniczek, K. Walkowiak, "Modeling and optimization of survivable P2P multicasting", *Computer Communications*, Vol. 34, Issue 8, Elsevier, 2011, s. 921-1042

M. Przewoźniczek, K. Walkowiak, M. Woźniak, "Optimizing distributed computing systems for k-nearest neighbors classifiers - evolutionary approach", *Logic Journal of the IGPL*, Vol. 19, Issue 2 , 2011, pp. 357-372

K. Walkowiak, Przewoźniczek M., K. Pająk, „ Heuristic Algorithms for Survivable P2P Multicasting", *Applied Artificial Intelligence* (praca w trakcie recenzji)

Czasopisma spoza listy filadelfijskiej:

M. Przewoźniczek, K. Walkowiak, "Quasi-hierarchical Evolution Algorithm for Flow Assignment in Survivable Connection-Oriented Networks", *International Journal of Applied Mathematics and Computer Science*, No. 4 Vol. 16, 2006, pp. 101-116

Pozostałe publikacje:

M. Przewoźniczek, K. Walkowiak, "Quasi-hierarchical Evolutionary Algorithm for Flow Optimization in Survivable MPLS Networks", *Lecture Notes in Computer Science*, Vol. 4707, Springer Verlag, 2007, pp. 330-342

M. Przewoźniczek, K. Walkowiak, "Evolutionary Algorithm for Congestion Problem in Connection-Oriented Networks", *Lecture Notes in Computer Science*, Vol. 3483, Springer Verlag, 2005, pp. 802-811