POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

Imię i nazwisko autora rozprawy: Waldemar Korłub
Dyscyplina naukowa: informatyka

ROZPRAWA DOKTORSKA

Tytuł rozprawy w języku polskim:
Optymalizacja strategii sieci inteligentnych agentów za pomocą programowania genetycznego w systemie rozproszonym realizującym paradygmat volunteer computing

Tytuł rozprawy w języku angielskim:
Optimizing strategies for a network of intelligent agents by means of genetic programming in a distributed volunteer computing system

Promotor *podpis*	Drugi promotor --- *podpis*
dr hab. inż. Jerzy Balicki, prof. nadzw. PW	---
Promotor pomocniczy --- *podpis*	Kopromotor --- *podpis*
---	---

Gdańsk, rok 2016

Spis treści

Wykaz ważniejszych oznaczeń **4**

Wprowadzenie **7**

1 Przegląd algorytmów ewolucyjnych ze szczególnym uwzględnieniem programowania genetycznego **10**

1.1 Taksonomia algorytmów ewolucyjnych 10

1.2 Programowanie ewolucyjne 12

1.3 Algorytmy genetyczne 13

1.4 Strategie ewolucyjne 16

1.5 Programowanie genetyczne 18

1.6 Ujednolicone podejście do przetwarzania ewolucyjnego 22

1.7 Ewolucyjna optymalizacja wielokryterialna 24

1.8 Algorytm harmoniczny 29

1.9 Wnioski i uwagi 32

2 Modele inteligentnych agentów **34**

2.1 Klasyfikacja środowisk agentowych 34

2.2 Wewnętrzne architektury agentów 36

2.3 Wybrane aspekty współpracy między agentami 40

2.4 Podstawowe założenia modelu środowiska 42

2.5 Zastosowania systemów agentowych 45

2.6 Wnioski i uwagi 48

3 Modelowanie agentowego systemu rozproszonego **50**

3.1 Charakterystyka wybranych gridów 50

3.2 Charakterystyka gridu *Comcute* 55

3.3 Eksperymentalne wyznaczenie wartości podstawowych parametrów agentów w gridzie *Comcute* 61
3.4 Model współpracy agentów w gridzie *Comcute* 73
3.5 Wnioski i uwagi 74

4 Problemy optymalizacji strategii zespołu inteligentnych agentów w środowisku wirtualnym 76
4.1 Zbiór strategii dopuszczalnych w podstawowym modelu gridu 76
4.2 Ocena jakości strategii zespołu agentów 85
4.3 Agenty-solwery do jednokryterialnej optymalizacji strategii zespołu agentów warstwy pośredniczącej 92
4.4 Wielokryterialna optymalizacja strategii zespołu agentów 97
4.5 Wnioski i uwagi 101

5 Metody wyznaczania optymalnych strategii sieci inteligentnych agentów 103
5.1 Programowanie genetyczne do optymalizacji jednokryterialnej 103
5.2 Wielokryterialne warianty programowania genetycznego 119
5.3 Kompromisowe wersje programowania genetycznego 130
5.4 Optymalizacja w dynamicznym środowisku 139
5.5 Zastosowanie modelu agentowego do wspomagania wykonania zadań obliczeniowych w gridzie *Comcute* 148
5.6 Wnioski i uwagi 164

Podsumowanie 167

Bibliografia 170

Wykaz rysunków 193

Wykaz algorytmów 196

Wykaz tabel 197

Dodatek A. Porównanie programowania genetycznego z innymi algorytmami 198

Dodatek B. Aplikacja Administratora Gridu AAG'16 205

Wykaz ważniejszych oznaczeń

α_v – agent nr v w systemie rozproszonym, $v = \overline{1,V}$,

β_j – j-ty rodzaj komputera do wykorzystania w gridzie, $j = \overline{1,J}$,

$\varepsilon_{\max}$ – maksymalny dopuszczalny pobór mocy elektrycznej przez komputery gridu [W],

η – stopień rozproszenia agentów w systemie,

$\eta_{\min}$ – minimalny wymagany stopień rozproszenia agentów w systemie,

ϑ_j – wydajność węzła typu β_j, zazwyczaj wyrażona w [FLOPS, (ang. *FLoating point Operations Per Second*)], $j = \overline{1,J}$,

$\vartheta_{\min}$ – minimalna wymagana wydajność systemu, zazwyczaj wyrażona w [FLOPS],

κ_i^{HDD} – rezerwa pamięci dyskowej HDD (ang. *Hard Disk Drive*) i-tego węzła [TB], $i = \overline{1,I}$,

κ_i^{RAM} – rezerwa pamięci operacyjnej RAM (ang. *Random-Access Memory*) i-tego węzła [GB], $i = \overline{1,I}$,

κ_i^{SSD} – rezerwa pamięci na dysku półprzewodnikowym SSD (ang. *Solid-State Drive*) i-tego węzła [TB], $i = \overline{1,I}$,

$\mu = [\mu_1, \ldots, \mu_k, \ldots, \mu_{19}]$ – wektor selektorów wymagań jakościowych gridu,

ν_j – liczba zainstalowanych komputerów j-tego rodzaju, $j = \overline{1,J}$,

$\hat{\nu}_j$ – limit liczby zainstalowanych komputerów j-tego rodzaju, $j = \overline{1,J}$,

$\xi = [\xi_1, \ldots, \xi_j, \ldots, \xi_J]$ – wektor kosztów zakupu komputerów różnego rodzaju [JM, JM – jednostka monetarna],

$\xi_{\max}$ – maksymalny dopuszczalny kosz zakupu komputerów [JM],

$\rho^{\leqslant}$ – relacja dominowania w zbiorze ocen $\boldsymbol{\mathcal{Y}}$,

$\tau = [\tau_{vu}]_{V \times V}$ – macierz skumulowanych czasów komunikacji pomiędzy agentami, elementy macierzy wyrażone w [s],

χ – program do wyznaczania strategii sieci agentów,

Γ – dostępność systemu,

Θ – łączna moc obliczeniowa gridu, zazwyczaj w [FLOPS],

Ξ – koszt zakupu komputerów [JM],

$\Omega = \{\omega_1, \ldots, \omega_i, \ldots, \omega_I\}$ – zbiór węzłów, w których rozmieszczone są agenty,

$h_j^{\min}$ – wielkość pamięci dyskowej HDD zajętej przez oprogramowanie systemowe na komputerze j-tego typu [TB], $j = \overline{1,J}$,

$\hat{h}_j^{\min}$ – wielkość pamięci na dysku półprzewodnikowym SSD zajętej przez oprogramowanie systemowe na komputerze j-tego typu [TB], $j = \overline{1,J}$,

h_v – wielkość pamięci dyskowej HDD zajmowanej przez v-tego agenta ze zbioru $\mathscr{A}$ [TB], $v = \overline{1,V}$,
$\hat{h}_v$ – wielkość pamięci na dysku półprzewodnikowym SSD zajmowanej przez v-tego agenta ze zbioru $\mathscr{A}$ [TB], $v = \overline{1,V}$,
hdd_j – wielkość pamięci dyskowej HDD komputera j-tego typu [TB], $j = \overline{1,J}$,
$r_j^{\min}$ – wielkość pamięci operacyjnej RAM zajętej przez oprogramowanie systemowe na komputerze j-tego typu [GB], $j = \overline{1,J}$,
r_v – wielkość pamięci operacyjnej RAM zajmowanej przez v-tego agenta ze zbioru $\mathscr{A}$ [GB], $v = \overline{1,V}$,
ram_j – wielkość pamięci operacyjnej RAM komputera j-tego typu [GB], $j = \overline{1,J}$,
ssd_j – wielkość pamięci na dysku półprzewodnikowym SSD komputera j-tego typu [TB], $j = \overline{1,J}$,
x – macierz opisująca strategię zespołu agentów w gridzie,
$z_1, \ldots, z_m, \ldots, z_M$ – zadania obliczeniowe,
$\mathscr{A} = \{\alpha_1, \ldots, \alpha_{V_W}, \alpha_{V_W+1}, \ldots, \alpha_V\}$ – zbiór agentów klasy W i S w gridzie *Comcute*,
$\mathscr{A}_S = \{\alpha_{V_W+1}, \ldots, \alpha_V\}$ – zbiór agentów klasy S w gridzie *Comcute*,
$\mathscr{A}_W = \{\alpha_1, \ldots, \alpha_{V_W}\}$ – zbiór agentów klasy W w gridzie *Comcute*,
$\mathscr{B} = \{\beta_1, \ldots, \beta_j, \ldots, \beta_J\}$ – zbiór dostępnych rodzajów komputerów,
E – łączny pobór mocy elektrycznej przez komputery gridu [W],
F – kryterium wektorowe oceny jakości strategii zespołu agentów,
F_n – kryterium cząstkowe oceny jakości strategii zespołu agentów,
$\boldsymbol{\mathcal{F}}$ – zbiór procedur wykorzystywanych w programowaniu genetycznym,
I – liczba węzłów, w których rozmieszczane są agenty,
$\widehat{I}$ – liczba węzłów obliczeniowych, które przetwarzają paczki danych,
$\boldsymbol{\mathcal{I}}$ – procedura *interpretacja* stosowana w programowaniu genetycznym,
J – liczba dostępnych rodzajów komputerów,
$\mathcal{J} = \{1, \ldots, j, \ldots, J\}$ – zbiór indeksów dostępnych rodzajów komputerów,
K – koszt wykonania zadań [JM],
K_{gr} – maksymalny dopuszczalny koszt wykonania zadań [JM],
$L(z_m)$ – liczba paczek danych dla zadania z_m, $m = \overline{1,M}$,
$L_S^{\alpha_v}(z_m)$ – liczba paczek danych dla zadania z_m przypadających na agenta α_v typu S, $m = \overline{1,M}$, $v = \overline{V_W+1,V}$,
$L_W(z_m)$ – liczba paczek danych dla zadania z_m przypadających na agenta typu W, $m = \overline{1,M}$,
M – liczba zadań obliczeniowych,
P_k – funkcja kary dla k-tego wymagania jakościowego, $k = \overline{1,19}$,
$P_R(z_m)$ – stopień replikacji danych zadania z_m, $m = \overline{1,M}$,
$P_W(z_m)$ – parametr wydajnościowy zadania z_m, $m = \overline{1,M}$,
$P_T(z_m)$ – czas przetwarzania pojedynczej paczki danych zadania z_m [s], $m = \overline{1,M}$,
S – rodzaj agenta dystrybucyjnego w gridzie *Comcute*,
$\mathcal{S}_m$ – zbiór agentów typu S realizujących zadanie z_m, $m = \overline{1,M}$,
$T = [t_{vj}]_{V \times J}$ – macierz skumulowanych czasów pracy agentów, elementy macierzy wyrażone w [s],
$\boldsymbol{\mathcal{T}}$ – zbiór symboli terminalnych wykorzystywanych w programowaniu genetycznym,
V – liczba agentów programistycznych,
W – rodzaj agenta zarządzającego w gridzie *Comcute*,

$\mathcal{W}_m$ – zbiór agentów typu W realizujących zadanie z_m, $m = \overline{1,M}$,
X – wektor opisujący strategię zespołu agentów,
$X^z = [X_1^z, \ldots, X_m^z, \ldots, X_M^z]^T$ – wektor przydziału zadań do agentów zarządzających,
$X^\alpha = [X_1^\alpha, \ldots, X_v^\alpha, \ldots, X_V^\alpha]^T$ – wektor przydziału agentów do węzłów,
$X^\beta = [X_1^\beta, \ldots, X_i^\beta, \ldots, X_I^\beta]^T$ – wektor przydziału typów komputerów do węzłów,
$\boldsymbol{\mathcal{X}}$ – zbiór możliwych strategii zespołu agentów w gridzie,
$\boldsymbol{\mathcal{Y}}$ – zbiór ocen strategii dopuszczalnych,
$\hat{Z}_{\mathrm{gr}}$ – maksymalne dopuszczalne obciążenie procesorów newralgicznego węzła [s],
$\hat{Z}_i$ – obciążenie procesorów i-tego węzła [s], $i = \overline{1,I}$,
$\hat{Z}_{\max}$ – obciążenie procesorów newralgicznego węzła [s],
$\hat{Z}_{\mathrm{suma}}$ – sumaryczne obciążenie procesorów wszystkich węzłów gridu [s],
$\hat{Z}_{\mathrm{Sgr}}$ – maksymalne dopuszczalne sumaryczne obciążenie procesorów systemu [s],
$\widetilde{Z}_{\mathrm{gr}}$ – maksymalne dopuszczalne sumaryczne obciążenie komunikacyjne newralgicznego węzła [s],
$\widetilde{Z}_i$ – obciążenie komunikacyjne i-tego węzła [s], $i = \overline{1,I}$,
$\widetilde{Z}_{\max}$ – obciążenie komunikacyjne newralgicznego węzła [s],
$\widetilde{Z}_{\mathrm{suma}}$ – sumaryczne obciążenie komunikacyjne węzłów gridu [s],
$\widetilde{Z}_{\mathrm{Sgr}}$ – maksymalne dopuszczalne obciążenie komunikacyjne systemu [s],
$\boldsymbol{\mathcal{Z}}(F_n, \mu)$ – zagadnienie optymalizacji jednokryterialnej w odniesieniu do strategii zespołu inteligentnych agentów,
$\boldsymbol{\mathcal{Z}}(F, \mu, \rho^{\leqslant})$ – zagadnienie optymalizacji wielokryterialnej w odniesieniu do strategii zespołu inteligentnych agentów.

Wprowadzenie

Kluczową rolę w systemach rozproszonych odgrywa warstwa pośrednicząca (ang. *middleware*), która odpowiada za nadzorowanie dostępnych zasobów. Odpowiednie zarządzanie zasobami sprzętowymi i programistycznymi pozwala na skrócenie czasu realizacji zadań, minimalizację kosztów czy zagwarantowanie oczekiwanego poziomu niezawodności systemu. Jest to szczególnie ważne we współczesnych zastosowaniach systemów rozproszonych obejmujących obszary, takie jak ekstrakcja wiedzy z rozległych zbiorów danych (ang. *Big Data*), inżynieria danych w różnorodnych systemach biznesowych (ang. *Data Science*) czy rozproszone metody sztucznej inteligencji.

Rosnąca skala i złożoność systemów rozproszonych utrudnia administrowanie infrastrukturą informatyczną. W koncepcji przetwarzania autonomicznego (ang. *autonomic computing*) zaproponowanej przez firmę *IBM* system powinien być zdolny do samodzielnej konfiguracji, optymalizacji i obrony przez atakami, a także samodzielnego wykrywania i naprawy awarii. Cele te realizowane są za pomocą autonomicznych komponentów zarządzających w warstwie pośredniczącej systemu. Prace w zakresie systemów tej klasy prowadzą również inni liderzy branży informatycznej, jak *Microsoft*, czy *Hewlett-Packard*.

W szczególności ważne są badania nad wykorzystaniem agentowo zorientowanej warstwy pośredniczącej w obliczeniach. Agenty programistyczne posiadają cechy, które odpowiadają na wymagania autonomicznych systemów rozproszonych. Zdolność agentów do reagowania na zmiany w systemie pozwala na obsługę awarii, a także na przetwarzanie dynamicznych strumieni danych. Mobilność umożliwia takie ulokowanie agentów zarządzających, które minimalizuje narzuty komunikacyjne w systemie. Z kolei autonomia i zdolność do proaktywnych zachowań pozwalają na decentralizację mechanizmów zarządzania i minimalizację negatywnego wpływu niedostępności wybranych węzłów na stabilność całego systemu. Dodatkowo agenty mogą służyć jako warstwa integracji pomiędzy różnymi systemami.

Zachowania agentów warstwy pośredniczącej wspierają strategię zarządzania systemem rozproszonym. Określa ona ulokowanie agentów, sposób przydzielania do nich zasobów programistycznych w postaci zadań do wykonania oraz przypisanie zasobów sprzętowych do węzłów systemu. Strategia zespołu agentów warstwy pośredniczącej może być zadana przez administratorów lub zaimplementowana przez programistów. Wiąże się to jednak z trudnością uwzględnienia wielu aspektów działania systemu rozproszonego, które są istotne z punktu widzenia interesariusza. Problem wyznaczenia efektywnej strategii zespołu agentów warstwy pośredniczącej można sformułować w postaci zagadnienia optymalizacji wielokryterialnej.

Niezwykle ważne i aktualne badania dotyczą możliwości zastosowania programowania genetycznego w odniesieniu do wielokryterialnego problemu wyznaczania strategii zespołów agentów zarządzających systemem rozproszonym. Istotne są modele pozwalające na opisywanie strategii agentów oraz metody ich wykorzystania w kontekście ewolucyjnych metaheurystyk optymalizacji ze szczególnym uwzględnieniem programowania genetycznego.

Ważnym środowiskiem, w którym można wykorzystać inteligentne agenty sterowane strategiami opracowanymi w sposób ewolucyjny, jest system obliczeniowy klasy grid realizujący paradygmat *volunteer computing*. W tym wypadku praca węzłów systemu może być kontrolowana przez agenty, które zespołowo dążą do osiągnięcia wspólnego celu – realizacji zleconych zadań obliczeniowych przy optymalizacji wybranych kryteriów i zachowaniu nałożonych ograniczeń. W przypadku tej klasy systemów rozważa się kryteria optymalizacji, takie jak: minimalizacja łącznego czasu potrzebnego do uzyskania wyników, minimalizacja obciążenia newralgicznego węzła, minimalizacja kosztów zakupu komputerów w gridzie, maksymalizacja łącznej wydajności systemu, czy maksymalizacja jego dostępności. Warto podkreslić, że w literaturze przedmiotu nie podejmowano do tej pory tak obszernie problematyki wyznaczania i automatycznej implementacji wielokryterialnej strategii sieci agentów w systemach rozproszonych. W tym kontekście niniejsza rozprawa powinna wypełnić choćby w minimalnym stopniu istniejącą lukę.

W ramach przyjętej koncepcji metodologicznej celem rozprawy jest opracowanie wielokryterialnej metody programowania genetycznego pozwalającej na optymalizację strategii zespołu agentów w systemie rozproszonym realizującym paradygmat *volunteer computing*. Rozszerzenie agentowej warstwy pośredniczącej o strategię uzyskaną za pomocą zaawansowanych algorytmów sztucznej inteligencji umożliwia efektywną pracę systemu w zadanych scenariuszach realizacji zadań obliczeniowych. Celem pomocniczym jest zbadanie możliwości usprawnienia funkcjonowania środowiska obliczeń rozproszonych *Comcute*, które zaprojektowano w 2012 roku i jest systematycznie rozwijane na Wydziale Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej.

W rozprawie sformułowano problem badawczy w postaci zagadnienia optymalizacji wielokryterialnej, dla którego wyznacza się reprezentację *Pareto*-optymalnych strategii sieci inteligentnych agentów dla wybranych kryteriów i ograniczeń w systemie rozproszonym realizującym paradygmat *volunteer computing*. Przedstawicielem tej klasy systemów jest wspomniany grid *Comcute*.

Na podstawie celu pracy i problemu badawczego postawiono następującą hipotezę badawczą: „Programowanie genetyczne w odniesieniu do sformułowanego problemu polioptymalizacji systemu rozproszonego realizującego paradygmat *volunteer computing* umożliwia wyznaczenie *Pareto*-optymalnych strategii realizowanych przez sieć agentów warstwy pośredniczącej."

Do osiągnięcia celu, rozwiązania problemu badawczego i weryfikacji hipotezy wykorzystano specyficzne metody badawcze dla informatyki, w tym modelowanie i algorytmizację, optymalizację wielokryterialną oraz eksperymenty numeryczne. Wynikami z przeprowadzonych prac są modele, zagadnienia optymalizacji i algorytmy programowania genetycznego pozwalające na wyznaczanie strategii dla zespołów agentów w środowisku systemu rozproszonego *Comcute*. Opracowane modele i algorytmy zaimplementowano, co umożliwia formułowanie nowych zagadnień optymalizacji

poprzez interaktywny wybór kryteriów i ograniczeń. Pozwoliło to na wykazanie uniwersalności i elastyczności opracowanych metod w rozwiązywaniu różnorodnych problemów polioptymalizacji.

Rozprawa składa się z pięciu rozdziałów. W rozdziale pierwszym scharakteryzowano algorytmy ewolucyjne ze szczególnym uwzględnieniem programowania genetycznego. Omówiono wybrane podejścia do przetwarzania ewolucyjnego. Przedstawiono ewolucyjne metody optymalizacji wielokryterialnej oraz algorytm harmoniczny jako przykład relatywnie nowej metaheurystyki ewolucyjnej do wyznaczania rozwiązań *Pareto*-optymalnych.

Rozdział drugi stanowi omówienie systemów agentowych. Przedstawiono klasyfikację środowisk agentowych i podstawowe założenia informatyczne w celu ich praktycznej realizacji. Scharakteryzowano również wewnętrzne architektury inteligentnych agentów i wybrane aspekty współpracy między agentami. Ponadto omówiono zastosowania inteligentnych agentów do optymalizacji środowisk rozproszonych.

W rozdziale trzecim scharakteryzowano model środowiska agentów kierujących pracą systemu *Comcute*. Omówiono wybrane systemy typu grid oraz platformę *Comcute* jako przykład gridu opartego o paradygmat wolontariatu obliczeniowego. Opisano kluczowe parametry systemu, które mają wpływ na przemieszczanie się jego agentów i przebieg obliczeń. Przedstawiono pomiary wykonane z użyciem wersji laboratoryjnej systemu, które pozwoliły na wyznaczenie podstawowych danych wejściowych do zagadnień optymalizacji.

Rozdział czwarty zawiera sformułowane problemy optymalizacji strategii sieci agentów w środowisku grida obliczeniowego. Wprowadzono ograniczenia formalne określające zbiór rozwiązań dopuszczalnych dla rozpatrywanych problemów. Zdefiniowano kryteria oceny jakości strategii zespołu agentów i ograniczenia istotne z punktu widzenia interesariuszy. Scharakteryzowano *agenty-solwery* do jednokryterialnej oraz wielokryterialnej optymalizacji strategii zespołu agentów warstwy pośredniczącej.

W rozdziale piątym omówiono metody wyznaczania strategii zespołu agentów w gridzie z użyciem programowania genetycznego dla sformułowanych problemów optymalizacji. Przedstawiono sposób wyznaczania sprawności rozwiązań w oparciu o wybrane funkcje celu i funkcje kary zdefiniowane dla każdego z ograniczeń. Omówiono warianty programowania genetycznego do wyznaczania rozwiązań *Pareto*-optymalnych dla wybranego zestawu kryteriów. Przedstawiono również metody selekcji rozwiązań kompromisowych. Scharakteryzowano wykorzystanie programowania genetycznego do implementacji programów wyznaczających strategię zespołu agentów w zależności od stanu systemu. Zamieszczono także wyniki wybranych eksperymentów numerycznych.

Po podsumowaniu i specyfikacji bibliografii, zamieszczono wykazy rysunków i tabel, a także dodatki, w których omówiono wyniki eksperymentów numerycznych z wykorzystaniem programowania genetycznego dla wybranych instancji z literatury przedmiotu. Ponadto, scharakteryzowano aplikację do wspomagania administratora gridu AAG'16.

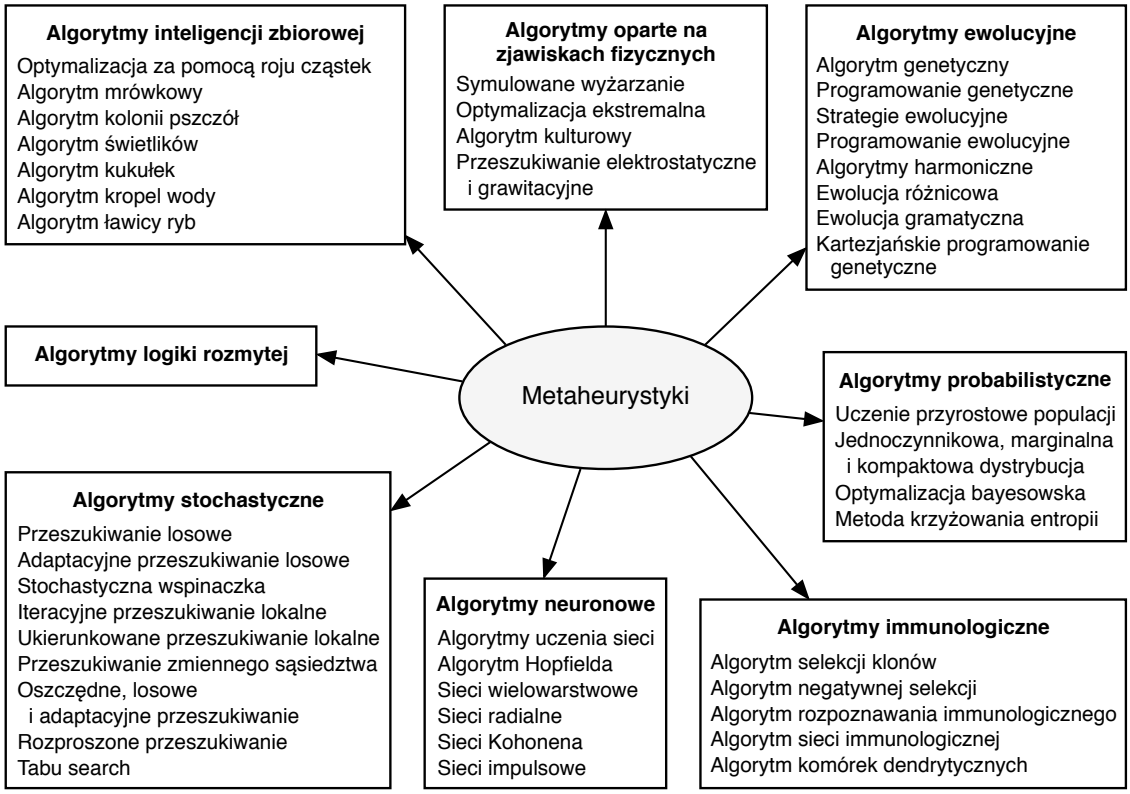
Rozdział 1

Przegląd algorytmów ewolucyjnych ze szczególnym uwzględnieniem programowania genetycznego

1.1 Taksonomia algorytmów ewolucyjnych

Algorytmy ewolucyjne należą do klasy probabilistycznych metod optymalizacji [72]. W początkowym okresie badań nad wykorzystaniem modelu ewolucji do uczenia maszynowego i rozwiązywania problemów optymalizacji wykształciły się trzy podstawowe nurty rozwijane przez niezależne zespoły: *programowanie ewolucyjne* (*Fogel et al.*, 1966 [81]), *algorytmy genetyczne* (*Holland*, 1975 [120]) i *strategie ewolucyjne* (*Rechenberg*, 1973 [226] oraz *Schwefel*, 1975 [236]). *Programowanie genetyczne* – jako czwarty istotny nurt – zostało sformułowane zgodnie ze współczesnym jego rozumieniem w latach 80. ubiegłego wieku przez *Cramera* [59], a nastepnie rozwinęło się w latach 90. za sprawą *Kozy* [162]. Mniej zaawansowane prace związane z metodami automatycznej implementacji programów ukazywały się już w latach 70. [185].

Ze względu na sposób działania wyróżnić można osiem klas metaheurystyk, co pokazano na rysunku 1.1 [184]. Algorytmy ewolucyjne są metaheurystykami opartymi na populacji rozwiązań. Dywersyfikacja i dostrajanie osobników odbywa się przy użyciu operacji genetycznych, takich jak krzyżowanie i mutacja. Algorytmy ewolucyjne zalicza się do metaheurystyk z pamięcią rozwiązań, przy czym rolę pamięci pełni populacja osobników [72]. Na rys. 1.1 w klasie algorytmów ewolucyjnych wyróżniono cztery główne podejścia wymienione powyżej, a także algorytmy harmoniczne [100] oraz inne ciekawe metaheurystyki, takie jak: ewolucja różnicowa, ewolucja gramatyczna, czy kartezjańskie programowanie genetyczne.



Rysunek 1.1: Podstawowe rodzaje metaheurystyk [184]

W algorytmach ewolucyjnych osobniki w populacji reprezentują punkty w przestrzeni poszukiwań. W kolejnych epokach każdemu osobnikowi przypisuje się wartość dopasowania, która określa jakość uzyskanego rozwiązania. Dopasowanie jest kluczowym kryterium na etapie selekcji, która ukierunkowuje poszukiwanie w stronę rozwiązań o pożądanych ocenach. Mutacja umożliwia utrzymanie różnorodności w populacji i rozszerza zakres poszukiwań na nowe obszary. Z kolei krzyżowanie pozwala na dostrojenie rozwiązań poprzez wymianę cech osobników. Modyfikacje osobników odbywają się w oparciu o ich genotyp. Istotnym elementem algorytmu ewolucyjnego jest odwzorowanie genotyp-fenotyp, które na podstawie genotypu zwraca rozwiązanie możliwe do zastosowania w dziedzinie problemu. Połączenie powyższych mechanizmów pozwala na wyznaczanie wysokiej jakości rozwiązań w wielu problemach optymalizacji [72].

Poszczególne odmiany algorytmów ewolucyjnych są ukierunkowane na różne obszary zastosowań, co ma wpływ na sposób reprezentacji rozwiązań, wykorzystywane struktury danych, operacje genetyczne i wprowadzane do algorytmów usprawnienia. Dostępnych jest wiele specjalistycznych implementacji algorytmów tej klasy, które łączą zalety różnych podejść w celu uzyskania najlepszych rezultatów w rozpatrywanych problemach optymalizacji [34]. Ze względu na to zaklasyfikowanie wybranego algorytmu do jednej kategorii jest często trudne [38].

1.2 Programowanie ewolucyjne

Programowanie ewolucyjne zostało oryginalnie zaproponowane jako metoda doboru parametrów dla automatów skończonych. Następnie zastosowano je w odniesieniu do wybranych zagadnień optymalizacji. Zazwyczaj wykorzystywana jest intuicyjna reprezentacja osobników w postaci zbioru parametrów problemu, więc nie jest konieczne mapowanie genotyp-fenotyp [21]. Jedyną operacją stosowaną do konstruowania nowych osobników jest mutacja, której poddawane są wszystkie rozwiązania w danej epoce. W przypadku populacji o rozmiarze μ otrzymujemy zatem μ nowych osobników. Konieczne jest przeprowadzenie selekcji w każdej epoce dla utrzymania stałego rozmiaru populacji.

Ponieważ w tym podejściu nie jest wykorzystywany mechanizm kojarzenia osobników, jedyną formą ich doboru jest selekcja środowiskowa. Wybór osobników realizowany jest w sposób niedeterministyczny za pomocą selekcji turniejowej. W celu wyłonienia osobnika, wybieranych jest p_t losowych elementów z populacji. Wylosowane osobniki są porównywane w oparciu o wartości przypisane im za pomocą funkcji dopasowania. Element cechujący się najlepszym dopasowaniem staje się zwycięzcą turnieju i przechodzi do kolejnej epoki. Powyższa procedura jest powtarzana do momentu wyłonienia μ osobników [198].

Rozmiar turnieju p_t jest parametrem pozwalającym na sterowanie presją selekcyjną. Większe wartości parametru p_t przekładają się na większą presję, gdyż osobniki o niższej jakości mają mniejszą szansę na wygranie turnieju. Duża presja selekcyjna przekłada się na szybką zbieżność populacji do najlepszych jakościowo osobników. Z drugiej strony oznacza to, że w dużym tempie zmniejsza się różnorodność rozwiązań, co może uniemożliwić wyjście poza ekstrema lokalne funkcji dopasowania [17].

Schemat działania programowania ewolucyjnego zaprezentowano w postaci algorytmu nr 1.1. Populacja początkowa składa się zazwyczaj z osobników wygenerowanych w sposób losowy (linia 1). Jeśli istnieją przesłanki wskazujące, że pewne obszary w przestrzeni rozwiązań są szczególnie warte eksploracji, to punkty z takich obszarów mogą zostać dołączone do populacji. Pozwala to na ukierunkowanie ewolucji na wybrane fragmenty domeny rozwiązań. Przesłanki takie mogą powstać w oparciu o wyniki z programowania ewolucyjnego lub na podstawie rezultatów z innego algorytmu optymalizacji wykorzystanego w celu redukcji przestrzeni poszukiwań [34].

Następnie osobniki poddawane są ocenie (linia 2), a najlepszy z nich jest zapamiętywany (linia 3). Główna pętla algorytmu zaczyna się w linii 4. Dopóki nie zostanie spełniony warunek zakończenia, wykonywane są kolejne iteracje odpowiadające następującym po sobie epokom. Każdy element populacji poddawany jest mutacji (linie 5-9), czego efektem jest zbiór osobników potomnych. Skonstruowane elementy poddawane są ocenie (linia 10). Następnie najlepsze rozwiązanie wybierane jest spośród wszystkich dotychczasowych osobników (linia 11). Ostatnim etapem jest selekcja środowiskowa pozwalająca na wybranie μ osobników do kolejnej epoki (linie 12-13). Po zakończeniu głównej pętli algorytmu zwracane jest najlepsze rozwiązanie uzyskane podczas wszystkich epok (linia 15).

Algorytm 1.1 Diagram programowania ewolucyjnego

```
1:  populacja := inicjalizacjaPopulacji(rozmiarPopulacji: μ);
2:  ocenaOsobnikow(populacja);
3:  najlepszeRozwiazanie := wybierzNajlepsze(populacja);
4:  while warunekZakonczeniaEwolucji == false do
5:      potomkowie := ∅;
6:      for rodzic_i ∈ populacja do
7:          potomek_i := mutacja(rodzic_i);
8:          potomkowie := potomkowie ∪ {potomek_i};
9:      end for
10:     ocenaOsobnikow(potomkowie);
11:     najlepszeRozwiazanie := wybierzNajlepsze(potomkowie, najlepszeRozwiazanie);
12:     populacja := populacja ∪ potomkowie;
13:     populacja := selekcjaSrodowiskowa(populacja);
14: end while
15: return najlepszeRozwiazanie;
```

Źródło: opracowanie własne na podstawie [34]

Warunek zakończenia ewolucji (linia 4) odnosi się do wybranych aspektów działania algorytmu. Maksymalna liczba epok pozwala na arbitralne ograniczenie czasu działania symulacji. Inne kryterium stopu związane jest z różnorodnością osobników. Jeśli jest ona zbyt mała, populacja traci zdolność do eksploracji przestrzeni możliwych rozwiązań. Zamiast tego następuje eksploatacja obszarów już częściowo przeszukanych w celu odnalezienia lokalnych optimów. Zbyt mała różnorodność rozwiązań może powodować przedwczesne zakończenie algorytmu [51].

Kryterium stopu może odnosić się również do założonej wartości funkcji celu. Niech optymalizacji podlega koszt pracy serwera obliczeniowego w chmurze. Jeśli serwer jest opłacany za każdą godzinę dzierżawy „z góry", to rozwiązania cechujące się czasem pracy nie większym niż 60 minut są równoważne z punktu widzenia kryterium optymalizacji. Zatem algorytm może zostać zatrzymany, gdy osiągnięto wartość funkcji celu na poziomie kosztów godzinnej dzierżawy. W wielu zastosowaniach warunek zakończenia jest połączeniem wszystkich wymienionych powyżej kryteriów.

1.3 Algorytmy genetyczne

Algorytmy genetyczne zaproponowano do rozwiązywania różnorodnych problemów kombinatorycznych [120]. Charakterystycznym elementem tego podejścia jest wykorzystanie łańcuchów binarnych do reprezentacji osobników, co nawiązuje do łańcuchów DNA organizmów żywych. Operacje genetyczne na elementach populacji są zdefiniowane w kontekście ciągów bitów i nie uwzględniają cech fenotypowych. Wyznaczenie rozwiązania, które może zostać wykorzystane w dziedzinie problemu, wymaga mapowania genotyp-fenotyp. Na podstawie binarnych łańcuchów zwraca ono reprezentacje parametrów liczbowych dla rozpatrywanego zagadnienia optymalizacji.

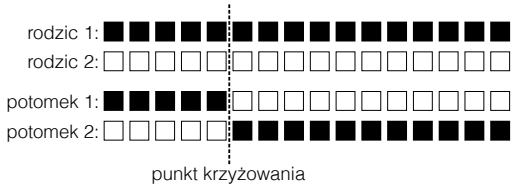
Warto zauważyć, że ten sam ciąg bitów może zostać zinterpretowany na wiele różnych sposobów. Przykładowo, 16-bitowy ciąg: „0011010101010101" może być zapisem liczby całkowitej bez znaku: 13653, liczby zmiennoprzecinkowej: 0,33325 $\approx 1/3$ lub pary 8-bitowych liczb całkowitych: 53

i 85. Mapowanie genotyp-fenotyp jest konieczne do interpretacji wyników algorytmu genetycznego w kategoriach dziedziny rozwiązywanego problemu [120].

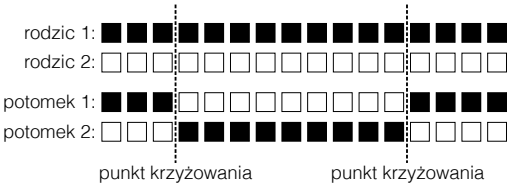
Mutacja osobników jest zazwyczaj realizowana jako zmiana wartości losowo wybranych bitów rozwiązania. Pozycje modyfikowanych bitów wybierane są w sposób losowy, a liczba wykonywanych zmian powinna zostać dobrana do długości reprezentacji. Mutacja wykonywana jest z określonym prawdopodobieństwem, które może podlegać zmianom w miarę upływu kolejnych epok [106].

Drugą operacją genetyczną w algorytmach tej klasy jest krzyżowanie. Jego celem jest redukcja przestrzeni poszukiwań i utrwalanie w populacji cech poprawiających jakość osobników. Krzyżowanie jest w opozycji do mutacji, która ze względu na losowy charakter rozszerza obszar poszukiwań. Kontrola relacji tych dwóch operacji pozwala na sterowanie przebiegiem algorytmu genetycznego. W początkowym etapie działania nacisk może zostać położony na identyfikację obszarów wokół optimów funkcji jakości i redukcję przestrzeni przeszukiwań do tych rejonów. Gdy obszary te zostaną zidentyfikowane, mutacja osobników pozwoli na poprawienie rozwiązań w ramach wyselekcjonowanych rejonów [34].

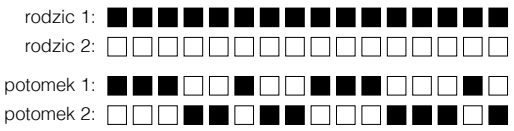
Krzyżowanie to operacja, w której dwa osobniki wymieniają się materiałem genetycznym. W efekcie powstają elementy potomne, które łączą cechy swoich rodziców. W algorytmach genetycznych krzyżowanie jest zdefiniowane w kontekście łańcuchów binarnych. W najprostszym podejściu wybierany jest pojedynczy punkt, względem którego następuje wymiana genów pomiędzy osobnikami [61]. Przykład takiego krzyżowania zobrazowano na rys. 1.2a.



(a) Krzyżowanie względem jednego punktu



(b) Krzyżowanie względem dwóch punktów



(c) Krzyżowanie jednorodne

Rysunek 1.2: Wybrane rodzaje krzyżowania w algorytmach genetycznych [61]

Rozszerzeniem opisanej powyżej metody jest krzyżowanie wykorzystujące dwa punkty (rys. 1.2b). Innym sposobem zapewnienia wymiany genów jest krzyżowanie jednorodne (ang. *uniform crossover*). W tym podejściu zamiast punktów przecięcia wybierany jest stosunek, w jakim geny rodziców zostaną połączone przy konstrukcji osobnika potomnego. Zazwyczaj wykorzystywany jest stosunek $1:1$, przy którym połowa genów pochodzi od pierwszego z rodziców, a pozostałe – od drugiego. Dla każdego bitu potomka wybierana jest w sposób losowy wartość znajdująca się na odpowiedniej pozycji w genotypie jednego z rodziców [61]. Ten rodzaj krzyżowania zobrazowano na rys. 1.2c.

Niezwykle istotny jest sposób doboru osobników, które są ze sobą krzyżowane. W algorytmach genetycznych występuje selekcja na etapie kojarzenia, która ma promować osobniki cechujące się wyższą jakością – ich geny są utrwalane w populacji za pomocą reprodukcji. Odróżnia to ten rodzaj doboru od selekcji środowiskowej, której celem jest utrzymanie stałego rozmiaru populacji w oparciu o zbiór osobników uzyskanych po wykonaniu operacji genetycznych [20].

Selekcja na etapie kojarzenia w algorytmach genetycznych ma zazwyczaj niedeterministyczny charakter. Prawdopodobieństwo p_i wyboru i-tego osobnika jest wprost proporcjonalne do jego sprawności, jak niżej [106]:

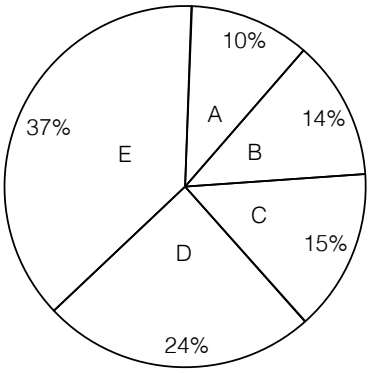
$$p_i = \frac{f(o_i)}{\sum_{j=1}^{\mu} f(o_j)}, \tag{1.1}$$

gdzie:
$O = \{o_1, \ldots, o_i, \ldots, o_\mu\}$ – zbiór osobników reprezentujący bieżącą populację,
$f : \mathcal{X} \to \mathbb{R}$ – funkcja sprawności, przy czym: $\mathcal{X}$ – przestrzeń przeszukiwań, $o_i \in \mathcal{X}$ dla $i = \overline{1,\mu}$,
$\mathbb{R}$ – zbiór liczb rzeczywistych.

Podstawowa metoda doboru osobników określana jest jako selekcja ruletkowa. Każdemu osobnikowi przypisać można wycinek koła ruletki o kącie $2\pi p_i$. Przykładową populację pięciu osobników o różnych ocenach z funkcji sprawności zaprezentowano na rys. 1.3a. Koło ruletki utworzone dla rozważanych osobników przedstawiono na rys. 1.3b.

Osobnik	$f(o_i)$	p_i
A	156,70	10%
B	219,38	14%
C	235,05	15%
D	376,08	24%
E	579,79	37%
Σ	1567	100%

(a) Oceny osobników



(b) Przykładowe koło ruletki

Rysunek 1.3: Selekcja ruletki
Źródło: opracowanie własne

Po zakończeniu etapów krzyżowania i mutacji następuje faza selekcji środowiskowej, która pozwala na utrzymanie stałego rozmiaru populacji. W przypadku algorytmów genetycznych wykorzystywana jest operacja zastąpienia osobników z poprzedniej epoki nową generacją rozwiązań bez uwzględniania jakości poszczególnych elementów populacji. Takie podejście niesie ze sobą ryzyko utraty najlepszych spośród uzyskanych dotychczas rozwiązań. Niemniej selekcja na etapie kojarzenia i nacisk na utrwalanie korzystnych cech w populacji redukują negatywny wpływ eliminacji niektórych rozwiązań [20].

Schemat omawianego podejścia przedstawiono w postaci algorytmu nr 1.2. W liniach 1-3 następuje inicjalizacja populacji, jej ocena i wybranie najlepszego osobnika. Następnie rozpoczyna

się główna pętla algorytmu (linia 4). Przed przystąpieniem do wykonywania operacji genetycznych przeprowadzana jest selekcja na etapie kojarzenia (selekcja godowa – linia 6). Gdy skonstruowany zostanie zbiór potencjalnych rodziców, rozpoczyna się ich krzyżowanie (linia 8). Uzyskane elementy potomne są poddawane mutacji (linie 9-10). Nowe osobniki są oceniane (linia 13), a najlepszy z nich – zapamiętany (linia 14). Ostatnim krokiem każdej iteracji jest zastąpienie poprzedniej populacji nową generacją rozwiązań (linia 15). Przed zwróceniem końcowego rozwiązania wykorzystuje się mapowanie genotyp-fenotyp (linia 17).

Algorytm 1.2 Diagram algorytmu genetycznego

```
1:  populacja := inicjalizacjaPopulacji(rozmiarPopulacji: μ);
2:  ocenaOsobnikow(populacja);
3:  najlepszeRozwiazanie := wybierzNajlepsze(populacja);
4:  while warunekZakonczeniaEwolucji == false do
5:      potomkowie := ∅;
6:      rodzice := selekcjaGodowa(populacja);
7:      for rodzic_i, rodzic_2 ∈ rodzice do
8:          (potomek_1, potomek_2) := krzyzowanie(rodzic_1, rodzic_2);
9:          potomek_1 := mutacja(potomek_1);
10:         potomek_2 := mutacja(potomek_2);
11:         potomkowie := potomkowie ∪ {potomek_1, potomek_2};
12:     end for
13:     ocenaOsobnikow(potomkowie);
14:     najlepszeRozwiazanie := wybierzNajlepsze(potomkowie, najlepszeRozwiazanie);
15:     populacja := zastapienie(populacja, potomkowie);
16: end while
17: return mapowanieGenotypFenotyp(najlepszeRozwiazanie);
```

Źródło: opracowanie własne na podstawie [34]

1.4 Strategie ewolucyjne

Strategie ewolucyjne zaproponowano jako metodę systematycznej analizy i udoskonalania projektów rozwiązań technicznych [39]. Jednym z oryginalnych zastosowań było projektowanie elementów pod kątem minimalizacji oporu powietrza. Nowe projekty powstawały na podstawie poprzednich, a ocena opracowanych rozwiązań odbywała się w tunelu aerodynamicznym. Jeśli w czasie eksperymentu nowy model uzyskał lepsze wyniki niż wcześniejszy, to zastępował go. W przeciwnym razie nowe rozwiązanie było odrzucane i zachodziła ponowna próba poprawienia poprzedniego projektu. Nakreśliło to podstawowy schemat strategii ewolucyjnych [39].

Strategie typu $(1+1)$ wykorzystują jednego osobnika, który poddany mutacji umożliwia skonstruowanie jednego potomka. Selekcja ma w tym przypadku charakter deterministyczny, ponieważ zawsze wybierany jest lepiej dopasowany osobnik. Rozwinięciem tej koncepcji są strategie $(\mu+1)$. Populacja składa się w tym przypadku z μ osobników. Nowe rozwiązanie powstaje na drodze rekombinacji cech dwóch losowo wybranych rodziców. Skonstruowany w ten sposób potomek dodatkowo poddawany jest mutacji. Aby utrzymać stały rozmiar populacji, najgorsze spośród $\mu+1$ rozwiązań jest odrzucane [34].

Szczególną rolę w przypadku strategii ewolucyjnych odgrywa mutacja osobników. W innych podejściach jest ona zazwyczaj mniej istotnym i losowym zaburzeniem rozwiązań. W tym przypadku mutacja jest podstawową operacją, która ma na celu przybliżenie populacji do punktu optymalnego. Analiza Rechenberga dla strategii $(1+1)$ pokazała, że pożądane tempo mutacji wynika z prawdopodobieństwa jej sukcesu, które powinno wynosić $1/5$ w celu uzyskania zbieżności do rozwiązania optymalnego [226].

W wypadku populacji o większym rozmiarze parametry mutacji stanowią dodatkową specyfikację osobników, obok zmiennych decyzyjnych z dziedziny problemu. Rozwiązania o wyższej jakości są utrwalane w populacji, a wraz z nimi również parametry mutacji, które doprowadziły do ich uzyskania. Dzięki temu w kolejnych epokach eksplorowane są te kierunki, które dotychczas pozwalały na poprawienie wyznaczonych rezultatów.

Strategia ewolucyjna powinna posiadać zdolność dostrajania parametrów mutacji bez zewnętrznej kontroli. Analiza strategii $(\mu+1)$ pokazała jednak, że cechuje się ona tendencją do redukowania tempa zmian w miarę upływu kolejnych epok, nawet jeśli prowadzi to do niepożądanej stagnacji [39]. Jako rozwiązanie tego problemu Schwefel zaproponował kolejne dwie odmiany strategii ewolucyjnych: $(\mu+\lambda)$ oraz (μ,λ). W ich przypadku w pojedynczej epoce generowany jest nie jeden, lecz λ osobników potomnych. Strategie różnią się sposobem wyboru rozwiązań, które powinny pozostać w populacji.

Zgodnie ze strategią $(\mu+\lambda)$ nowe osobniki są dodawane do zbioru dotychczasowych, a następnie odrzuca się λ rozwiązań o najniższej jakości. W przypadku strategii (μ,λ) wszystkie osobniki z poprzedniej epoki są odrzucane po zakończeniu etapu prokreacji, a wybór μ rozwiązań odbywa się jedynie spośród λ nowych osobników. Może to oznaczać utratę najlepszych spośród wyznaczonych dotychczas rozwiązań. Warunkiem koniecznym, lecz nie wystarczającym zbieżności tej metody jest przyrost na poziomie $\lambda/\mu > 1$. Jeśli warunek nie jest spełniony, nie występuje presja selekcyjna, której zadaniem jest ukierunkowanie ewolucji w stronę rozwiązań optymalnych. Graniczny przypadek $\lambda = \mu$ odpowiada przeszukiwaniu losowemu przestrzeni możliwych rozwiązań. Brak presji selekcyjnej sprawia, że każda mutacja jest uznawana za sukces, co uniemożliwia dostrojenie parametrów strategii [38].

Liczba osobników potomnych w przypadku strategii $(\mu+\lambda)$ nie jest ograniczona żadnymi warunkami. Ze względu na fakt, że selekcja środowiskowa ma charakter deterministyczny, posiada ona cechę elitaryzmu – najlepsze rozwiązania zawsze będą zachowane w populacji. Występowanie elitaryzmu jest warunkiem wystarczającym dla udowodnienia zbieżności strategii ewolucyjnej [38]. Przedstawione powyżej podejścia posiadają ważne obszary zastosowań. Przykładowo, strategie (μ,λ) sprawdzają się w przeszukiwaniu nieograniczonych i wielowymiarowych przestrzeni rozwiązań. Z kolei dla problemów o dyskretnych i skończonych przestrzeniach rozwiązań zazwyczaj stosuje się strategie $(\mu+\lambda)$ [34].

Istotnym elementem strategii ewolucyjnych jest rekombinacja osobników. W odróżnieniu od krzyżowania w algorytmach genetycznych, w którym para rodziców wyznacza parę potomków, w strategiach ewolucyjnych wiele rozwiązań może podlegać rekombinacji w celu uzyskania jednego potomka [61]. Ponadto rodzice wybierani są z populacji w sposób losowy, bez uwzględnienia ich

ocen. Liczba rozwiązań ρ wybieranych do rekombinacji jest jednym z zewnętrznych parametrów strategii (obok rozmiaru populacji μ i liczby potomków λ), które nie stanowią części opisu osobnika. Zestaw zewnętrznych parametrów strategii oznacza się jako: $(\mu/\rho \dagger \lambda)$ [61]. Algorytm nr 1.3 prezentuje diagram strategii ewolucyjnej.

Algorytm 1.3 Diagram strategii ewolucyjnej

```
1: populacja := inicjalizacjaPopulacji(rozmiarPopulacji: μ);
2: ocenaOsobnikow(populacja);
3: najlepszeRozwiazanie := wybierzNajlepsze(populacja);
4: while warunekZakonczeniaEwolucji == false do
5:     potomkowie := ∅;
6:     for i := 1 to λ do
7:         populacjaRodziców := selekcjaGodowa(populacja, ρ);
8:         potomek_i := rekombinacja(populacjaRodziców);
9:         potomek_i := mutacja(potomek_i);
10:        potomkowie := potomkowie ∪ {potomek_i};
11:    end for
12:    ocenaOsobnikow(potomkowie);
13:    populacja := selekcjaSrodowiskowa(populacja, potomkowie);
14:    najlepszeRozwiazanie := wybierzNajlepsze(populacja, najlepszeRozwiazanie);
15: end while
16: return najlepszeRozwiazanie;
```

Źródło: opracowanie własne na podstawie [34]

1.5 Programowanie genetyczne

Programowanie genetyczne wyróżnia się pośród innych odmian algorytmów ewolucyjnych rodzajem osobników, które składają się na populację. W tym przypadku są to programy komputerowe mające za zadanie rozwiązać wybrany problem. W szczególności, mogą to być implementacje programów do poszukiwania strategii sieci agentów. Celem ewolucji jest opracowanie algorytmu, który wyznaczy rozwiązanie dla zadanego problemu w zależności od parametrów wejściowych. Istotną cechą programowania genetycznego jest konieczność operowania na osobnikach o zmiennej długości [161].

W przypadku programowania genetycznego nie jest konieczna wiedza odnośnie struktury programów. Poszczególne programy w populacji mogą reprezentować algorytmy o dowolnym poziomie złożoności. Ze względu na tę cechę, programowanie genetyczne stosowane jest w obszarach, w których nie dysponujemy wiedzą *a priori* na temat struktury końcowych rozwiązań, np. eksploracja danych [4, 177], bioinformatyka [213] czy analiza rynków kapitałowych [216, 227].

Jednym ze sposobów reprezentacji osobników w programowaniu genetycznym są wyrażenia symboliczne określane również jako S-wyrażenia. Każde S-wyrażenie definiowane jest jako symbol lub wyrażenie w postaci $(a\ b)$, gdzie a i b również są S-wyrażeniami. Pierwszy element uporządkowanej pary $(a\ b)$ zazwyczaj wskazuje na operację ze zdefiniowanego zbioru dostępnych procedur. Drugi element stanowi kolejną parę lub symbol terminalny z określonego z góry zbioru. Przykładowo,

(lg 8) oraz ($*$ (6 7)) stanowią poprawne S-wyrażenia w notacji prefiksowej [161]. Zbiór wykorzystanych procedur $\boldsymbol{\mathcal{F}}$ składa się z dwóch elementów: $\boldsymbol{\mathcal{F}} = \{lg, *\}$, natomiast zbiór symboli terminalnych $\boldsymbol{\mathcal{T}}$ obejmuje liczby naturalne.

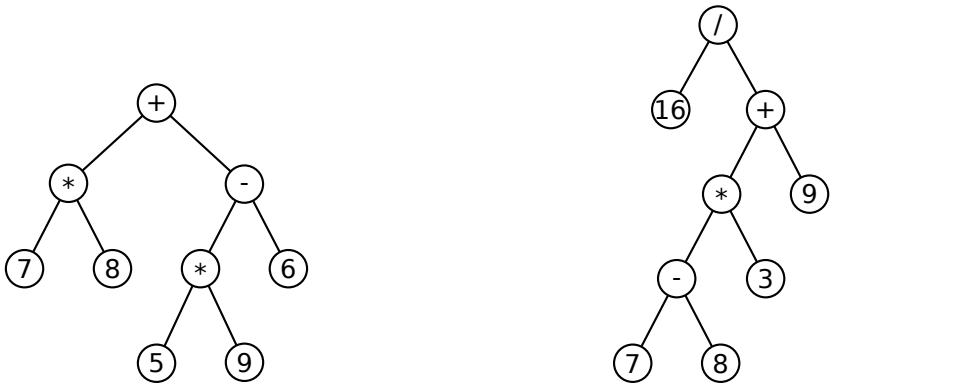
W ogólności dla zbioru procedur $\boldsymbol{\mathcal{F}}$ oraz zbioru symboli terminalnych $\boldsymbol{\mathcal{T}}$ zdefiniowanych następująco:

$$\boldsymbol{\mathcal{F}} = \{f_1, \ldots, f_n, \ldots, f_N\}, \tag{1.2}$$

$$\boldsymbol{\mathcal{T}} = \{t_1, \ldots, t_m, \ldots, t_M\}, \tag{1.3}$$

spełniony musi być *warunek zgodności* mówiący, że każda z procedur jako swój argument musi akceptować wszystkie wartości, jakie mogą być zwracane przez dowolną procedurę oraz wszystkie spośród zdefiniowanych terminali. Daje to gwarancję, że S-wyrażenia implementowane za pomocą operacji genetycznych będą poprawne niezależnie od doboru elementów w poszczególnych parach (a b) [161].

Elementy populacji w postaci wyrażeń symbolicznych reprezentowane są zazwyczaj jako drzewa. Węzły wewnętrzne odpowiadają operacjom ze zdefiniowanego zbioru procedur, a ich argumentami są wartości zwracane przez węzły potomne. Liście w drzewie odpowiadają elementom ze zbioru symboli terminalnych. Na rysunku 1.4 zaprezentowano przykłady drzew zbudowanych na postawie wybranych S-wyrażeń.



(a) Drzewo S-wyrażenia (+ (∗ (7 8)) (− (∗ (5 9)) 6)) (b) Drzewo S-wyrażenia (/ 16 (+ (∗ (− (7 8)) 3) 9))

Rysunek 1.4: Przykłady drzew reprezentujących wybrane S-wyrażenia
Źródło: opracowanie własne

Do inicjalizacji populacji początkowej zazwyczaj wykorzystuje się dwie metody konstrukcji osobników. W metodzie *konstruowania pełnych drzew* jako kolejne węzły wybiera się losowe elementy wyłącznie ze zbioru procedur $\boldsymbol{\mathcal{F}}$ aż do osiągnięcia pożądanej głębokości [161]. Następnie, aby nie przekroczyć założonego limitu wybierane są elementy ze zbioru symboli terminalnych $\boldsymbol{\mathcal{T}}$. Wszystkie budowane w ten sposób osobniki cechują się drzewami o zbliżonej strukturze, a różnice wynikają jedynie z obecności procedur o różnej liczbie argumentów w zbiorze $\boldsymbol{\mathcal{F}}$.

W drugiej metodzie *przyrostowego konstruowania drzew* na każdym poziomie głębokości wybiera się losowe elementy ze zbioru procedur oraz ze zbioru terminali [161]. Jeśli wylosowany

zostanie symbol terminalny, generowany węzeł staje się liściem i „rozrost" gałęzi drzewa będzie zatrzymany. Po osiągnięciu założonej głębokości, wybór wierzchołka następuje wyłącznie ze zbioru symboli terminalnych, aby nie została przekroczona maksymalna liczba poziomów drzewa. W budowanych w ten sposób drzewach zazwyczaj nie wszystkie liście są na tym samym poziomie, a struktury konstruowanych drzew są zdeterminowane liczebnościami zbiorów $\mathcal{F}$ i $\mathcal{T}$. Jeśli zbiór procedur jest znacząco liczniejszy niż zbiór terminali ($|\mathcal{F}| \gg |\mathcal{T}|$), drzewa są zbliżone do tych uzyskiwanych przy użyciu pierwszej metody. W wypadku, gdy $|\mathcal{F}| \ll |\mathcal{T}|$, metoda generuje drzewa o niewielkiej głębokości, niezależnie od przyjętego limitu.

Żadne z powyższych podejść nie gwarantuje dużej różnorodności pod względem kształtu lub rozmiaru konstruowanych drzew [219]. Z tego względu *Koza* zaproponował inicjalizację *hybrydową* (ang. *ramped half-and-half*). Połowa osobników jest w tym podejściu generowana z wykorzystaniem metody konstruowania pełnych drzew, druga połowa – metody przyrostowego konstruowania drzew. Równocześnie wykorzystywana jest sekwencja progów maksymalnej głębokości drzew dla różnych podgrup osobników [161].

Operacje genetyczne definiowane są w kontekście struktury drzew reprezentujących programy komputerowe. Krzyżowanie polega na zamianie fragmentów drzew między dwoma osobnikami. W obrębie każdego z drzew rodziców wybierany jest losowo punkt krzyżowania. Następnie poddrzewa o korzeniach w wybranych węzłach są wymieniane między osobnikami [161].

Wybór punktów krzyżowania spośród węzłów drzew nie odbywa się przeważnie z równomiernym rozkładem prawdopodobieństwa. W drzewach reprezentujących S-wyrażenia średnia liczba potomków pojedynczego węzła wewnętrznego jest nie mniejsza niż dwa. Oznacza to, że większość węzłów stanowią liście [219]. Przy równomiernym rozkładzie prawdopodobieństwa wyboru węzłów, większość operacji krzyżowania polega na wymianie niewielkich fragmentów drzew, a znaczna część ogranicza się tylko do liści. Aby temu zapobiec, zazwyczaj 90% punktów krzyżowania wybieranych jest spośród węzłów wewnętrznych, a tylko 10% – ze zbioru liści [161]. Oprócz tempa krzyżowania p_c, ważnym parametrem jest współczynnik wrażliwości węzłów wewnętrznych na krzyżowanie p_w, który w powyższym przypadku wynosi 0,9.

Mutacja osobników może być realizowana na kilka sposobów. *Mutacja pojedyncza* polega na wybraniu losowego węzła drzewa i zmianie jego wartości. Gdy wybrano liść zawierający symbol terminalny, zastępowany jest on innym terminalem. W przypadku, gdy wybrano węzeł wewnętrzny, odpowiadająca mu procedura zostaje zamieniona na inną o takiej samej liczbie argumentów. *Mutacja poddrzewa* polega na losowym wyborze poddrzewa z programu i zastąpieniu go wygenerowanym losowo poddrzewem.

Na etapie selekcji najczęściej stosowana jest selekcja turniejowa [219]. W jej przypadku dobór opiera się na tym, czy osobnik posiada wyższą jakość od pozostałych uczestników turnieju, a nie na tym, o ile jest ona wyższa. Odpowiada to przeskalowaniu jakości rozwiązań i pozwala na utrzymanie stałej presji selekcyjnej, co zapobiega zdominowaniu populacji przez dobrze dopasowanego osobnika [219].

Schemat programowania genetycznego zaprezentowano w postaci algorytmu nr 1.4. Po inicjalizacji populacji początkowej (linie 1-3), rozpoczyna się główna pętla algorytmu (linia 4). W każdej

epoce konstruowany jest zbiór potomków (linie 5-21), na który składają się osobniki będące efektem krzyżowania (linie 8-12), rozwiązania podlegające mutacji (linie 13-17) oraz przeniesione w efekcie reprodukcji osobniki z poprzedniej epoki (linie 18-21). Wybór operacji do wykonania odbywa się w linii 7.

Algorytm 1.4 Diagram programowania genetycznego

```
1: populacja := inicjalizacjaPopulacji(rozmiarPopulacji: μ);
2: ocenaOsobnikow(populacja);
3: najlepszeRozwiazanie := wybierzNajlepsze(populacja);
4: while warunekZakonczeniaEwolucji == false do
5:     potomkowie := ∅;
6:     while |potomkowie| < |populacja| do
7:         operacjaGenetyczna := wyborOperacji();
8:         if operacjaGentyczna == krzyżowanie then
9:             (rodzic1, rodzic2) := selekcjaGodowa(populacja);
10:            (potomek1, potomek2) := krzyzowanie(rodzic1, rodzic2);
11:            potomkowie := potomkowie ∪ {potomek1, potomek2};
12:        end if
13:        if operacjaGentyczna == mutacja then
14:            rodzic := selekcjaGodowa(populacja);
15:            potomek := mutacja(rodzic);
16:            potomkowie := potomkowie ∪ {potomek};
17:        end if
18:        if operacjaGentyczna == reprodukcja then
19:            rodzic := selekcjaGodowa(populacja);
20:            potomkowie := potomkowie ∪ {rodzic};
21:        end if
22:    end while
23:    ocenaOsobnikow(potomkowie);
24:    najlepszeRozwiazanie := wybierzNajlepsze(potomkowie, najlepszeRozwiązanie);
25:    populacja := zastąpienie(populacja, potomkowie);
26: end while
27: return najlepszeRozwiazanie;
```

Źródło: opracowanie własne na podstawie [34]

Zazwyczaj operacja krzyżowania jest wykorzystywana do wyznaczenia około 90% osobników nowej populacji. Kolejne 8% rozwiązań jest efektem reprodukcji, w której wybrany osobnik jest kopiowany w niezmienionej postaci do następnej epoki. Pozostałe rozwiązania są efektem mutacji [163]. Możliwy jest także inny scenariusz, w którym z tempem p_c krzyżowane są wszystkie drzewa w populacji, a następnie przeprowadzana jest mutacja wszystkich drzew w tempie p_m. Po uzyskaniu kompletnego zbioru potomków, nowe osobniki są poddawane ocenie, a najlepszy z nich zostaje zapamiętany (linie 23-24). Nie występuje selekcja środowiskowa, a jedynie zastąpienie populacji z poprzedniej epoki nowymi osobnikami (linia 25).

Silnie typowane programowanie genetyczne STGP (ang. *Strongly Typed Genetic Programming*) pozwala na definiowanie typów wartości przyjmowanych i zwracanych przez procedury ze zbioru $\mathcal{F}$ oraz typów symboli terminalnych [196]. Podejście to jest przydatne, gdy rozwiązywany problem wykorzystuje dane różnych typów i trudne jest zdefiniowanie operacji dla wszystkich możliwych kombinacji argumentów. Przykładowo, operacja dodawania wektora 5-elementowego do macierzy

o wymiarach 3×4 nie posiada interpretacji matematycznej, więc nie jest możliwe zdefiniowanie operatora dodawania o naturalnej semantyce dla tych typów danych [196]. Osobniki, w których wystąpi taka operacja, mogą być karane obniżeniem wartości *fitness* do poziomu uniemożliwiającego ich reprodukcję. Pozwala to na wyeliminowanie wadliwych rozwiązań, jednak w wielu problemach powoduje odrzucanie znacznej części populacji, pogarszając sprawność programowania genetycznego [219]. Zastosowanie silnego typowania zawęża obszar poszukiwań do programów, które są poprawne.

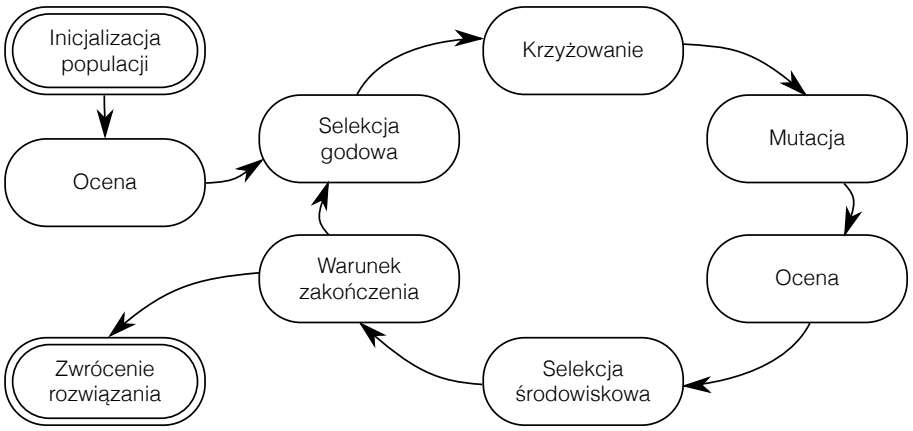
Podejście STGP pozwala na definiowanie lokalnych reguł poprawności drzew. *Programowanie genetyczne oparte o gramatykę* (ang. *Grammar-based Genetic Programming*) umożliwia dodatkowo formułowanie reguł globalnych [265]. Poza definicjami wymaganych typów gramatyka opisuje ograniczenia odzwierciedlające wiedzę na temat problemu. Pozwala to na dalsze zawężenie przestrzeni poszukiwań i ukierunkowanie algorytmu na rozwiązania o zadanej strukturze [190].

Powyższe podejście wykorzystano w *ewolucji gramatycznej* (ang. *Grammatical Evolution*), w której zamiast drzew programów wprowadzono liniowe reprezentacje osobników [205]. Ciąg bitów koduje w tym przypadku numery reguł gramatyki wykorzystywanych do zbudowania wynikowego programu. Na etapie mapowania genotyp-fenotyp reprezentacja liniowa może być przekształcona na drzewo programu na podstawie zadanej gramatyki. Dzięki temu wynikowe programy są niezależne od wybranej metody generowania osobników. W kontekście rozwiązań o liniowych reprezentacjach wykorzystać można również inne algorytmy ewolucyjne lub nawet zupełnie inne metaheurystyki [204].

Z kolei w przypadku *kartezjańskiego programowania genetycznego* CGP (ang. *Cartesian Genetic Programming*) zamiast drzew wykorzystywane są grafy programów. Dostępne procedury są ulokowane na dwuwymiarowej siatce wierszy i kolumn. Krawędzie między elementami w różnych kolumnach oraz ich powiązania z parametrami wejściowymi i wyjściowymi określają działanie programu [194]. Osobniki kodowane są liniowo na poziomie genotypu jako ciągi liczb całkowitych opisujących krawędzie grafu [193]. Dla osobników reprezentujących program wykorzystywana jest wyłącznie mutacja. Wykazano, że w tym przypadku krzyżowanie nie poprawia w istotny sposób wyników algorytmu [193]. Z drugiej strony krzyżowanie ma duże znaczenie, gdy na osobnika składa się wiele programów, z których każdy odpowiada za wyznaczenie innego parametru rozwiązania [260, 261].

1.6 Ujednolicone podejście do przetwarzania ewolucyjnego

Każde z przedstawionych podejść posiada cechy, które wyróżniają je na tle pozostałych [34]. Wszystkie omówione algorytmy opierają się jednak na tym samym schemacie działania, który przedstawiono na rysunku 1.5. Podczas projektowania algorytmów dla specyficznych problemów optymalizacji często korzystne jest połączenie cech wywodzących się z różnych nurtów. Pozwala to na uniknięcie wad poszczególnych podejść i poprawę rezultatów uzyskiwanych w rozpatrywanym zagadnieniu optymalizacji [38].



Rysunek 1.5: Diagram algorytmu ewolucyjnego
Źródło: opracowanie własne na podstawie [34]

Niektóre etapy można pominąć, jeśli ma to uzasadnienie w specyfice rozwiązywanego zagadnienia. Z kolei w pozostałych etapach wykorzystuje się te spośród dostępnych opcji, które prowadzą do rezultatów najwyższej jakości. Przykładowo, na etapie selekcji zastosować można jedną z czterech podstawowych procedur: selekcję proporcjonalną (np. ruletkową), selekcję turniejową, ranking liniowy lub deterministyczną selekcję $(\mu/\rho \stackrel{+}{,} \lambda)$. Różnią się one poziomem presji selekcyjnej i możliwościami jej kontroli [20].

Większa presja selekcyjna oznacza szybsze zanikanie różnorodności w populacji. Dywersyfikacja osobników powinna być utrzymywana za pomocą mutacji. Brak różnorodności rozwiązań na wczesnym etapie ewolucji uniemożliwia wyjście poza optima lokalne funkcji dopasowania [17]. Algorytmy ewolucyjne, które w głównej mierze opierają się na rekombinacji lub krzyżowaniu, przeważnie wykorzystują operatory selekcji cechujące się mniejszą presją selekcyjną. Z kolei w implementacjach, w których mutacja odgrywa kluczową rolę, zazwyczaj wprowadza się większą presję selekcyjną [20].

Jednym z kluczowych aspektów algorytmów ewolucyjnych jest reprezentacja osobników, która determinuje operacje genetyczne. Poszczególne odmiany algorytmów wykorzystują reprezentacje, które wynikają ze specyfiki obszarów ich zastosowań. Reprezentacje fenotypowe ułatwiają definiowanie operatorów genetycznych gwarantujących uzyskanie rozwiązań dopuszczalnych. W przypadku operacji na poziomie genotypu weryfikacja ograniczeń wymaga przeprowadzenia mapowania genotyp-fenotyp [61].

Zastosowania algorytmów ewolucyjnych obejmują bardzo szerokie spektrum problemów, m.in.: predykcję ruchu ulicznego [124], rozpoznawanie akcji użytkownika na podstawie odczytów z czujników [60, 273], prognozowanie i wykrywanie zmian nowotworowych [171, 180, 233], analizę białek organizmów żywych [225], wyznaczanie trajektorii pojazdów podwodnych [23], klasyfikację obrazów [248], czy eksplorację danych [70]. Projekty prowadzone są również w zakresie optymalizacji przydziału zasobów w systemach rozproszonych [26, 149, 247].

Ważnym nurtem w obrębie programowania genetycznego jest ewolucyjne dostrajanie aplikacji (ang. *genetic improvement*) [232]. W tym podejściu jako osobniki populacji początkowej wykorzystuje się istniejące programy komputerowe rozwiązujące określone problemy. Za pomocą operacji genetycznych następuje ich optymalizacja [266]. Metoda może odnosić się do wydajności [214, 262] bądź jakości aplikacji rozumianej jako liczba występujących w niej błędów, styl kodu źródłowego, czy przydatność zwracanych wyników [110]. *Langdon* za pomocą genetycznego dostrajania poprawił sprawność programów do przetwarzania trójwymiarowych obrazów medycznych [172] i analizy genomu [170].

Programowanie genetyczne może być również wykorzystywane jako metoda inżynierii wstecznej. Prowadzono badania w zakresie automatycznej refaktoryzacji kodu, jego modularyzacji i testów regresji [115]. Opracowane rozwiązania pozwalają na identyfikację powtarzających się fragmentów aplikacji [262], a także na automatyczne rozpoznawanie i naprawę błędów występujących w programach [173, 263]. *Arcuri* zaproponował model nawiązujący do relacji pomiędzy drapieżnikami a ofiarami, w którym ewolucji podlega zarówno aplikacja, jak i jej testy akceptacyjne [19]. Koewolucja obu elementów pozwala na utrzymanie presji selekcyjnej wymuszającej identyfikację kolejnych błędów. Proponowane są również metody rozszerzania aplikacji o nowe funkcjonalności poprzez ich transplantację z innych pakietów oprogramowania [33], a także metody automatycznej implementacji nowych procedur w oparciu o zdefiniowane testy akceptacyjne [114].

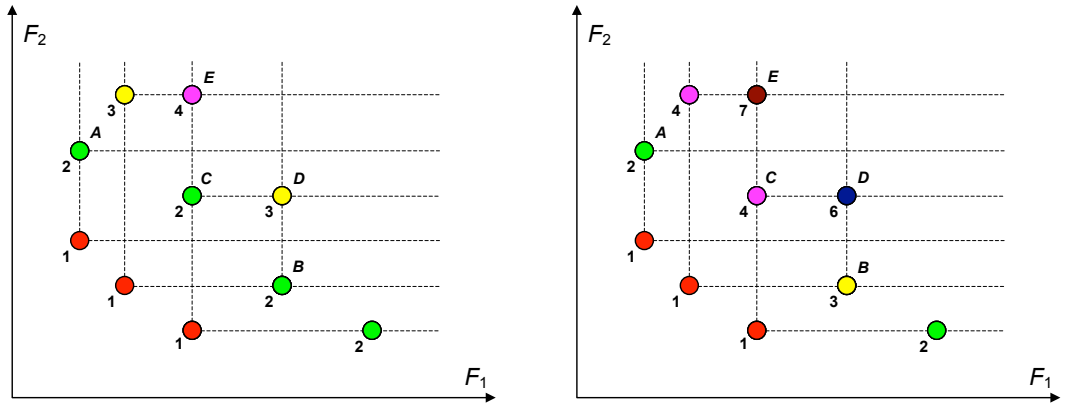
1.7 Ewolucyjna optymalizacja wielokryterialna

Pierwszy wielokryterialny algorytm ewolucyjny VEGA (ang. *Vector Evaluated Genetic Algorithm*) został opracowany w 1985 roku przez *Schaffera* [235]. Idea opiera się na podziale populacji rozwiązań na tyle równolicznych podpopulacji, ile jest kryteriów cząstkowych. Selekcja osobników zachodzi w każdej podpopulacji w oparciu o dopasowanie uwzględniające tylko jedno kryterium. Natomiast mutacja i krzyżowanie realizowane są w całej populacji [235].

Zauważono, że oceny o umiarkowanych wartościach kryteriów są pomijane, co jest wadą algorytmu VEGA. Z tego powodu *Goldberg* opracował procedurę nadawania rang do sortowania poziomów dominacji w populacji [106]. Następnie *Srinivas* i *Deb* opracowali algorytm NSGA (ang. *Non-dominated Sorting Genetic Algorithm*) z procedurą nadawania rang [245].

Fonseca i *Fleming* zaproponowali w algorytmie MOGA (ang. *Multi-Objective Genetic Algorithm*) inną procedurę, w której rangi wyznaczane są na podstawie *liczby osobników dominujących* [82]. Dotychczas ani eksperymentalnie ani teoretycznie nie rozstrzygnięto, która procedura jest bardziej efektywna podczas wyznaczania rozwiązań *Pareto*-optymalnych [55]. Na rysunku 1.6 przedstawiono rangi obliczone za pomocą obu procedur.

Przykładowo, liczba ocen dominujących nad punktami B i C (rys. 1.6b) jest większa niż odpowiadające tym punktom numery poziomów niezdominowania (rys. 1.6a). W konsekwencji, oceny B i C cechują się niższymi prawdopodobieństwami selekcji wyznaczonymi na podstawie rang wg procedury *Fonseci-Fleminga*. Może to prowadzić do pomijania rozwiązań „pośrednich" lub przedwczesnej zbieżności do minimów lokalnych [83].



(a) W oparciu o numer poziomu niezdominowania rozwiązania

(b) W oparciu o liczbę osobników dominujących

Rysunek 1.6: Wybrane procedury nadawania rang [276]

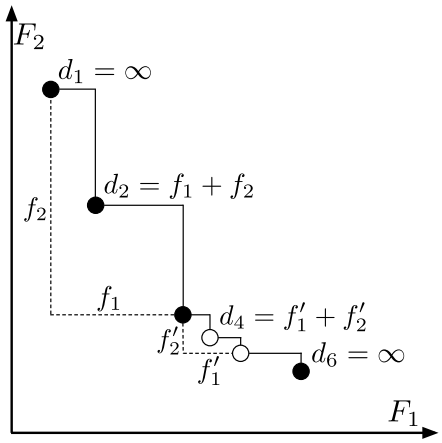
Horn i *Nafpliotis* skonstruowali wielokryterialny algorytm NPGA (ang. *Niched Pareto Genetic Algorithm*), wprowadzając procedurę niszowania, która pomaga w utrzymaniu różnorodności w populacji poprzez eliminowanie podobnych do siebie osobników [121]. Podobieństwo może być określane w oparciu o usytuowaniu ocen rozwiązań w przestrzeni kryterialnej lub na podstawie rozmieszczenia osobników w przestrzeni rozwiązań.

Jedną ze stosowanych metod niszowania jest *fitness sharing*. W podejściu tym sprawność osobnika jest zmniejszana proporcjonalnie do liczby rozwiązań znajdujących się w jego sąsiedztwie. Konieczne jest w tym przypadku określenie metryki podobieństwa osobników. Dla rozwiązań reprezentowanych na poziomie genotypu jako łańcuchy binarne, wykorzystana może zostać odległość *Levenshteina*, która określa dystans edycyjny pomiędzy ciągami symboli. Rozmiar sąsiedztwa (promień niszy) jest kontrolowany za pomocą parametru σ_{share} [234].

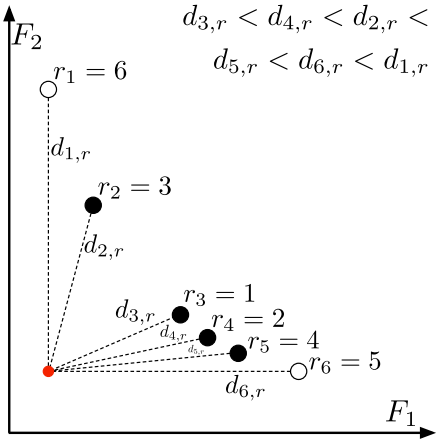
Deb wykorzystał mechanizm niszowania oparty o maksymalizowaną miarę oddalenia ocen w przestrzeni kryterialnej (ang. *crowding distance*), opracowując nową wersję algorytmu NSGA-II [62]. W celu wyznaczenia zagęszczenia osobniki o takiej samej randze sortowane są względem kolejnych kryteriów optymalizacji. Początkowa wartość miary wynosi $d = 0$ dla wszystkich rozwiązań. Następnie dla każdego z kryteriów, pierwszy i ostatni osobnik na posortowanej liście otrzymuje miarę oddalenia $d \to \infty$. Miary dla pozostałych osobników powiększane są o odległość pomiędzy ich najbliższymi sąsiadami, co zobrazowano na rys. 1.7a.

Mniejsze wartości miary d wskazują na większe zagęszczenie osobników. Operator $\prec_n$ dla rozwiązań o tej samej randze preferuje osobniki, których oceny usytuowane są w rejonach przestrzeni kryterialnej o mniejszym zagęszczeniu. Spośród rozwiązań kandydujących do kolejnej populacji wybierane są te, które wskazuje operator $\prec_n$ [64]. Na rysunku 1.7a odrzucono osobniki oznaczone okręgami.

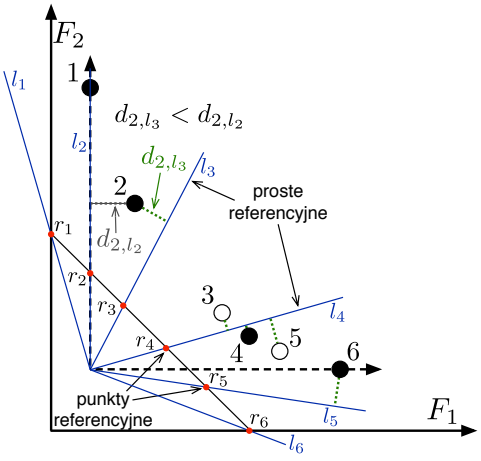
Rozszerzony algorytm R-NSGA-II wzbogacono o wykorzystanie punktów referencyjnych dla określenia zagęszczenia osobników [65]. W tym podejściu wyznaczana jest odległość euklidesowa pomiędzy elementami populacji a rozwiązaniem referencyjnym. Następnie osobniki są sortowane



(a) NSGA-II: d – miara oddalenia od innych rozwiązań



(b) R-NSGA-II: r – ranking na podstawie odległości od punktu referencyjnego



(c) NSGA-III: stopień pokrycia punktów referencyjnych

Rysunek 1.7: Wyznaczanie wartości miary oddalenia i miary zagęszczenia osobników w algorytmach z rodziny NSGA
Źródło: opracowanie własne na podstawie [62, 65, 238]

rosnąco według uzyskanych odległości, co wyznacza ranking. Pozycja rozwiązania w rankingu staje się jego miarą zagęszczenia r względem punktu referencyjnego. Na rysunku 1.7b rozwiązanie referencyjne oznaczono kolorem czerwonym. Operator $\prec_n$ preferuje osobniki o niższych wartościach miary r. W przypadku wielu punktów referencyjnych, rankingi oddalenia są budowane dla każdego punktu z osobna. Miara zagęszczenia jest najwyższą pozycją osobnika w dowolnym z rankingów.

Deb i *Jain* zaproponowali w 2014 roku nową wersję algorytmu NSGA-III, która poprawia wyniki dla problemów obejmujących więcej niż trzy kryteria optymalizacji [63]. Punkty referencyjne są w tym przypadku rozłożone na hiperpłaszczyźnie przecinającej osie przestrzeni kryterialnej. Na etapie wyboru rozwiązań do kolejnej populacji wykonywana jest normalizacja przestrzeni kryterialnej względem skrajnych ocen osobników aktualnej populacji. Po tej operacji, osobniki o skrajnych ocenach usytuowane są na osiach znormalizowanej przestrzeni kryterialnej. Następnie dla każdego

punktu referencyjnego wyznaczana jest prosta referencyjna przecinająca początek znormalizowanego układu ocen i rozpatrywany punkt.

Zbiór osobników kandydujących do kolejnej populacji obejmuje rozwiązania o randze granicznej oraz o rangach lepszych. Dla wszystkich rozwiązań z tego zbioru wyznaczana jest najbliższa im prosta referencyjna. Rozwiązania wybierane są w taki sposób, aby obejmowały jak największą liczbę prostych referencyjnych. Omówione podejście jest przydatne, gdy znany jest kształt frontu *Pareto* [63].

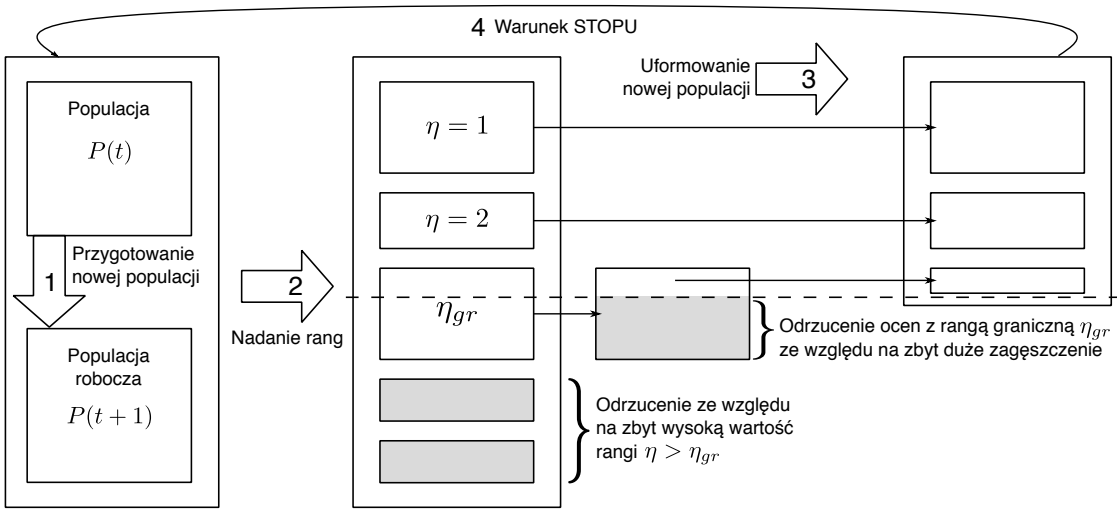
Na rysunku 1.7c zaprezentowano metodę wyznaczania zagęszczenia w algorytmie NSGA-III na przykładowym zbiorze ocen osobników o numerach od 1 do 6. Osie znormalizowanej przestrzeni kryterialnej oznaczono liniami przerywanymi. Proste referencyjne wyróżniono kolorem niebieskim i ponumerowano zgodnie z indeksami punktów referencyjnych: dla punktu r_i – prosta l_i, $i = \overline{1,6}$. Dla każdej oceny wyznacza się najbliższą prostą referencyjną. Ocenę nr 2 przypisano do prostej l_3, ponieważ odległość d_{2,l_3} jest mniejsza niż odległość do prostej l_2. Odcinki koloru zielonego oznaczone linią przerywaną wskazują odległość do najbliższej prostej dla każdej oceny.

W szczególnych przypadkach ocena może leżeć na prostej referencyjnej, np. ocena osobnika nr 1 oraz prosta l_2. Niektórym prostym przypisano więcej niż jedną ocenę, np. prosta l_4 i oceny o numerach 3, 4 i 5. Z kolei prostym referencyjnym l_1 oraz l_6 nie przypisano żadanych ocen. Oceny rozwiązań zapewniające pokrycie największej liczby prostych referencyjnych oznaczono czarnymi punktami, oceny osobników odrzuconych – okręgami. Warto zauważyć, że zmiana sposobu wyznaczania zagęszczenia skutkuje wyborem innych osobników do kolejnej epoki. Na rysunkach 1.7a-1.7c przedstawiono oceny o takich samych współrzędnych w przestrzeni kryterialnej. W każdym przypadku odrzucono inne punkty.

Na rysunku 1.8 przedstawiono diagram procedury aktualizacji populacji w algorytmach NSGA-II i NSGA-III. Kolejne kroki pozwalają na nadanie rang rozwiązaniom, a następnie odrzucenie tych osobników, które uzyskały rangi wykraczające poza wartość graniczną η_{gr}. Część osobników z oceną równą η_{gr} również zostaje odrzucona ze względu na zbyt duże zagęszczenie. Jak pokazano na rysunku 1.7, wybór osobników do odrzucenia zależy od przyjętej miary zagęszczenia lub miary oddalenia. Zaletą miary opartej na oddaleniu od innych rozwiązań (algorytm NSGA-II) jest zdolność adaptacji do populacji występujących w kolejnych epokach. Ponadto nie wymaga ona dodatkowych danych wejściowych w postaci punktów referencyjnych [238].

Zaproponowany w 2015 roku algorytm U-NSGA-III [238] może zostać wykorzystany do rozwiązywania problemów jednokryterialnych oraz wielokryterialnych. Rozwiązanie bazuje na algorytmie NSGA-III, który rozszerzono o możliwość adaptacji do problemu w oparciu o zadaną liczbę kryteriów, bez konieczności podawania dodatkowych parametrów.

W algorytmie SPEA (ang. *Strength Pareto Evolutionary Algorithm*) wykorzystuje się podpopulację rozwiązań niezdominowanych P', którym przypisywane jest większe prawdopodobieństwo reprodukcji niż osobnikom z populacji zasadniczej P, co zapobiega utracie rozwiązań efektywnych [280]. Wartość dopasowania osobnika z podpopulacji P' zawiera się w przedziale [0,1) i zależy



Rysunek 1.8: Diagram procedury aktualizacji populacji w NSGA-II i NSGA-III [62, 63]

od liczby rozwiązań ze zbioru P, które są przez niego dominowane. Natomiast dopasowanie osobnika z populacji P jest sumą wartości *fitness* rozwiązań ze zbioru P', które nad nim dominują, powiększoną o 1. Nie jest istotne wzajemne dominowanie się osobników w obrębie zbioru P.

Selekcja godowa obejmuje wszystkie rozwiązania ze zbiorów P i P'. W algorytmie SPEA preferuje się niższe wartości *fitness* – dają one większe prawdopodobieństwo reprodukcji osobnika. Przy przyjętym sposobie wyznaczania dopasowania oznacza to, że preferowane są rozwiązania o ocenach w rzadziej eksplorowanych obszarach przestrzeni kryterialnej i frontu *Pareto*. Pozwala to na unikanie nadmiernego zagęszczenia osobników. Ten mechanizm niszowania nie wymaga dodatkowych parametrów jak np. promień niszy [280].

Algorytmy SPEA2 [279] oraz SPEA2+ [134] wprowadziły usprawnienia w zakresie zbieżności i zarządzania różnorodnością populacji. Dalsze propozycje usprawnień towarzyszyły licznym aplikacjom algorytmów z rodziny SPEA do rzeczywistych problemów optymalizacji wielokryterialnej [3, 132, 183]. Algorytm SPEA2 zastosowano także w kontekście programowania genetycznego. Drzewa programów oceniano w dwukryterialnej przestrzeni, która reprezentowała kryteria poprawności działania i rozmiaru programu. Pozwoliło to na kontrolowanie niekorzystnego rozrostu drzew w populacji [41]. Z kolei w podejściu DGEP (ang. *Diversity Guided Evolutionary Programming*) tempo i siła mutacji uzależnione są od różnorodności osobników w populacji [5].

W grupie algorytmów ewolucyjno-neuronowych jedną z pierwszych odmian wielokryterialnych opracowano w 1995 roku dla problemu optymalizacji przydziału zasobów w systemie komputerowym [27]. Zapoczątkowało to powstanie wielu algorytmów ewolucyjno-neuronowych dla zagadnień wielokryterialnych.

Adaptacyjny wielokryterialny algorytm ewolucyjny AMEA zawiera mechanizmy umożliwiające zmianę tempa mutacji oraz krzyżowania w zależności od numeru epoki [22]. Algorytm AMEA wykorzystano do wyznaczenia wysokiej jakości przydziałów modułów programistycznych do komputerów w systemach rozproszonych typu grid [24].

W wektorowych strategiach ewolucyjnych zrezygnowano z krzyżowania, ale ewolucji podlegają także wariancje zmiennych decyzyjnych. W szczególności strategia ewolucyjna PAES (ang. *Pareto Archived Evolution Strategy*) została opracowana przez *Knowlesa* i *Corne'a* do wyznaczania rozwiązań *Pareto*-optymalnych [137]. Zastosowane *archiwum* zabezpiecza przed utratą rozwiązań efektywnych. Decyzja o usunięciu osobników z archiwum opiera się na ich zagęszczeniu. *Corne et al.* w algorytmie PESA (ang. *Pareto Envelope-based Selection Algorithm*) opracowali miarę zagęszczenia, w której uwzględnia się liczebności ocen w hiperześcianach należących do przestrzeni wielokryterialnej [57]. Preferowane są rozwiązania z hiperześcianów o mniejszym zagęszczeniu.

W optymalizacji wielokryterialnej oprócz algorytmów ewolucyjnych stosowane są również inne metaheurystyki. W [187] zamieszczono przegląd wielokryterialnych algorytmów przeszukiwania tabu. Interesujące metaheurystyki, w tym algorytmy symulowanego wyżarzania, omówiono w [255]. Metody wykorzystania logiki rozmytej w zagadnieniach polioptymalizacji omawiają *Farina* i *Amato* [74].

1.8 Algorytm harmoniczny

Algorytmy harmoniczne (ang. *Harmony Search – HS*) zaliczane są do grupy algorytmów ewolucyjnych. Inspiracją jest proces improwizacji muzyków w czasie gry na instrumentach [89]. Wykorzystuje się fakt, że kompozytor wybiera linie melodyczne dla różnych instrumentów, a dyrygent modyfikuje brzmienie utworu w zależności od doświadczenia i możliwości orkiestry. Z kolei muzycy jazzowi podczas improwizowanych sesji dobierają tonację w trakcie gry, aby jak najlepiej dopasować się do linii melodycznej całego zespołu [90, 92]. Zarówno orkiestra po pewnej liczbie prób, jak również muzycy jazzowi po niedługiej improwizacji zazwyczaj wykonują utwór prawie perfekcyjnie.

Zaliczenie algorytmu harmonicznego do algorytmów ewolucyjnych wynika z wielu podobieństw. Pamięć harmoniczna (ang. *Harmony Memory – HM*) przechowująca rozwiązania może zostać porównana do populacji osobników. Mutacja odbywa się na drodze improwizacji, a odpowiednikiem krzyżowania jest konstrukcja nowego rozwiązania na podstawie losowo wybranych wartości z pamięci harmonicznej [98].

Istotną rolę w algorytmie harmonicznym pełnią parametry, za pomocą których kontroluje się przebieg przeszukiwania przestrzeni rozwiązań [95]:

- *BW* (ang. *Bandwidth of Generations*) – zakres modyfikacji zmiennej decyzyjnej wybranej z pamięci harmonicznej;
- *HMCR* (ang. *Harmony Memory Considering Rate*) – prawdopodobieństwo wykorzystania pamięci harmonicznej do wyznaczenia wartości zmiennej decyzyjnej;
- *HMS* (ang. *Harmony Memory Size*) – rozmiar pamięci harmonicznej;
- $NG_{\max}$ (ang. *Number of Generations*) – maksymalna liczba iteracji algorytmu;
- *PAR* (ang. *Pitch Adjusting Rate*) – prawdopodobieństwo modyfikacji zmiennej decyzyjnej wybranej z pamięci harmonicznej.

Algorytm harmoniczny w podstawowej wersji można zastosować do rozwiązywania jednokryterialnych problemów optymalizacji z $J_{\max}$ ciągłymi zmiennymi decyzyjnymi. Uwzględniane są wymagania odnośnie wartości zmiennych decyzyjnych w postaci wektorów ograniczeń dolnych oraz ograniczeń górnych, odpowiednio: $l = [l_1, \ldots, l_j, \ldots, l_{J_{\max}}]^T$ oraz $u = [u_1, \ldots, u_j, \ldots, u_{J_{\max}}]^T$. Zagadnienie optymalizacji definiuje się następująco [99]:

$$\min_{x \in \mathcal{X}} f(x), \tag{1.4}$$

gdzie:
$f : \mathbb{R}^{J_{max}} \to \mathbb{R}$ – funkcja celu,
$x = [x_1, \ldots, x_j, \ldots, x_{J_{\max}}]$ – wektor zmiennych decyzyjnych,
$\mathcal{X}$ – zbiór rozwiązań dopuszczalnych.

Pamięć harmoniczna zawiera *HMS* rozwiązań składających się z $J_{\max}$ zmiennych decyzyjnych. Dodatkowo w pamięci *HM* zapisywane są wartości funkcji sprawności wyliczone dla rozwiązań [209]. Pamięć harmoniczna może mieć zatem postać macierzy o *HMS* wierszach i $J_{\max} + 1$ kolumnach, jak niżej [99]:

$$HM = \begin{bmatrix} x_1^1 & \cdots & x_j^1 & \cdots & x_{J_{max}}^1 & \mathit{fitness}(x^1) \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ x_1^i & \cdots & x_j^i & \cdots & x_{J_{max}}^i & \mathit{fitness}(x^i) \\ \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ x_1^{HMS} & \cdots & x_j^{HMS} & \cdots & x_{J_{max}}^{HMS} & \mathit{fitness}(x^{HMS}) \end{bmatrix}. \tag{1.5}$$

Wartości początkowe pamięci *HM* wyznaczane są w sposób losowy zgodnie z poniższą zależnością [99]:

$$x_j^i = l_j + r_{2j}^i (u_j - l_j),\ i = \overline{1, HMS},\ j = \overline{1,N}, \tag{1.6}$$

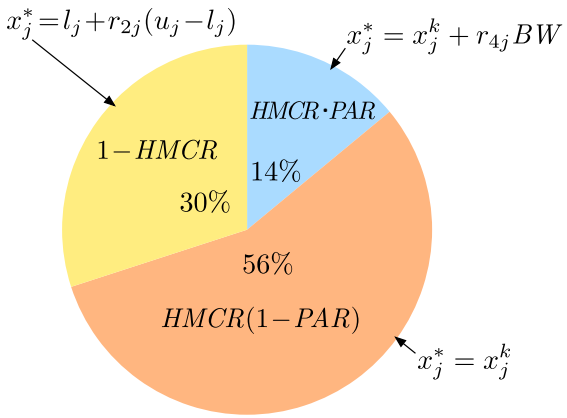
gdzie:
i – numer rozwiązania w pamięci *HM*,
r_{2j}^i – wartość losowa z przedziału $[0; 1]$ używana do wyznaczenia j-tej zmiennej decyzyjnej w i-tym rozwiązaniu.

Nowe rozwiązanie $x^* = [x_1^*, \ldots, x_j^*, \ldots, x_{J_{max}}^*]^T$ w algorytmie harmonicznym wyznacza się za pomocą improwizacji [94]. Algorytm HS jest algorytmem stochastycznym, a do wyznaczenia każdej zmiennej decyzyjnej w wektorze x^* wykorzystuje cztery wartości losowe: $r_{1j},\ r_{2j},\ r_{3j} \in [0; 1]$ oraz $r_{4j} \in [-1; 1]$, dla $j = \overline{1,J_{\max}}$. Sposób wyznaczenia wartości zmiennej decyzyjnej zależy od parametrów *HMCR* i *PAR*, jak niżej [209]:

$$x_j^* = \begin{cases} l_j + r_{2j}(u_j - l_j), & \text{dla } r_{1j} > HMCR, \\ x_j^k, & \text{dla } r_{1j} \leqslant HMCR,\ r_{3j} > PAR, \\ x_j^k + r_{4j} BW, & \text{dla } r_{1j} \leqslant HMCR,\ r_{3j} \leqslant PAR, \end{cases} \tag{1.7}$$

gdzie k – losowo wybrany numer rozwiązania z pamięci *HM*.

Pierwszy przypadek w zależności (1.7) dla $r_{1j} > HMCR$ odpowiada wybraniu losowej wartości zmiennej decyzyjnej x_j^* z pominięciem pamięci HM. Drugi przypadek dla $r_{1j} \leqslant HMCR$ i $r_{3j} > PAR$ to wykorzystanie wartości występującej w jednej z wcześniejszych improwizacji, które zapisano w pamięci harmonicznej. W ostatnim przypadku (dla $r_{1j} \leqslant HMCR$ i $r_{3j} \leqslant PAR$) również wybierana jest wartość zmiennej decyzyjnej spośród wcześniejszych rozwiązań, jednak dodatkowo podlega ona mutacji, której tempo wynika z wartości parametrów r_{4j} i BW. Zależność (1.7) opisuje *regułę trójwartościowej ruletki* [209], którą zobrazowano na rysunku 1.9 dla przykładowych wartości parametrów $HMCR$ i PAR.

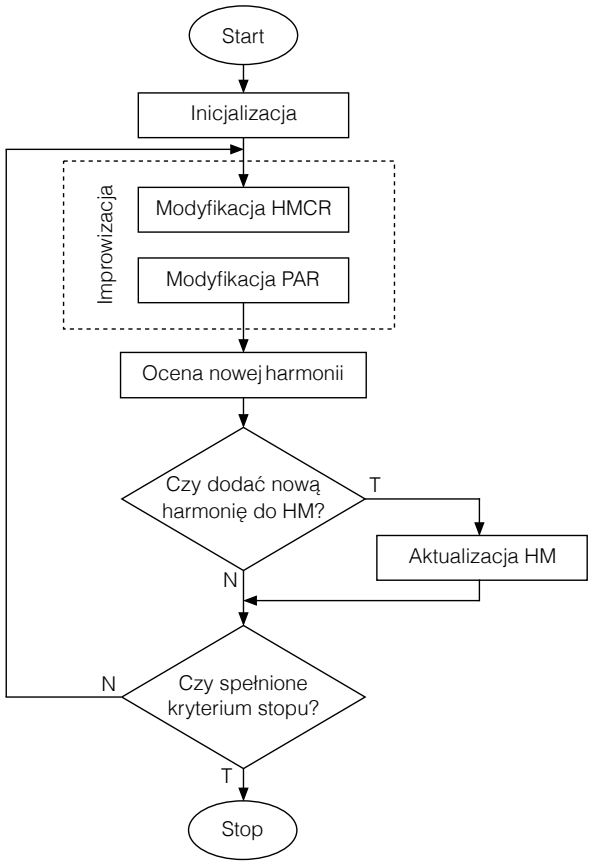


Rysunek 1.9: Reguła trójwartościowej ruletki dla parametrów $HMCR = 0{,}7$ oraz $PAR = 0{,}2$ [209]

Nowe rozwiązanie x^* trafia do pamięci harmonicznej, jeśli przewyższa pod względem sprawności przynajmniej jedno z zapisanych wcześniej rozwiązań. Równocześnie osobnik o najniższej sprawności zostaje wyparty z pamięci HM [209]. Improwizacje są powtarzane do osiągnięcia założonego limitu iteracji NG_{max}. Diagram algorytmu harmonicznego przedstawiono na rysunku 1.10.

Zastosowania algorytmu przeszukiwania harmonicznego są bardzo różnorodne. Przykładowo, *Gil-Lopez et al.* zaaplikowali tę metodę do projektowania sieci telekomunikacyjnej, aby wyznaczyć kompromis pomiędzy kosztem systemu a jego przepustowością [103]. Ważne aplikacje odnoszą się do medycyny, robotyki, problemów sterowania i zarządzania zużyciem energii [88, 89, 91, 101, 184]. Jedno z interesujących zastosowań dotyczy zautomatyzowania komponowania utworów muzycznych [96]. Algorytmy harmoniczne wykorzystywane są również w zagadnieniach optymalizacji przydziału zasobów w gridach [156] i w szeregowaniu zadań [86, 97].

Algorytm harmoniczny porównywano z innymi metodami optymalizacji. *Kim et al.* wykazali, że w problemie optymalizacji sieci wodociągowej algorytm HS wyznacza wyniki wyższej jakości niż algorytm genetyczny [135]. Z kolei *Paluszak* potwierdził szybszą zbieżność algorytmu harmonicznego w porównaniu z algorytmem ewolucyjnym w testowych problemach optymalizacji rozważanych przez *De Jonga* [209].



Rysunek 1.10: Diagram algorytmu harmonicznego dla zagadnień optymalizacji jednokryterialnej [209]

1.9 Wnioski i uwagi

Programowanie genetyczne należy do klasy metaheurystyk ewolucyjnych, do których zalicza się także: programowanie ewolucyjne, algorytmy genetyczne i strategie ewolucyjne. Ponadto, do metod wykorzystujących niektóre paradygmaty podejścia ewolucyjnego, należy zaliczyć metodę ewolucji różnicowej, metodę ewolucji gramatycznej i algorytmy harmoniczne. W algorytmach memetycznych wykorzystuje się kooperację algorytmów ewolucyjnych z innymi metaheurystykami, np. w algorytmie AMEA wykorzystuje się *tabu search* jako mutację w algorytmie ewolucyjnym [25, 28].

W wypadku programowania genetycznego badacze weryfikują, czy źródłowe programy komputerowe, za pomocą których reprezentowany jest osobnik, umożliwiają ewolucyjne wyznaczanie rozwiązań wyższej jakości niż osobniki kodowane binarnie czy też osobniki kodowane za pomocą liczb rzeczywistych. Ze względu na fakt, że programowanie genetyczne obejmuje wytwarzanie oprogramowania, realizowane są projekty nawiązujące do zastosowań w inżynierii oprogramowania, w tym do automatycznego wytwarzania modułów programistycznych, testowania aplikacji, usuwania błędów oraz transplantacji fragmentów kodu z programu-dawcy do programu-biorcy.

Programowanie genetyczne stosowane jest także w uczeniu maszynowym oraz w optymalizacji ze względu na maksymalizację funkcji sprawności. W wypadku uczenia maszynowego dopasowanie osobnika jest zazwyczaj przekształconą funkcją błędu, podobnie jak w sztucznych sieciach neuronowych. Za pomocą selekcji, krzyżowania i mutacji dąży się do wytworzenia nowej generacji lepiej przystosowanych programów komputerowych aż do uzyskania aplikacji cechującej się akceptowalnym błędem dla zbioru uczącego. Zdolności skonstruowanego programu genetycznego w zakresie uogólniania zdobytej wiedzy na przypadki, które nie zostały opisane w zbiorze uczącym, weryfikowane są za pomocą zbioru testowego. Umożliwia to uniknięcie przeuczenia lub niedouczenia aplikacji.

Wykorzystanie programowania genetycznego w optymalizacji jednokryterialnej polega na uwzględnieniu maksymalizowanej funkcji celu oraz funkcji kary za niespełnienie ograniczeń. Jeśli funkcja celu jest minimalizowana, to maksymalizuje się kryterium ze znakiem przeciwnym. W wypadku braku ograniczeń funkcja kary nie występuje. W niektórych podejściach zastępuje się metodę karania rozwiązań niedopuszczalnych inną metodą zmniejszającą sprawność osobników. Wartości zmiennych decyzyjnych wyznaczane są zazwyczaj za pomocą programów o strukturze drzewa, przy czym drzewa powiązane z różnymi zmiennymi decyzyjnymi są rozłączne.

Węzły wewnętrzne drzew reprezentują procedury, a liście – terminale ze zdefiniowanego z góry zbioru. Procedury stosowane do konstrukcji drzew powinny spełniać *warunek zgodności*, który wymaga, aby każda z procedur jako swoje argumenty akceptowała wszystkie wartości, jakie mogą być zwracane przez dowolną procedurę oraz wszystkie spośród zdefiniowanych terminali. Drzewa populacji początkowej mogą być budowane za pomocą metody konstruowania pełnych drzew, metody przyrostowego konstruowania drzew lub metody hybrydowej.

Programowanie genetyczne stosowane jest również w optymalizacji wielokryterialnej. Opiera się ono na wykorzystaniu wielokryterialnych algorytmów ewolucyjnych, takich jak: NSGA, SPEA, PAES czy AMEA. Za pomocą powyższych algorytmów przekształcane są populacje programów komputerowych, których dopasowanie zależy od rankingu dominowania wyznaczonego za pomocą procedury *Goldberga* lub *Fonseci-Flemminga*. Aby nie utracić potencjalnych rozwiązań optymalnych w sensie *Pareto*, wykorzystywane jest archiwum zewnętrzne lub realizowana jest selekcja elitarystyczna typu $(\mu + \lambda)$.

Za pomocą programowania genetycznego wyznacza się nie tylko wartości zmiennych decyzyjnych rozwiązań wysokiej jakości, ale zazwyczaj także „niewielkie" programy. Właśnie te kompaktowe programy mogą być użyte w celu wyznaczenia rozwiązań również dla innych instancji zagadnienia optymalizacji różniących się danymi wejściowymi. Czas potrzebny na uzyskanie nowych rozwiązań jest w tym wypadku znacząco krótszy niż czas obliczeń za pomocą metaheurystycznych metod optymalizacji.

Rozdział 2

Modele inteligentnych agentów

2.1 Klasyfikacja środowisk agentowych

Rozproszona sztuczna inteligencja obejmuje trzy główne obszary: równoległą sztuczną inteligencję (ang. *parallel AI – PAI*), systemy rozproszonego rozwiązywania problemów (ang. *distributed problem solving – DPS*) oraz systemy wieloagentowe (ang. *multiagent systems – MA*) [201]. Nurt *PAI* skupia się na adaptacji algorytmów sztucznej inteligencji do wykorzystania na wieloprocesorowych komputerach, klastrach obliczeniowych i akceleratorach graficznych w celu przyspieszenia obliczeń [42, 175]. Systemy klasy *DPS* opierają się na koncepcji rozproszonych agentów, które dążą do osiągnięcia wspólnego celu. Natomiast w systemach klasy *MA* występować mogą agenty niekooperatywne, które konkurują ze sobą, aby zrealizować własne cele [201].

Wooldridge opisał pięć trendów, które przyczyniły się do popularyzacji agentów: wzrost powszechności komputerów, integracja systemów, rozwój sztucznej inteligencji, delegacja realizacji zadań z ludzi na komputery oraz zorientowanie na użytkownika poprzez wprowadzanie kolejnych warstw abstrakcji, które ukrywają szczegóły działania komputerów [269]. Uwzględnianie uczekiwań użytkowników szczególnie widoczne jest na przykładzie interfejsów komputerowych, które przeszły drogę od tekstowych terminali do środowisk graficznych, a obecnie ewoluują w stronę rzeczywistości wirtualnej. Trend ten widoczny jest również w rozwoju języków programowania: od kodu maszynowego, poprzez asemblery, języki proceduralne aż do języków programowania obiektowego.

Paradygmat obiektowy pozwolił na konstruowanie modułów programistycznych w postaci samodzielnych i kompletnych – przez posiadanie własnego stanu – komponentów [118]. Było to istotne do sformułowania modelu aktorów. Pod pojęciem aktora programistycznego rozumiemy inteligentny komponent, który posiada adres umożliwiający komunikację. Ponadto moduły tej klasy wyposażone są w mechanizm sterujący ich zachowaniami [117]. Podobne cechy przypisuje się również agentom [269].

Wśród innych podobieństw między aktorami i agentami wymienić można autonomiczną realizację współbieżnych akcji, komunikację poprzez wymianę wiadomości i mechanizmy koordynacji

działań dla realizacji postawionych celów. Oba modele pozwalają na projektowanie systemów zbudowanych z komponentów o dynamicznych powiązaniach. Rozwiązania te cechują się elastycznością, zdolnością adaptacji do zmieniających się warunków i otwartością rozumianą jako możliwość integracji systemów w oparciu o standardowe protokoły wymiany komunikatów [128].

Jednakże pomiędzy omawianymi modelami występują istotne różnice, a wymienione cechy wspólne wyrażają się w nich na różne sposoby. Autonomia rozumiana jest jako zdolność do samodzielnego decydowania o realizowanych akcjach. Samodzielność aktorów przejawia się w działaniach *reaktywnych* – po otrzymaniu wiadomości decydują, w jaki sposób ją obsłużyć. Agentom przypisuje się dodatkowo zdolność do działań *proaktywnych*, które nie stanowią odpowiedzi na zewnętrzne bodźce, lecz są efektem wewnętrznych intencji mających doprowadzić do realizacji celów [269].

Ponadto agenty, poza odbieraniem wiadomości, mogą również odbierać sygnały ze środowiska za pośrednictwem perceptorów i reagować na nie, używając efektorów. Interakcje między agentami nie ograniczają się do bezpośredniej wymiany komunikatów, jak w przypadku aktorów. Agent może zmodyfikować stan środowiska w sposób, który zostanie zauważony przez innego agenta i wywoła jego działanie. Takie interakcje mają charakter pośredni.

Wiadomości przesyłane między aktorami zawierają adresy identyfikujące odbiorców. Z kolei komunikaty przekazywane w systemie agentowym nie muszą wskazywać adresata. Zamiast tego specyfikuje się cechy agenta, dla którego przeznaczona jest informacja. Przykładem może być zlecenie realizacji usługi, które zostanie odebrane przez wszystkie agenty zdolne do jej wykonania, co rozpocznie proces negocjacji pomiędzy zleceniodawcą a potencjalnymi usługodawcami [264].

Ponadto komunikacja między aktorami ukierunkowana jest przeważnie na wymianę danych, podczas gdy agenty przygotowują wiadomości w oparciu o ontologie definiujące semantykę komunikatu, co pozwala na modelowanie złożonych procesów interakcji. Przykładowo, ontologia negocjacji wykonania usługi może zawierać obiekty związane z zapytaniem ofertowym, ofertą, odpowiedzią zleceniodawcy, czy kosztem realizacji. Pozwala to na udział w negocjacjach agentom, dla których wybrana ontologia jest zrozumiała [128].

Ważny aspekt, który istotnie odróżnia agenty od aktorów, wiąże się z koordynacją działania komponentów. W przypadku aktorów sprowadza się ona do kontroli dostępu do współdzielonych zasobów, zapewnienia ich spójności i synchronizacji wykonania we współbieżnym środowisku. Koordynacja agentów ma na celu formowanie wirtualnych organizacji, które zrzeszają agenty wspierające się w realizacji celów. Nacisk jest zatem położony na wymianę wiedzy i usług w zorganizowanych zespołach, a nie na synchronizację i niskopoziomowe zarządzanie zasobami [128].

Systemy agentowe stosowane są w różnorodnych obszarach badawczych i przemysłowych. W zastosowaniach tej klasy wykorzystuje się wybrane architektury wewnętrzne agentów i środowiska ich działania. Wyróżnia się środowiska *dostępne*, w których agent może uzyskać komplet informacji o stanie otoczenia, a także środowiska *zamknięte*, w których osiągalna jest jedynie częściowa wiedza, a jej poszerzanie wymaga eksploracji systemu. *Determinizm* środowiska oznacza, że ta sama akcja agenta zawsze powoduje taką samą zmianę stanu otoczenia. W wielu zastosowaniach środowisko jest *niedeterministyczne*, a wyniki podejmowanych działań zależą od innych agentów, upływu czasu, czy zmian zachodzących w otoczeniu [231].

Środowisko niedeterministyczne może być dodatkowo scharakteryzowane jako *dynamiczne*, co oznacza, że jego stan może zmieniać się niezależnie od działań agentów. Natomiast w środowiskach *statycznych* przyczyną modyfikacji stanu są zachowania agentów. Scharakteryzowanie otoczenia wymaga ponadto określenia, czy ma ono charakter *dyskretny*, co oznacza, że zbiór możliwych stanów jest policzalny. W przeciwnym razie określane jest ono jako *ciągłe*. Kolejną cechą środowiska jest *epizodyczność*, która wskazuje, że akcje agentów zależą jedynie od aktualnego stanu otoczenia. Jeśli brane pod uwagę są również stany przeszłe, to środowisko określane jest jako *nieepizodyczne* [231].

2.2 Wewnętrzne architektury agentów

Ze względu na wewnętrzną architekturę agenty można zaliczyć do jednej z trzech kategorii, które określają role i relacje pomiędzy perceptorami, stanem i komponentem decyzyjnym. Agenty *czysto reaktywne* podejmują działania wyłącznie w oparciu o aktualny stan środowiska, który jest przekazywany do komponentu decyzyjnego. Wybór akcji realizowany jest przez odwzorowanie w postaci funkcji, której dziedzina obejmuje możliwe stany środowiska, a przeciwdziedzina – dostępne zachowania. Agent nie posiada wewnętrznego stanu pomiędzy kolejnymi epizodami [264].

W przypadku *agentów z percepcją* stan środowiska jest wstępnie przetwarzany na poziomie perceptorów. Ich rola polega na klasyfikowaniu sygnałów docierających z otoczenia do kategorii specyfikujących cechy środowiska, które są istotne z punktu widzenia akcji agenta. Taka konstrukcja pozwala na uproszczenie komponentu decyzyjnego, ponieważ nie musi on uwzględniać wszystkich możliwych stanów wejściowych agenta. Zamiast tego opiera się na zbiorze znanych cech. Podejście to jest szczególnie przydatne w środowiskach ciągłych, w których liczba możliwych stanów jest nieskończona [270].

Architektura *agentów ze stanem* pozwala na uwzględnienie wcześniejszych zdarzeń w środowisku podczas podejmowania decyzji odnośnie kolejnej akcji. Gdy następuje zmiana odczytu z perceptorów lub otrzymywana jest nowa wiadomość, komponent decyzyjny wykorzystuje informacje przechowywane w aktualnym wewnętrznym stanie agenta, aby wybrać akcję. Realizacja akcji może doprowadzić do zmiany wewnętrznego stanu, co wpływa na przyszłe decyzje agenta. Zdarzenie modyfikacji stanu wewnętrznego również może wyzwalać działanie komponentu decyzyjnego, umożliwiając zachowania proaktywne [264].

Sposób konstrukcji komponentu decyzyjnego, perceptorów i mechanizmu utrzymywania stanu definiowany jest przez tzw. konkretne architektury. Jedna z metod podejmowania decyzji przez agenty opiera się na rachunku logicznym pierwszego rzędu, w którym system opisuje się zbiorem aksjomatów. Odczyty z perceptorów są argumentami reguł wnioskowania, a ich rezultaty przekładają się na akcje do wykonania. Rozwinięciem tej koncepcji jest wykorzystanie planerów. Rozważa się zbiór predykatów charakteryzujących środowisko, zbiór czynności możliwych do wykonania wraz z opisem ich skutków, a także predykat określający cel agenta. Planer po zakończeniu wnioskowania, wyznacza ciąg operacji prowadzących do zadanego celu. Zaletą powyższego podejścia jest przejrzysty dla projektanta opis środowiska i reguł działania agenta [68].

Z drugiej strony proces podejmowania decyzji staje się niejawny dla agenta. Po określeniu danych wejściowych, planer przeprowadza wnioskowanie, nie umożliwiając wglądu w jego przebieg. Nie jest możliwa analiza, jaki wpływ na wynik wnioskowania ma zmiana odczytu z perceptorów, bez ponownego uruchomienia planera [158]. Co więcej, czas potrzebny na wnioskowanie nie jest stały i może zmieniać się w zależności od danych wejściowych. Problemy te ograniczają zastosowania omawianej metody w systemach cechujących się dużą dynamiką, w których konieczne jest szybkie podejmowanie decyzji.

Interesujące podejście do projektowania komponentu decyzyjnego opiera się na założeniu, że inteligentne zachowania nie wynikają z abstrakcyjnych reguł logiki, lecz z interakcji ze światem i mają charakter emergentny – powstają na skutek złożenia wielu prostych zachowań lub odruchów [264]. *Brooks* zaproponował architekturę opartą na przesłankach (ang. *subsumption architecture*), które mają postać reguł wskazujących akcję do wykonania dla określonego odczytu z perceptora, np. jeśli *brak pamięci RAM*, to *relokacja agenta do innnego komputera.* [44]. Dla wybranego stanu środowiska, perceptory agenta mogą zwracać wiele rozpoznanych cech, co oznacza, że aktywnych będzie wiele przesłanek. Wybór jednej z nich opiera się na hierarchii zachowań agenta: wykonana zostanie ta spośród dopasowanych akcji, której przypisano najwyższy priorytet. Odpowiada to wyborowi działania dla zaspokojenia najpilniejszej potrzeby, a gdy ta zostanie spełniona – potrzeb wyższego rzędu, niczym w piramidzie *Maslowa* [188].

Do zalet architektury opartej o przesłanki zalicza się możliwość oszacowania czasu podejmowania decyzji i łatwość modelowania zachowań. Projektant definiuje jedynie zbiór reguł i ich priorytety. W przypadku rozbudowanych hierarchii i złożonych środowisk, które generują wiele bodźców, trudno jest przewidzieć, jakie zachowania wyłonią się ze zdefiniowanych reguł. Wymaga to testów w środowisku docelowym i eksperymentalnego dostrojenia priorytetów. Trudne jest również modelowanie agentów dążących do realizacji złożonych celów.

Architektura *BDI* (ang. *belief-desire-intention*) opiera się na modelu wnioskowania praktycznego, który zaproponował *Bratman* [43]. Celem wnioskowania jest wybór akcji pozwalających na zaspokojenie wymagań jednostki [218]. Model przekonań agenta (ang. *beliefs*) determinuje, jakie stany środowiska są poprawne i osiągalne przy aktualnych warunkach otoczenia. Spośród tego zbioru agent wybiera stan najbardziej pożądany, formułując żądanie (ang. *desire*) jego osiągnięcia. Na drodze do realizacji celu formułowane są cele pośrednie, które przekładają się na akcje agenta. Po wykonaniu wybranych akcji, procedura interpretacji zostaje powtórzona dla nowego stanu środowiska, co obrazuje algorytm nr 2.1 [223].

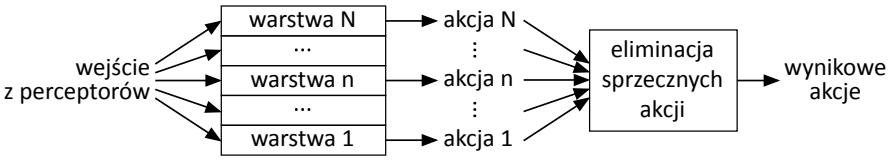
Cele w modelu *BDI* mogą być trwałe i przenoszone pomiędzy kolejnymi iteracjami pętli interpretera (linie 4-12). Odbywa się to dopóki nie zaistnieje jedna z trzech sytuacji: cel zostanie osiągnięty, cel okaże się nieosiągalny lub cel zostanie zastąpiony przez inny cel. Istotne jest, że przekonania agenta mogą być błędne, jeśli nie posiada on pełnej wiedzy o otoczeniu. Różne agenty w tym samym środowisku mogą posiadać odmienną wiedzę, a w efekcie – budować inne zbiory przekonań [218].

Algorytm 2.1 Diagram interpretera BDI

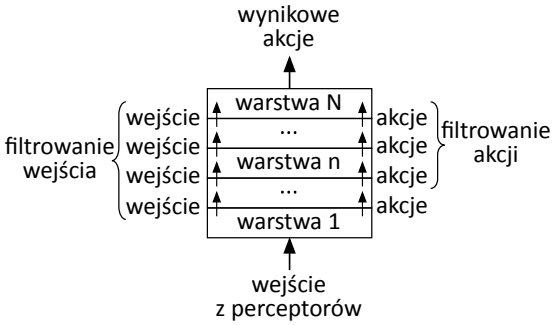
```
1: cele := ∅;
2: intencje := ∅;
3: e := odczytStanuŚrodowiska();
4: while true do
5:     dostępneStanyDocelowe := analizaPrzekonań(e);
6:     cele := formowaniePragnień(cele, dostępneStanyDocelowe);
7:     intencje := aktualizacjaIntencji(intencje, cele);
8:     realizacjaAkcji(intencje);
9:     e := odczytStanuSrodowiska();
10:    usunięcieZrealizowanychCelów(cele);
11:    usunięcieNieosiągalnychCelów(cele);
12: end while
```

Źródło: opracowanie własne na podstawie [223]

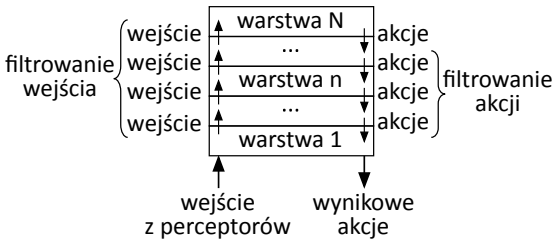
Kolejna grupa architektur wewnętrznych to modele warstwowe, w których wyróżnia się podsystemy odpowiedzialne za różne typy zachowań [264]. Na rysunku 2.1 przedstawiono trzy rodzaje architektur warstwowych. W przypadku warstw poziomych (rys. 2.1a), każda z warstw otrzymuje bezpośrednio sygnały odczytane z perceptorów i zwraca decyzje odnośnie akcji. Podejście to jest podobne do podejścia zastosowanego w architekturze opartej na przesłankach, jednak w tym przypadku warstwa może realizować złożone procesy wnioskowania w przeciwieństwie do prostych reguł w modelu *Brooksa*. Różne warstwy mogą wskazać inne akcje do wykonania. Konieczne jest zatem wprowadzenie komponentu, który eliminuje wykluczające się akcje. Agent wykonuje te spośród wybranych zachowań, które nie kolidują ze sobą. Duża liczba warstw i zachowań przekłada się na skomplikowaną konstrukcję komponentu, który może inicjować sprzeczne akcje.



(a) Warstwy poziome



(b) Warstwy pionowe (jeden przebieg)



(c) Warstwy pionowe (dwa przebiegi)

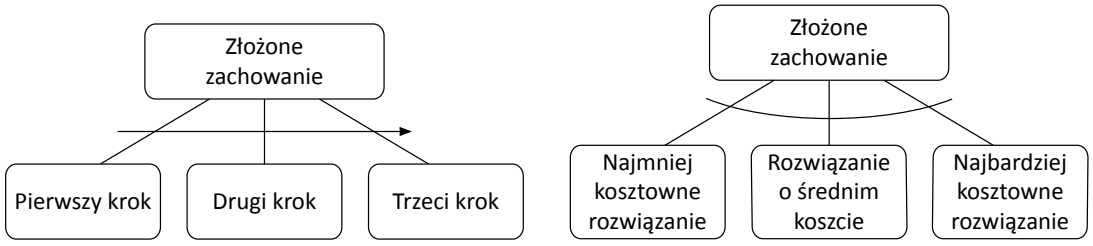
Rysunek 2.1: Przepływ informacji z perceptorów i decyzji przy wyborze akcji w architekturach warstwowych

Źródło: opracowanie własne na podstawie [264]

Zastosowanie warstw pionowych (rys. 2.1b i 2.1c) ułatwia wybór akcji do wykonania. W tym przypadku odczyty z perceptorów kierowane są na wejście najniższej warstwy. Następnie każda kolejna warstwa pełni rolę filtra dla wyników z poprzednich warstw. W modelu z pojedynczym przebiegiem (rys. 2.1b), ostatnia warstwa zwraca akcję do wykonania. Z kolei w przypadku dwóch przebiegów, decyzja odnośnie wybranej akcji jest przesyłana w kierunku przeciwnym niż propagowane są informacje z perceptorów (rys. 2.1c). Niższe warstwy filtrują akcje, które zostały wytypowane przez warstwy wyższe. Wadą omawianych modeli jest brak odporności na błędy, które mogą być propagowane do kolejnych warstw [264].

Inna architektura pozwalająca na zdekomponowanie procesu decyzyjnego na zbiór prostszych i łatwiejszych w modelowaniu elementów, opiera się na koncepcji drzew zachowań [125]. W tym podejściu definiowany jest główny cel działań agenta, a następnie identyfikowane są kroki – cele pośrednie – pozwalające na wykonanie zadania. Każdy z celów pośrednich może zostać zdekomponowany na cele pomocnicze, co odwzorowuje struktura drzewa zachowań. W liściach drzew zgromadzone są akcje do wykonania na drodze do realizacji celów.

Wykonanie zadania opisanego w korzeniu drzewa bądź poddrzewa może wymagać osiągnięcia wszystkich celów potomnych (rys. 2.2a) lub zrealizowania jednego spośród rozwiązań alternatywnych (rys. 2.2b) [158, 178]. W przypadku alternatywy zachowań, kolejność podejmowanych akcji wynika z kosztu możliwych rozwiązań – agent rozpoczyna od realizacji najmniej kosztowego zadania, a jeśli ono się nie powiedzie, wykonywane są kolejne.



(a) Sekwencja zachowań potomnych (b) Wybór zachowań potomnych

Rysunek 2.2: Wybrane strategie realizacji zachowań [158]

Drzewa zachowań pozwalają na intuicyjny opis złożonych scenariuszy postępowania ukierunkowanych na osiągnięcie założonego celu. Modelowanie odbywa się w kategoriach zrozumiałych przez człowieka, a wykorzystana struktura danych ułatwia śledzenie powiązań między akcjami. Ponadto strategia opisana przy pomocy drzewa zachowań nie musi być statyczna. Przykładowo, koszty węzłów alternatywnych mogą zmieniać się wraz ze stanem środowiska. Dodatkowo węzły drzewa mogą uwzględniać warunki określające, czy akcja powinna zostać rozpoczęta przy aktualnym stanie środowiska. Mechanizmy te pozwalają na reagowanie na zmiany zachodzące w otoczeniu [158]. Strategię zespołu agentów można opisać zbiorem drzew, które składają się na las zachowań.

Komponent decyzyjny agenta może również wykorzystywać metody sztucznej inteligencji, np. drzewo decyzyjne lub sieć neuronową do klasyfikacji odczytów z perceptorów. Podejmowane decyzje mogą opierać się na wynikach metaheurystyk, takich jak algorytm mrówkowy czy algorytm ewolucyjny. Szczególnie interesujące jest zastosowanie w komponencie decyzyjnym programowania

genetycznego. Umożliwia to konstrukcję agentów dążących do osiągnięcia celów, które są zgodne z kryteriami optymalizacji wykorzystanymi przez metaheurystykę.

2.3 Wybrane aspekty współpracy między agentami

Współpraca jest kluczową aktywnością pozwalającą na realizację celów agentów. Jest równie ważna w systemach rozproszonego rozwiązywania problemów *DPS*, w których wszystkie agenty dążą do osiągnięcia tego samego celu, jak i w systemach wieloagentowych *MA*, w których każdy z agentów może posiadać własne cele – nierzadko sprzeczne z dążeniami innych. Jeśli zadania agenta nie mogą zostać zrealizowane wyłącznie za pomocą jego własnych działań, pojawia się konieczność negocjacji w celu ustalenia zasad współpracy.

W systemach klasy *MA* agenty decydują, czy podjąć współpracę oraz jak ma ona przebiegać. W rozwiązaniach typu *DPS* przedmiotem negocjacji jest określenie zasad kooperacji, gdyż wszyscy uczestnicy dążą do rozwiązania to samego problemu. Negocjacje mogą dotyczyć dostępu do współdzielonych zasobów, rozlokowania komponentów w środowisku rozproszonym, czy podziału zadań pomiędzy agenty [56].

Zarówno w systemach *MA*, jak i *DPS* niektóre zachowania agentów uczestniczących w negocjacjach mają charakter kooperatywny – dotyczą tych samych celów, a współpraca może pozwolić na ich realizację w krótszym czasie lub przy mniejszym koszcie. Pozostałe zachowania są konkurencyjne i wymagają kompromisu w zakresie podziału zasobów, aby wymagania każdego agenta zostały spełnione [278]. W toku negocjacji agenty mogą podjąć decyzję o rozluźnieniu niektórych ograniczeń, jeśli umożliwi to spełnienie innych.

Kraus w [164] odniósł się do podstawowych aspektów i kryteriów oceny negocjacji pomiędzy agentami. Jednym z kluczowych parametrów jest czas negocjacji. Przedłużający się okres pertraktacji powoduje, że opóźnia się rozpoczęcie wykonania właściwych zadań. Może to w skrajnych przypadkach doprowadzić do niedopełnienia terminu ich realizacji. Preferowane są zatem krótkotrwałe negocjacje. Prostsze protokoły pozwalają skrócić czas negocjacji. Są one również łatwiejsze do implementacji.

Kolejnym kryterium jest efektywność negocjacji. Jest ona zazwyczaj rozumiana jako poziom satysfakcji negocjujących agentów z osiągniętego wyniku. Pożądane są protokoły wyznaczające rozwiązania *Pareto*-optymalne. W sytuacjach konfliktowych nie jest możliwe podniesienie poziomu zadowolenia żadnego z agentów, bez pogarszania poziomów zadowolenia dla innych agentów. Ponadto, jeśli istnieje choć jedno rozwiązanie lepsze dla uczestników negocjacji niż odstąpienie od współpracy, to negocjacje powinny zakończyć się sukcesem.

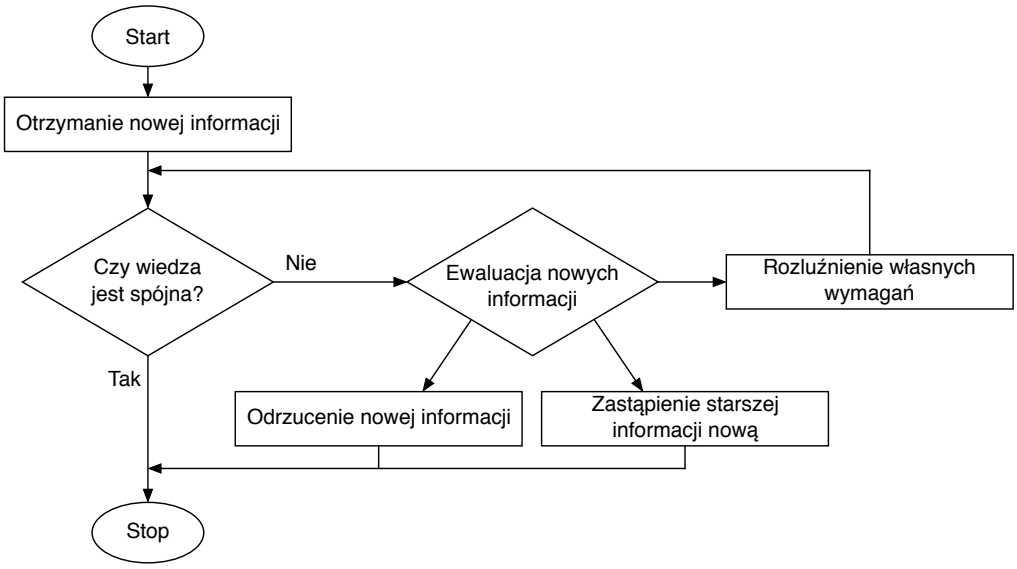
Istotnym kryterium jest stabilność modelu negocjacji. Jeśli nie istnieją przesłanki motywujące agenta do odstąpienia od przyjętej strategii, w sytuacji gdy wszystkie inne agenty jej przestrzegają, to może ona zostać określona jako stabilna. Innymi słowy, postępowanie zgodnie z wynegocjowanym rozwiązaniem jest najkorzystniejsze dla wszystkich uczestników. Nadużycia nie są w stanie zmienić wyniku negocjacji tak, aby podnieść poziom satysfakcji wybranego agenta.

Negocjacje można również klasyfikować pod kątem sposobu ich prowadzenia. Jeśli występuje w nich bezstronny podmiot pośredniczący w interakcjach pomiędzy pertraktującymi stronami, to określa się je jako mediacje. W przeciwnym przypadku uczestnicy komunikują się w sposób bezpośredni.

W przypadku systemów *DPS* współpraca jest ukierunkowana na rozwiązanie zadanego problemu bez centralnego planowania. Wszystkie agenty dążą do realizacji tego samego celu, jednak każdy z nich może posiadać dodatkowe indywidualne wymagania, co zazwyczaj prowadzi do konfliktów. Poprawa rozwiązania pod względem kryteriów jednego agenta może pogorszyć je pod względem kryteriów preferowanych przez inne [195]. Za pomocą rozproszonego planowania agent antycypuje rolę swoich lokalnych działań na drodze do realizacji globalnych celów, bez konieczności przekazywania globalnego stanu środowiska do pozostałych agentów [56].

Lander wskazuje, że konflikty wśród kooperatywnych agentów mogą być rezultatem różnej bądź niekompletnej wiedzy na temat środowiska, niewłaściwych założeń, czy odmiennych technik wyznaczania rozwiązań [169]. Ponadto agenty rozwiązujące problemy wielokryterialne, mogą uzyskiwać różne niezdominowane rozwiązania.

Lander zaproponował schemat przetwarzania nowych informacji przez agenta w celu wypracowania kompromisowego rozwiązania (rys. 2.3). Po otrzymaniu nowej informacji agent weryfikuje, czy jest ona spójna z jego aktualną wiedzą, a w szczególności z ograniczeniami, które nakłada on na akceptowane rozwiązania. Jeśli wiedza nie jest spójna, agent może zastąpić starszą informację nową, odrzucić nową informację lub rozluźnić swoje wymagania w celu uzyskania kompromisu [169].



Rysunek 2.3: Wstępne przetwarzanie nowych informacji przez agenta [169]

W przypadku systemów klasy *MA Rosenschein* i *Zlotkin* [230] proponują trzy obszary, w których mogą być prowadzone negocjacje. Pierwszy obszar obejmuje zastosowania agentów zorientowanych na zadania. W tym przypadku przedmiotem negocjacji jest takie rozlokowanie zadań w ramach dostępnych zasobów, które będzie korzystne dla wszystkich uczestników, np. z punktu widzenia

ograniczeń na czasy realizacji. Drugi obszar to systemy agentowe, w których kluczowe są zmiany wprowadzane w środowisku. Celem negocjacji jest dobór akcji modyfikujących stan systemu w sposób korzystny dla wszystkich agentów, przy uwzględnieniu indywidualnych ograniczeń każdego z nich. Ostatni obszar odnosi się do agentów zainteresowanych gromadzeniem dóbr. W tym przypadku celem jest maksymalizacja zysków agentów, a nie spełnienie ograniczeń.

W gridzie *Comcute* przedmiotem negocjacji jest wyłonienie grupy agentów warstwy pośredniczącej, które będą uczestniczyły w realizacji zadania obliczeniowego. Grid *Comcute* jest przykładem systemu klasy *DPS*. Celem agentów jest wykonania zleconych zadań obliczeniowych. Nowe zadanie jest przypisywane do jednego z agentów gridu, który odpowiada za uformowanie grupy roboczej. W tym celu rozsyła ofertę realizacji zadania do pozostałych agentów i oczekuje na zgłoszenia gotowości do rozpoczęcia pracy.

Odpowiedź agenta na ofertę zależy od jego aktualnego stanu. W przypadku agentów, które nie wykonują innych zadań, wysyłana jest odpowiedź z informacją o gotowości. Jeśli agent jest zaangażowany w realizację innych zadań, może zignorować nową ofertę lub kontynuować negocjacje. Agent zgłasza gotowość, jeśli przyjęcie nowego zadania nie spowoduje przekroczenia ograniczeń odnośnie obciążenia procesorów bądź pamięci operacyjnej. Możliwe jest również wysłanie zapytania o szczegóły zadania, które pozwolą ocenić, czy wymagania będą zachowane.

Agent, który zainicjował negocjacje, będzie je prowadził do momentu uformowania grupy roboczej o liczebności odpowiadającej wymaganemu stopniowi rozproszenia zadania obliczeniowego. Gdy to nastąpi, kolejne zgłoszenia są odrzucane. Negocjacje mogą zakończyć się niepowodzeniem, jeśli niedostateczna liczba agentów zgłosi gotowość do realizacji zadania.

2.4 Podstawowe założenia modelu środowiska

Istnieje wiele platform wspomagających wytwarzanie i wdrażanie agentów. Część z nich ma charakter platform ogólnego przeznaczenia, inne projektowane były z myślą o jednym obszarze badań bądź zastosowań [199]. Realizacje poszczególnych systemów opierają się na wspólnym zbiorze założeń, które składają się na typowy model środowiska agentowego.

Podstawowe założenie dotyczy modelu programistycznego do implementacji agentów. Większość spośród znaczących platform agentowych opiera się na obiektowych językach programowania [7]. *Kravari* i *Bassiliades* porównali 24 platformy agentowe będące w powszechnym użyciu zarówno w zastosowaniach naukowych, jak i przemysłowych w 2015 roku. Blisko 80% przeanalizowanych przez nich rozwiązań wykorzystuje język *Java* [166]. Na drugim miejscu uplasował się – również obiektowy – język *C++*, trzecim językiem był *C*.

Zbieżność między platformami obiektowymi a platformami agentowymi wynika z podobieństw w modelowaniu systemów. Podmioty z dziedziny rozwiązywanego problemu reprezentowane są przez agenty bądź obiekty cechujące się stanem w postaci pól danych. Posiadają one również zachowania wyrażone jako metody, które mogą być wywoływane przez inne obiekty funkcjonujące w wybranym systemie informatycznym [239].

Pomiędzy omawianymi podejściami istnieją jednak fundamentalne różnice. Obiekty są jednostkami oprogramowania o statycznych z reguły powiązaniach wyrażonych przez relacje dziedziczenia i kompozycji, a także w postaci wywołań metod w kodzie źródłowym aplikacji. Ponadto, jeśli interfejs klasy w programowaniu obiektowym zawiera określoną metodę, to zawsze możliwe jest jej użycie – obiekt nie może odmówić jej wywołania. Sterowanie jest kontrolowane zewnętrznie za pomocą wywołań metod przez inne obiekty [202].

W przypadku agentów zamiast bezpośrednich wywołań metod, następuje wymiana komunikatów. Agent może odmówić wykonania operacji lub całkowicie zignorować otrzymaną wiadomość. Powiązania mają charakter dynamiczny – na wyemitowany komunikat odpowiedzieć mogą różne agenty w zależności od aktualnego kontekstu. Agent może w wybranej sytuacji podjąć się realizacji zadania, a w innej z niego zrezygnować. Dodatkowo sterowanie kontrolowane jest wewnętrznie przez agenta w oparciu o jego cele. Innym przejawem wewnętrznego sterowania są działania proaktywne agentów [202]. Agenty są zatem autonomiczne, dynamicznie powiązane i zdolne do zmiany zachowań. Programowanie obiektowe dostarcza naturalnego sposobu reprezentacji agentów, ale obiekty nie będą agentami, dopóki nie zostaną im nadane odpowiednie cechy [179].

Ważnym aspektem platformy agentowej są wykorzystywane mechanizmy komunikacji. Broker wiadomości umożliwia przesyłanie informacji między agentami. Komponent tego typu występuje w platformach JADE [249], EMERALD [165], FLAME [237], czy Akka [254]. Zazwyczaj wykorzystywane są dwa katalogi pozwalające na adresowanie komunikatów. *Katalog osobowy* zawiera spis agentów, które występują w systemie i umożliwia adresowanie wiadomości do konkretnych odbiorców.

Z kolei *katalog branżowy* klasyfikuje agenty pod kątem ich możliwości i oferowanych usług. Pozwala to agentom-zleceniodawcom na wyszukiwanie usługodawców, zdolnych do zrealizowania określonego zadania. Mechanizm ten jest przydatny, gdy nie jest istotne, który agent wykona zlecenie oraz gdy zleceniodawca zamierza wybrać usługodawcę oferującego najkorzystniejsze warunki, np. najniższy koszt. Wybrane platformy agentowe oferują komunikację bez brokera wiadomości w modelu *peer-to-peer*, np.: INGENIAS Development Kit [107], MaDKit [112], czy Repast [167].

Ze względu na zastosowania agentów w heterogenicznych środowiskach, konieczny jest uniwersalny protokół komunikacyjny. Standard FIPA [79] definiuje zarówno protokół wymiany komunikatów [77], format przesyłanych wiadomości [76], jak i język opisu danych z możliwością definiowania własnych ontologii [78]. Do platform oferujących pełną zgodność ze standardem FIPA zaliczyć można środowiska: JADE, Jadex, JACK oraz EMERALD. Częściową zgodność oferują, m.in.: JAS, Jason, AGLOBE, Agent Factory, SeSAm i GAMA [166]. Standard FIPA obejmuje również specyfikację abstrakcyjnej architektury środowiska i zarządzania agentami [75, 80]. Specyfikacje FIPA definiują reguły konstrukcji kompletnej platformy agentowej [35]. Interoperacyjność z innymi platformami może zostać zapewniona poprzez zgodność ze standardem OMG MASIF [203], w którym opisano reguły interakcji pomiędzy systemami.

Mobilność jest istotną cechą agentów, która stawia specyficzne wymagania dla praktycznych realizacji systemów tej klasy. Konieczność zapewnienia mechanizmów migracji agentów jest jedną z przyczyn dużej popularności języka *Java* wśród platform agentowych [166]. Maszyna wirtualna

Javy (ang. *Java Virtual Machine – JVM*) jest dostępna dla popularnych systemów operacyjnych, a także dla urządzeń mobilnych z systemem *Android* [108, 207]. Kod źródłowy kompilowany jest do postaci pośredniej, która może być łatwo przenoszona między systemami. Dodatkowo mechanizmy serializacji obiektów umożliwiają migrację instancji agenta między maszynami wirtualnymi *Javy* na różnych komputerach [36]. Nie wymaga to użycia dodatkowych zewnętrznych bibliotek, zmniejszając narzut wprowadzany przez platformę wykonawczą [128].

Istotne założenia wynikają z wymagań bezpieczeństwa stawianych platformie agentowej. Dla negocjujących agentów zazwyczaj kluczowa jest poufność komunikacji, możliwość weryfikacji autentyczności wiadomości i ich niezaprzeczalność. Podstawowe mechanizmy obejmują szyfrowanie informacji wymienianych z brokerem komunikatów. Bardziej zaawansowane rozwiązania oferują bezpieczeństwo *end-to-end* pomiędzy komunikującymi się agentami. W tym podejściu dostęp do treści wiadomości ma jedynie nadawca i odbiorca – żadne inne agenty, ani nawet broker, nie są w stanie odczytać zawartości komunikatu. Wprawdzie fakt wymiany informacji może zostać odnotowany przez agenty, z którymi współdzielony jest kanał komunikacyjny, jednak treść pozostaje dostępna jedynie dla uprawnionych stron.

Uzyskanie takiego efektu wymaga szyfrowania z użyciem klucza współdzielonego między nadawcą a adresatem lub – w silniejszej odmianie – zastosowania kryptografii asymetrycznej. Szyfrowanie z użyciem pary kluczy: prywatnego i publicznego, eliminuje problem ustalania klucza współdzielonego pomiędzy komunikującymi się agentami. Pozwala to również na weryfikację wiarygodności przesyłanych komunikatów. Bezpieczeństwo *end-to-end* z wykorzystaniem kryptografii asymetrycznej oferują platformy Agent Factory, EMERALD i JADE [37]. Z kolei środowisko JADEX oferuje mechanizmy bezpieczeństwa oparte o klucz współdzielony [166].

Ważną cechą platformy agentowej jest możliwość zarządzania zaufaniem. W decyzjach odnośnie rozpoczęcia współpracy między agentami kluczową rolę odgrywa przyjęta miara zaufania do innych agentów [111, 228]. Modele definiujące miary zaufania oferuje platforma EMERALD. Dodatkowo posiada ona mechanizmy do oceny efektów negocjacji z punktu widzenia interesów zespołu agentów i interesów wybranych uczestników [1, 166].

Ze względu na opisane zalety platformy EMERALD rozwiązanie to wydaje się być najbardziej efektywne w zastosowaniach do systemów rozproszonych. Z drugiej strony, istniejące gridy, takie jak system *Comcute*, zazwyczaj nie są przystosowane do wprowadzenia oprogramowania klasy EMERALD bez istotnych modyfikacji w istniejącym kodzie źródłowym warstwy pośredniczącej. W tej sytuacji alternatywnym podejściem jest implementacja agentów rozszerzających możliwości aktualnych modułów programistycznych gridu. W systemie *Comcute* zrealizowano to poprzez hermetyzację dotychczasowych modułów w obrębie maszyn wirtualnych, które wyposażono w dodatkowe oprogramowanie o cechach agentowych. Pod pojęciem agenta w gridzie *Comcute* rozumie się zatem maszynę wirtualną wyposażoną w oprogramowanie warstwy pośredniczącej i agentowe zachowania, które umożliwiają koordynację pracy systemu. Środowiskiem działania agentów jest chmura prywatna *OpenStack* skonfigurowana na serwerach systemu *Comcute*.

2.5 Zastosowania systemów agentowych

Systemy agentowe stosowane są w obliczeniach rozproszonych [11], systemach internetowych [46], grach komputerowych [138, 182], sztucznej inteligencji [240], robotyce [252], zdalnym nauczaniu [250], systemach finansowych i bankowych [140], a także w infrastrukturach inteligentnych miast, regionów i państw [54]. Agenty programistyczne są wykorzystywane do modelowania złożonych systemów i zachodzących w nich interakcji.

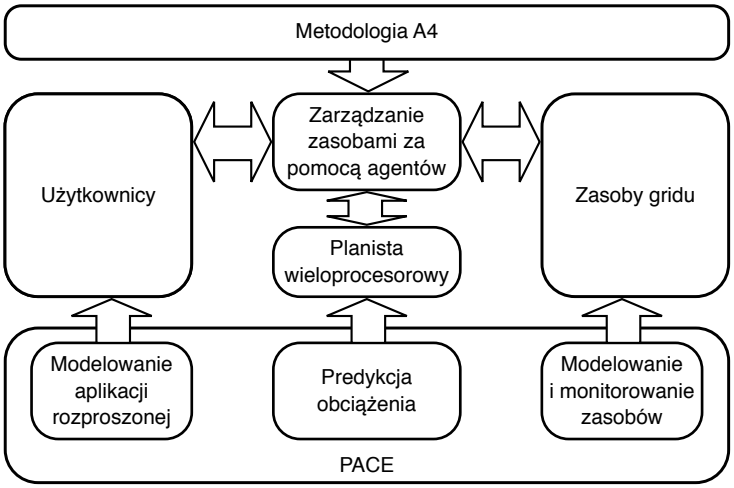
Metodologia A4 (ang. *Agile Architecture and Autonomous Agents*) jest dedykowana do budowy skalowalnych systemów rozproszonych. W tym podejściu zasoby i funkcjonalności są reprezentowane jako usługi oferowane przez *agentów-usługodawców*. Usługa może polegać na wykonaniu obliczeń z wykorzystaniem karty graficznej albo na dostępie do danych zapisanych w węźle bazodanowym. *Agenty-konsumenci* wywołują usługi, aby zrealizować zadania użytkowników. Agent może być jednocześnie konsumentem i usługodawcą, co pozwala na budowanie złożonych hierarchii wykonania zadań [49].

Koordynacja systemu rozproszonego odbywa się poprzez interakcje między agentami. Usługodawcy wykorzystują mechanizm rozgłaszania usług do publikowania informacji o oferowanych możliwościach i parametrach wykonania, które wynikają z zasobów przypisanych agentowi, takich jak: akcelerator obliczeniowy, pamięć na dysku półprzewodnikowym SSD (ang. *Solid-State Drive*), czy urządzenia zewnętrzne. Z kolei konsumenci posługują się mechanizmem wyszukiwania usług, aby odnaleźć te, które najlepiej odpowiadają wymaganiom zadania [48]. Skonstruowaną w ten sposób agentową warstwę pośredniczącą można wykorzystać do realizacji różnorodnych zadań obliczeniowych. Programistyczne agenty są w tym przypadku uniwersalną warstwą abstrakcji dla różnych zasobów i umożliwiają integrację heterogenicznych systemów [49].

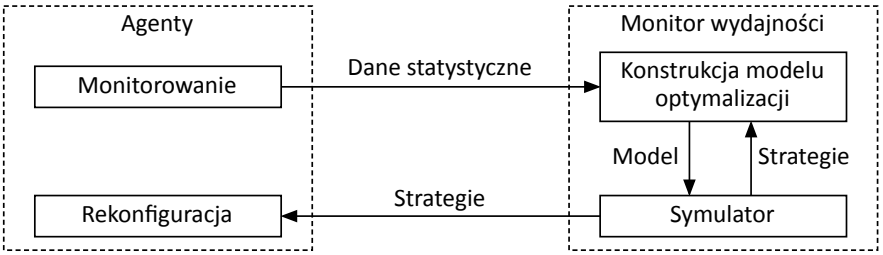
Metodologia A4 wykorzystuje narzędzie PACE [47] do predykcji obciążenia gridu. Użytkownik zlecając zadanie obliczeniowe dołącza model wykonania zadania w środowisku rozproszonym. Z kolei *agenty-usługodawcy* specyfikują model dostępnych zasobów gridu. W oparciu o te informacje odbywa się predykcja obciążeń systemu. Agenty wykorzystują oszacowane obciążenia przy podejmowaniu decyzji odnośnie ulokowania zasobów. Na rysunku 2.4 zaprezentowano wykorzystanie metodologii A4 do zarządzania zasobami gridu.

Praca agentów zarządzających może być optymalizowana za pomocą doradczego monitora wydajności PMA (ang. *Performance Monitor and Advisor*), który przedstawiono na rysunku 2.5. Parametry agentów zarejestrowane w trakcie pracy systemu są wykorzystywane do budowy modelu optymalizacji. Następnie za pomocą symulacji wyznacza się nowe strategie zarządzania gridem. Etap symulacji może być powtarzany wielokrotnie aż do uzyskania rozwiązań o pożądanych własnościach, np. konfiguracji cechującej się zadanym poziomem niezawodności. Uzyskane strategie są wykorzystywane do rekonfiguracji systemu rozproszonego [49].

Inteligentne agenty wykorzystywane są również w analizie *Big Data*. *Twardowski* i *Ryżko* zaproponowali model przetwarzania oparty o architekturę lambda [253]. Podejście to umożliwia zarówno uwzględnienie strumieni danych napływających w czasie rzeczywistym, jak i wsadowe przetwarzanie



Rysunek 2.4: Zarządzanie zasobami za pomocą agentów [49]

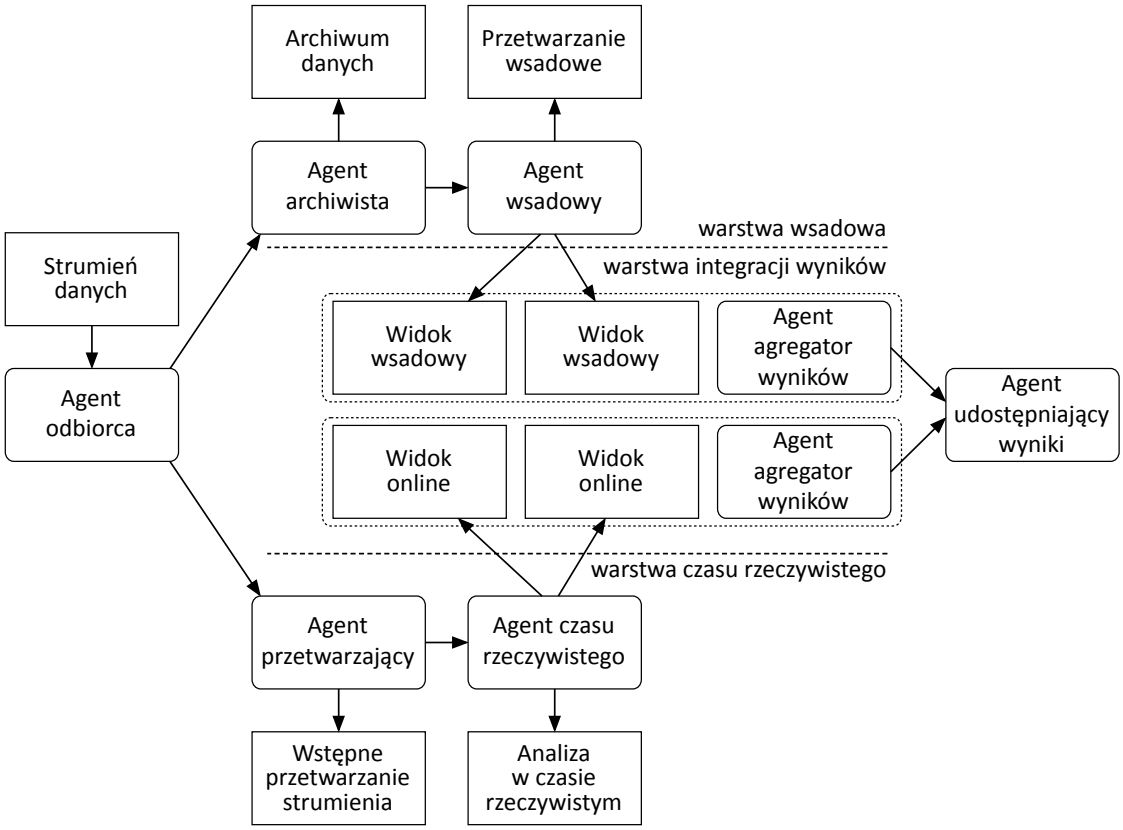


Rysunek 2.5: Doradczy monitor wydajności PMA [49]

danych zapisanych uprzednio w pamięci masowej [186]. Agenty w warstwie pośredniczącej stosowane są do integracji wielu narzędzi wykorzystywanych na różnych etapach pozyskiwania, przetwarzania i archiwizowania danych. Z kolei w analizie zgromadzonych danych uczestniczą agenty posługujące się metodami sztucznej inteligencji, np. maszynami wektorów nośnych [31, 50, 153]. Na rysunku 2.6 przedstawiono przetwarzanie *Big Data* z wykorzystaniem agentów. Model może zostać zastosowany również w gridach opartych o wolontariat obliczeniowy [155].

Systemy agentowe posiadają wiele aplikacji w dziedzinie finansów. Obejmują one modelowanie i symulację zdarzeń zachodzących na rynkach kapitałowych, przewidywanie trendów, a także automatyczne realizowanie transakcji w imieniu inwestora [141, 211, 241]. Wykorzystywane architektury pozwalają agentom na podejmowanie decyzji przy niepełnej wiedzy na temat otoczenia oraz na funkcjonowanie wśród innych niekooperatywnych agentów. Cechy te odpowiadają realiom rynków finansowych, na których działa wiele konkurujących ze sobą podmiotów dążących do maksymalizacji zysku. Równocześnie duża liczba istotnych czynników i krótki czas na podjęcie decyzji uniemożliwiają wyczerpującą analizę wszystkich informacji [144].

W sytuacji, gdy szybkość reakcji na zachodzące zdarzenia jest kluczowa dla powodzenia transakcji, komputerowy agent ma przewagę nad inwestorem. Z tego względu inteligentne agenty wprowadza się do systemów inwestycyjnych. W oparciu o przewidywania odnośnie przyszłych trendów



Rysunek 2.6: Agentowe przetwarzanie *Big Data* w czasie rzeczywistym w architekturze lambda [253]

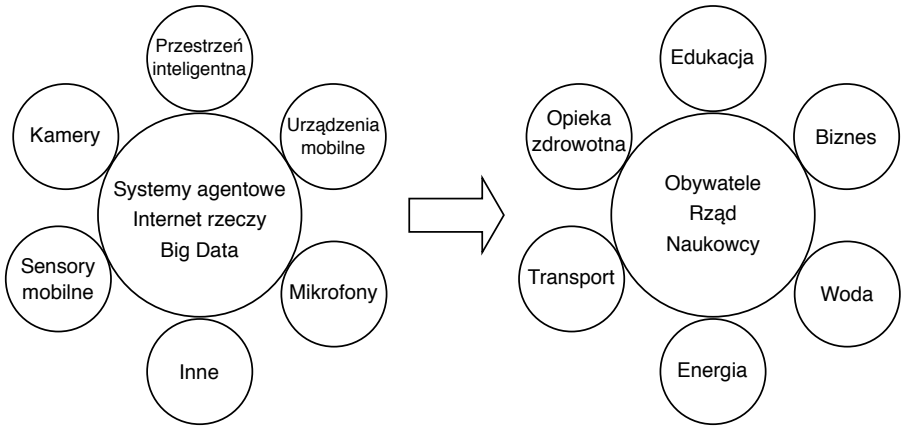
agenty realizują zautomatyzowane transakcje i są zdolne do wykonywania setek operacji finansowych na sekundę [211]. Przy tak intensywnej działalności inwestycyjnej istnieje jednak duże ryzyko strat w przypadku awarii komponentu decyzyjnego. W 2012 roku firma *Knight Capital Group* poniosła na nowojorskiej giełdzie straty na poziomie 440 milionów dolarów w ciągu 30 minut. System inwestycyjny wykonał ponad 4 miliony błędnych transakcji zanim zdołano go wyłączyć [215].

Agenty stosowane są także w *e-nauczaniu*. Masowość kształcenia uniemożliwia indywidualne podejście nauczyciela do każdego odbiorcy, co przekłada się na pogorszenie efektywności nauczania. Rolę konsultanta pełnić może inteligentny agent, który dostosowuje zakres materiału do indywidualnych potrzeb odbiorcy [143]. W oparciu o dotychczasowe postępy studenta agent może wyznaczyć kolejne tematy do zrealizowania lub wskazać zagadnienia wymagające powtórzenia [87]. Komputerowy agent jest dostępny dla odbiorców o dowolnej porze i może zostać w łatwy sposób zwielokrotniony w miarę naboru nowych studentów [2].

Zarówno studenci, jak i wspomagające ich agenty mogą być reprezentowani przez awatary w wirtualnym środowisku edukacyjnym niczym gracze w grze komputerowej. Oprogramowanie *Open Wonderland* [206] umożliwia kreowanie środowisk edukacyjnych, w których studenci współpracują, aby wykonać eksperymenty przewidziane w programie nauczania. Grywalizacja w edukacji aktywizuje odbiorców, wprowadza elementy pracy zespołowej i rywalizacji, a także zwiększa satysfakcję z nauki, redukując poczucie obowiązku [126, 130].

Inteligentny agent może wspomagać również nauczyciela poprzez raportowanie postępów studentów. Analizując indywidualne wyniki każdego z nich, agent może wskazać tych, którzy wymagają szczególnej uwagi. Z kolei w oparciu o efekty pracy całej grupy określić można zagadnienia, które należy omówić bardziej szczegółowo [143].

Ważnym obszarem zastosowań systemów agentowych jest oprogramowanie wykorzystywane w ramach infrastruktury informatycznej inteligentnych miast (ang. *smart cities*). W systemach tego typu konieczna jest integracja wielu heterogenicznych komponentów, m.in.: geograficznie rozproszonych czujników, urządzeń mobilnych, układów automatyki przemysłowej, kamer czy urządzeń telekomunikacyjnych (rys. 2.7). Wszystkie składowe wykorzystują charakterystyczne dla siebie protokoły komunikacyjne, formaty danych i kanały transmisyjne. Inteligentne agenty mogą pełnić rolę uniwersalnej warstwy integracji dla różnorodnych komponentów systemu [54, 174].



Rysunek 2.7: Zastosowanie systemów agentowych w inteligentnych miastach [145]

Funkcje oferowane przez czujniki i inne urządzenia składające się na infrastrukturę inteligentnego miasta mogą być udostępniane w postaci usług agentów. Dzięki temu dostęp do nich odbywa się w oparciu o standardowe mechanizmu rozgłaszania i wyszukiwania usług. Prowadzi to do przetwarzania w architekturze XaaS (ang. *everything-as-a-service*), w której każdy zasób i informacja jest usługą możliwą do wykorzystania przez agenty funkcjonujące w systemie [93, 274]. Komunikacja odbywa się z wykorzystaniem ontologii, która opisuje relacje pomiędzy usługami i określa sposoby dostępu do nich.

2.6 Wnioski i uwagi

Intensywny rozwój sztucznej inteligencji umożliwia delegację zadań wykonywanych wcześniej wyłącznie przez ludzi na inteligentne agenty programistyczne. Inteligentne agenty cechują się zdolnością adaptacji do zmieniających się warunków. Przez działania reaktywne agenty decydują, w jaki sposób zareagować na zachodzące w ich otoczeniu zdarzenia. Dodatkowo agentom przypisuje się zdolność do działań proaktywnych. Nie stanowią one odpowiedzi na zewnętrzne bodźce, lecz są efektem dążenia do osiągnięcia celów agenta.

Wyróżnia się systemy klasy *DPS* do rozproszonego rozwiązywania problemów, w których wszystkie agenty dążą do tych samych celów i aby je osiągnąć, koordynują działania. Koordynacja agentów ma na celu formowanie wirtualnych organizacji, które zrzeszają agenty wspierające się w realizacji zadań. W efekcie oczekiwane rezultaty można osiągnąć w krótszym czasie lub przy mniejszym koszcie. Z kolei w systemach klasy *MA* agenty mogą posiadać różne cele i konkurować o dostęp do zasobów.

Komponent decyzyjny agenta zazwyczaj wykorzystuje metody sztucznej inteligencji, np. drzewo decyzyjne lub sieć neuronową. Podejmowane decyzje mogą opierać się na wynikach metaheurystyk, takich jak algorytm mrówkowy czy algorytm ewolucyjny. Szczególnie interesujące jest zastosowanie w komponencie decyzyjnym programu, który zaimplementowano automatycznie za pomocą programowania genetycznego. Skonstruowany w ten sposób agent wykonuje akcje, które prowadzą do optymalizacji wybranych aspektów jego pracy zgodnie z kryteriami optymalizacji przyjętymi w programowaniu genetycznym.

Inteligentne agenty wykorzystuje się do wspomagania pracy systemów finansowych. Komputerowy agent ma przewagę nad inwestorem pod względem szybkości realizowanych transakcji i zdolności do analizy danych. Agenty stosowane w systemach zdalnego nauczania wspomagają nauczycieli oraz aktywizują studentów, co prowadzi do poprawy efektów kształcenia. Inne ważne obszary zastosowań obejmują gry komputerowe, infrastruktury inteligentnych miast czy robotykę. Agenty mogą również pełnić funkcję uniwersalnej warstwy integracji dla heterogenicznych systemów. Zasoby reprezentowane są w tym przypadku jako usługi agentów. Dostęp do nich odbywa się w oparciu o standardowe mechanizmy publikacji usług i ich konsumpcji.

Rozwiązania agentowe stosowane są również w warstwie pośredniczącej systemów rozproszonych, w tym gridów komputerowych. System *Comcute* zmodernizowano do postaci grida agentowego. Zastosowano mechanizmy negocjacji do wyznaczania grup roboczych odpowiedzialnych za realizację zadań. Ponadto umożliwiono migrację agentów pomiędzy węzłami systemu. Oprócz agentów zarządzających i agentów dystrybuujących dane i zadania do zasobów obliczeniowych, można wyróżnić *agenty-solwery* do optymalizacji strategii zespołu agentów warstwy pośredniczącej gridu przy uwzględnieniu nałożonych ograniczeń.

Odpowiedni dobór strategii sieci agentów zarządzających pozwala na uzyskanie pożądanych cech systemu, np. wymaganego poziomu niezawodności lub kosztu realizacji zadań. W szczególności programowanie genetyczne może dokonać implementacji kompaktowego programu, który zostanie wykorzystany w komponencie decyzyjnym agentów warstwy pośredniczącej. Dzięki temu agenty mogą autonomicznie przemieszczać się w gridzie i decydować o rozdziale zasobów. Dane wejściowe dla *agentów-solwerów* aktualnie specyfikuje administrator systemu, przy czym wskazane jest zastąpienie jego pracy przez zespół agentów zbierających dane. Istotnym przyszłościowym rozszerzeniem gridu *Comcute* jest implementacja agentów zarządzających zasobami sprzętowymi, które zastąpią administratora przy rekonfiguracji systemu.

Rozdział 3

Modelowanie agentowego systemu rozproszonego

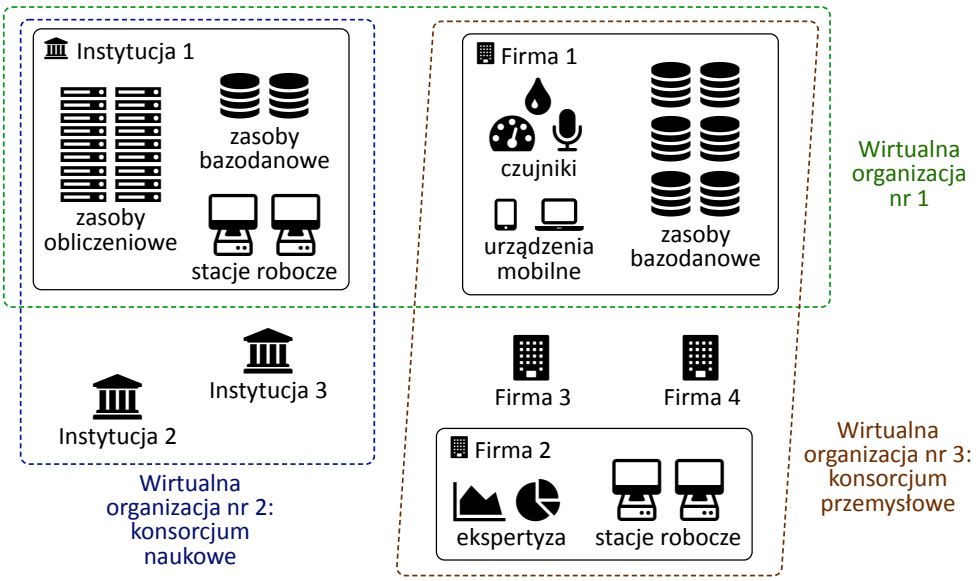
3.1 Charakterystyka wybranych gridów

Zespoły inteligentnych agentów funkcjonują w różnorodnych systemach rozproszonych ze szczególnym uwzględnieniem gridów komputerowych. Pierwsze gridy powstały w połowie lat dziewięćdziesiątych XX wieku i były wykorzystywane w problemach naukowych wymagających dużych mocy obliczeniowych [84]. Połączenie superkomputerów z różnych placówek badawczych pozwoliło na budowę systemu cechującego się wydajnością, jaka nie była osiągalna dla żadnego z udziałowców z osobna. Każdy z zaangażowanych podmiotów udostępniał część swoich zasobów, a w zamian uzyskiwał możliwość korzystania z tych, które zaoferowali inni [275].

Koncepcja gridu komputerowego odnosi się do problemu współdzielenia i koordynacji rozproszonych zasobów należących do niezależnych podmiotów, które zrzeszają się w wirtualne organizacje realizujące wspólne cele [84]. Zasoby obejmować mogą nie tylko moc obliczeniową, ale również pamięć dyskową i operacyjną, specjalistyczne urządzenia, czujniki, dane oraz oprogramowanie, w tym usługi [123]. Warto zauważyć, że jeden podmiot może uczestniczyć w wielu wirtualnych organizacjach, co pokazano na rysunku 3.1.

Charakterystycznym wymaganiem dla systemów typu grid jest konieczność integracji heterogenicznych komponentów i zarządzania dynamicznymi powiązaniami między nimi. Dynamika gridu wynika z jego rozproszonego charakteru oraz z modelu wirtualnych organizacji, których formowanie umotywowane jest potrzebami udziałowców [58]. Choć podmioty udostępniają zasoby pozostałym uczestnikom gridu, to każdy z nich posiada lokalną autonomię [277].

Zarządzanie składowymi odbywa się w wielu domenach. Zasoby poszczególnych podmiotów zazwyczaj znajdują są w różnych lokalizacjach geograficznych, a połączenia pomiędzy nimi



Rysunek 3.1: Podmioty zrzeszone w wirtualnych organizacjach
Źródło: opracowanie własne na podstawie [84]

nie wykorzystują dedykowanych kanałów transmisyjnych, lecz ogólnodostępne łącza telekomunikacyjne [168]. Kluczowe dla efektywnej pracy gridu jest oprogramowanie warstwy pośredniczącej (ang. *middleware*), które integruje komponenty systemu oraz koordynuje dostęp do zasobów [58, 176].

Dzięki zastosowaniu warstwy pośredniczącej zleceniodawca obliczeń nie musi znać szczegółów interakcji pomiędzy składowymi. Zamiast tego postrzega grid jako wirtualny komputer posiadający zasoby o określonych parametrach [277]. Komunikacja w obrębie warstwy pośredniczącej odbywa się zazwyczaj w oparciu o standardowe protokoły (np. *SOAP* [271]) z wykorzystaniem uniwersalnych formatów danych, takich jak *XML* [272] i *JSON*. Dzięki temu możliwa jest integracja różnych systemów operacyjnych, platform wykonawczych i aplikacji [159].

Gridy komputerowe istotnie różnią się od innych systemów przetwarzania rozproszonego. W przypadku klastrów obliczeniowych węzły są zgromadzone w tej samej lokalizacji i połączone dedykowanymi sieciami typu *InfiniBand*. W chmurze obliczeniowej wiele podmiotów wykorzystuje te same rozproszone zasoby, jednak każdy z nich realizuje własne cele, a infrastruktura jest zarządzana przez jednego usługodawcę. Z kolei w systemach klasy *B2B* (ang. *business-to-business*) udostępniane są zazwyczaj informacje i usługi, a nie zasoby komputerów pozwalające na skoordynowane przetwarzanie danych w wielu lokalizacjach [84].

Systemy klasy grid scharakteryzować można jako otwarte, współbieżne i przezroczyste dla użytkowników platformy współdzielenia zasobów w wirtualnych organizacjach [222]. Równoległe przetwarzanie danych odbywa się z wykorzystaniem wielu węzłów obliczeniowych dla przyspieszenia realizacji zadań. Zastosowanie otwartych standardów ułatwia przyłączanie nowych węzłów w celu zwiększenia wydajności. Grid może być rozszerzony o nowe usługi za pomocą modernizacji sprzętu lub oprogramowania, a także poprzez dołączanie nowych podmiotów do systemu [229].

Pakiet *Globus Toolkit* [104] jest przykładem oprogramowania warstwy pośredniczącej dla gridów komputerowych. Zawiera moduły wspomagające realizację zadań obliczeniowych, dostęp do danych, wyszukiwanie i monitorowanie usług oraz bezpieczeństwo systemu. W oprogramowaniu zaimplementowano mechanizmy poufnej wymiany informacji oraz mechanizmy uwierzytelniania i autoryzacji. Pakiet obejmuje również narzędzia ułatwiające implementację aplikacji rozproszonych [217].

Dostęp do danych odbywa się w oparciu o protokół *GridFTP*, który zoptymalizowano pod kątem wydajności i bezpieczeństwa. Umożliwia on efektywny transfer danych, które rozproszono między węzłami systemu. Poszczególne fragmenty plików mogą być wysyłane z różnych lokalizacji. W protokole *GridFTP* są realizowane współbieżne operacje wejścia/wyjścia. Dzięki temu dla łącza o teoretycznej przepustowości na poziomie 30 Gb/s uzyskano efektywną wydajność transferu 27,3 Gb/s, podczas którego dane były wysyłane z pamięci operacyjnej nadawcy do pamięci operacyjnej odbiorcy. W przypadku odczytu z dysku i zapisu na dysk odbiorcy uzyskano wydajność 17 Gb/s [8].

Za uruchamianie zadań i monitorowanie ich działania odpowiada moduł GRAM (ang. *Grid Resource Allocation and Management*). W trakcie realizacji zadania wykonywany jest transfer danych wejściowych do węzłów obliczeniowych, przetwarzanie danych, cykliczne weryfikowanie postępu i jego raportowanie do zleceniodawcy oraz transfer wyników po zakończeniu wszystkich operacji. Natomiast *Zarządcy Zasobów Lokalnych* (ang. *Local Resource Manager* – LRM) monitorują obciążenie komputerów i przyjmują zadania do uruchomienia od modułu GRAM [105].

Komponent LRM zarządza wykorzystaniem CPU, kart graficznych z obsługą technologii CUDA [45] lub OpenCL [133], akceleratorów obliczeniowych oraz innych urządzeń, w które wyposażono węzeł gridu. W związku z tym *Zarządca Zasobów Lokalnych* jest związany z architekturą komputera, na którym go ulokowano. Z kolei moduł GRAM stanowi warstwę abstrakcji dla wielu różnorodnych zasobów dostępnych w gridzie. Obsługuje kilka standardowych implementacji komponentu LRM oraz umożliwia wykorzystanie dodatkowych adapterów, które pozwalają na integrację ze specyficznymi zasobami [105].

Pakiet *Globus Toolkit* zastosowano w wielu projektach e-nauki. Wykorzystano go w *European DataGrid*, który projektowano pod kątem gromadzenia danych pochodzących z *Wielkiego Zderzacza Hadronów* (ang. *Large Hadron Collider*) [268]. Omawiane oprogramowanie zastosowano także w inicjatywach, takich jak *Earth System Grid* [69] oraz *Network for Earthquake Engineering and Simulation* [258]. Pakiet *Globus Toolkit* był wykorzystywany również przez firmy, takie jak: *Hewlett-Packard*, *IBM* i *Oracle* [104].

Ważną grupę wśród gridów komputerowych stanowią systemy oparte o wolontariat obliczeniowy. W tym przypadku podmiotami budującymi wirtualną organizację są wolontariusze oferujący moc obliczeniową komputerów oraz koordynator zadań, który dysponuje modułami obliczeniowymi i danymi do przetwarzania. Gridy wolontariackie cechują się dużą dynamiką, ponieważ uczestnicy obliczeń mogą wielokrotnie podłączać i odłączać komputery w różnych odstępach czasu [29].

W przypadku obliczeń wolontariackich minimalizuje się liczbę kroków potrzebnych do udostępnienia mocy obliczeniowej. Wiele projektów akceptuje anonimowych uczestników, co przekłada się na charakterystyczne wymagania w zakresie wiarygodności obliczeń. Dane wejściowe są zazwyczaj replikowane i przesyłane do kilku wolontariuszy. Porównanie uzyskanych wyników pozwala określić poziom ich wiarygodności. W przypadku rozbieżności te same dane wejściowe są ponownie replikowane, a obliczenia – powtarzane [29]. Mechanizmów tych nie stosuje się w gridach zrzeszających firmy i istytucje [129]. W tym przypadku węzły obliczeniowe są pod kontrolą wiarygodnych podmiotów, które gwarantują poprawność rezultatów obliczeń i symulacji realizowanych w gridzie [119].

Infrastruktura *BOINC* (ang. *Berkeley Open Infrastructure for Network Computing*) jest wykorzystywana jako warstwa pośrednicząca w kilkudziesięciu projektach opartych o wolontariat obliczeniowy. Jednym z pierwszych systemów tego typu, który uzyskał popularność, jest projekt *SETI@home* [257]. Jego zadaniem jest analiza szumu radiowego rejestrowanego z kosmosu w celu identyfikacji sygnałów, które mogły zostać wyemitowane przez cywilizacje pozaziemskie.

Ważnym projektem wykorzystującym oprogramowanie *BOINC* jest *Einstein@Home* [9]. W tym przypadku analizowane są dane rejestrowane przez wykrywacz fal grawitacyjnych *LIGO*, teleskop satelitarny *Fermi GST* oraz największy na świecie radioteleskop w obserwatorium *Arecibo*, w celu wykrywania pulsarów. Wykorzystując moc obliczeniową udostępnioną przez wolontariuszy, odkryto istnienie około 50 nieznanych wcześniej pulsarów [9].

Najwyższą wydajnością wśród gridów wolontariackich cechuje się projekt *Folding@Home* [210] prowadzony na *Stanford University*. We wrześniu 2016 roku dysponował on mocą obliczeniową na poziomie 86,8 PFLOPS (ang. *FLoating point Operations Per Second*). W projekcie realizuje się symulacje zmian zachodzących w strukturach białek. Jego rezultaty mogą przyczynić się do opracowania leków na choroby uznawane dotychczas za nieuleczalne [210].

Z kolei projekt GIMPS (ang. *Great Internet Mersenne Prime Search*) [191] ukierunkowano na wyszukiwanie dużych liczb pierwszych. W styczniu 2016 roku odkryto 49 liczbę pierwszą *Mersenne'a* o wartości $2^{74\,207\,281} - 1$. Jej zapis w systemie dziesiętnym wymaga 22 338 618 cyfr. Test *Lucasa* weryfikujący, czy liczba jest pierwsza wymagał miesiąca obliczeń na komputerze wyposażonym w procesor Intel Core i7-4790 [191]. Projekt GIMPS wykorzystuje oprogramowanie *Entropia* do dystrybucji danych i zarządzania zasobami gridu [52].

Architektura systemu *BOINC* opiera się na modelu klient-serwer. Aplikacje obliczeniowe oraz *jednostki robocze* (ang. *workunit*) umieszczane są na serwerach. *Jednostka robocza* definiuje dane wejściowe do przetworzenia i wymagania, takie jak: moc obliczeniowa, pamięć operacyjna i dyskowa oraz oczekiwany czas uzyskania wyników. Wolontariusze instalują na komputerach oprogramowanie klienckie, które umożliwia wybór, w jakich godzinach moc obliczeniowa będzie udostępniana, ile przestrzeni dyskowej i pamięci RAM może zostać wykorzystane i w jakim stopniu obciążyć można interfejsy sieciowe komputera [15].

Po dokonaniu konfiguracji wolontariusz może przystąpić do obliczeń. Oprogramowanie klienckie komunikuje się z *serwerem nadzorującym wykonanie zadań* (ang. *scheduler server*) i przesyła parametry pracy określone przez wolontariusza. Serwer wybiera *jednostkę roboczą*, która spełnia zadane

ograniczenia i odsyła ją do klienta. *Jednostka robocza* zawiera informacje o lokalizacji danych wejściowych oraz parametry startowe dla aplikacji obliczeniowej. Dane pobierane są z *serwera danych* (ang. *data server*), po czym rozpoczynają się obliczenia na komputerze wolontariusza. Obsługa *jednostki roboczej* może zająć od kilku sekund do kilkuset godzin w zależności od wybranego projektu obliczeniowego. Obliczone wyniki odsyłane są do *serwera nadzorującego wykonanie zadań* [15].

W celu zapewnienia wiarygodności wyników *serwer nadzorujący wykonanie zadań* wykorzystuje komponent *walidator* (ang. *validater*), który porównuje wyniki uzyskane przez różnych wolontariuszy i określa ich zgodność. Komponent *tranzycjoner* (ang. *transitioner*) odpowiada za replikację obliczeń w przypadku rozbieżnych rezultatów. Z kolei *asymilator* zapisuje wyniki w bazie danych projektu, gdy uda się potwierdzić ich wiarygodność. Ryzyko nadużyć jest minimalizowane poprzez wysyłanie *jednostki roboczej* tylko raz do danego wolontariusza [15].

Infrastruktura systemu *BOINC* może być skalowana zarówno na poziomie *serwerów nadzorujący wykonanie zadań*, jak i *serwerów danych*. W przypadku niedostępności serwerów czas do ponowienia próby nawiązania połączenia rośnie w sposób wykładniczy wraz z każdym niepowodzeniem. Redukuje to ryzyko przeciążenia serwerów przywróconych do pracy po okresowej awarii. Schemat wykładniczego zwiększania czasu oczekiwania wykorzystywany jest również w przypadku awarii aplikacji obliczeniowej na komputerze wolontariusza [15].

Polska Infrastruktura Gridowa [139] obejmuje zasoby instytucji naukowych ulokowanych w Gdańsku (CI TASK), Krakowie (ACK CYFRONET AGH), Poznaniu (PCSS), Warszawie (ICM UW) i Wrocławiu (WCSS). Oferuje łączną moc obliczeniową na poziomie 3,642 PFLOPS, przestrzeń dyskową o rozmiarze 2,25 PB oraz 53 tys. rdzeni obliczeniowych. Infrastruktura obejmuje klastry obliczeniowe udziałowców projektu *PL-Grid*, który zrealizowano w latach 2009-2012. Kolejne projekty *PLGrid Plus* (2011-2014) i *PLGrid NG* (2014-2015) umożliwiły rozbudowę gridu o dedykowane usługi obliczeniowe i narzędzia dziedzinowe dla strategicznych obszarów badań naukowych [139].

Oferta projektu *PL-Grid* dla środowisk naukowych obejmuje: obliczenia ogólnego przeznaczenia na klastrach wysokiej wydajności, przetwarzanie w chmurze z wykorzystaniem maszyn wirtualnych, dostęp do dedykowanych usług obliczeniowych oraz specjalistyczne pakiety oprogramowania. Grid pozwala na pracę z aplikacjami stosowanymi w badaniach z zakresu biologii (np. *AutoDock*, *Gromacs*), fizyki (np. *OpenFOAM*, *ANSYS Fluent*), chemii kwantowej (np. *MOPAC*, *CFOUR*) czy obliczeń numerycznych (np. *MATLAB*, *R*). Oprogramowanie uruchamiane jest w węzłach gridu, po czym użytkownicy mogą korzystać z niego, posługując się narzędziami do zdalnej pracy [139].

Oprócz omówionych powyżej rozwiązań stosowanych jest wiele innych platform gridowych. Oprogramowanie *gLite* [73] było wykorzystywane w ramach inicjatywy *EGEE* (ang. *Enabling Grids for E-sciencE*). Z kolei projekt *European Middleware Initiative* (*EMI*) integruje rozwiązania wielu dostawców, w tym oprogramowanie *gLite* oraz pakiety *ARC* [200], *UNICORE* [256] i *dCache* [66]. Projekt *EMI* dostarcza rozwiązań na potrzeby *Europejskiej Infrastruktury Gridowej* [71], w której istotnym komponentem jest również *PL-Grid*.

3.2 Charakterystyka gridu *Comcute*

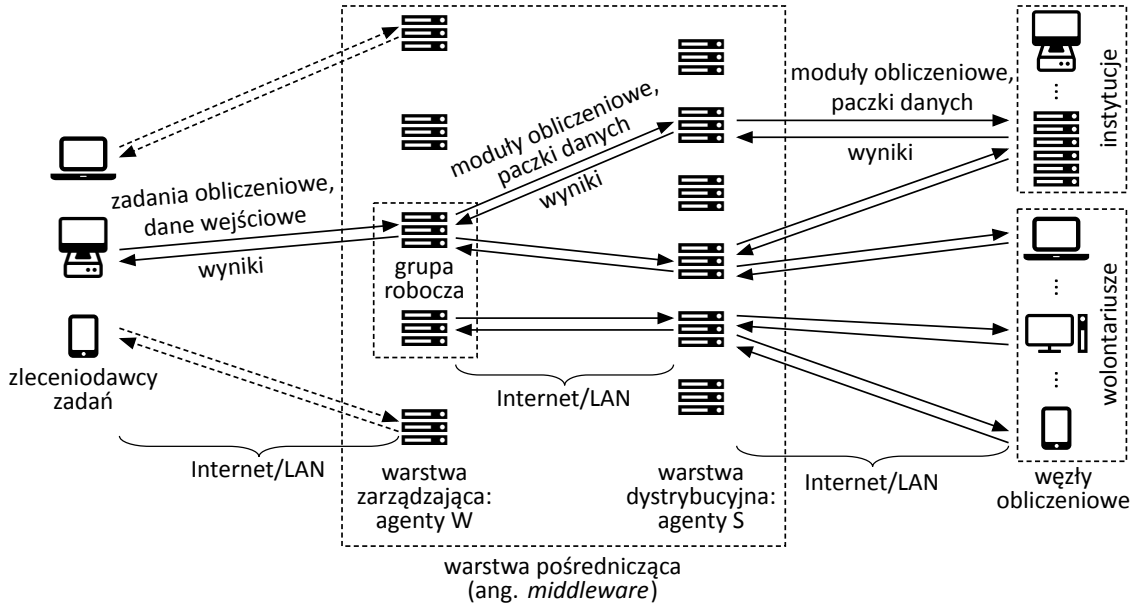
Grid *Comcute PG* [131] jest eksperymentalnym systemem do obliczeń rozproszonych, w którym wykorzystuje się moc obliczeniową wolontariuszy. System opracowano na Politechnice Gdańskiej do zastosowań w sytuacjach kryzysowych [30]. Warstwowa architektura gridu *Comcute* umożliwia jego skalowanie oraz obsługę awarii i sytuacji wyjątkowych [154]. Innowacyjność rozwiązania polega na możliwości dołączenia do obliczeń poprzez stronę internetową projektu (rys. 3.2) bez konieczności instalacji dedykowanej aplikacji klienckiej [32].

Rysunek 3.2: Kluczowy fragment strony głównej projektu *Comcute* [131]

Zakłada się, że w sytuacjach kryzysowych wolontariusze oraz pracownicy administracji powinni udostępnić moc obliczeniową komputerów w celu wsparcia obliczeń na potrzeby działań antykryzysowych. Wyeliminowanie dedykowanego oprogramowania klienckiego pozwala uniknąć konieczności ingerencji administratora przed przyłączeniem komputera do gridu, co skraca czas potrzebny na zwiększenie mocy obliczeniowej systemu *Comcute* [146]. Prowadzone obliczenia mogą służyć symulacji działań w sytuacji kryzysowej w celu wyznaczenia strategii do zastosowania w warunkach rzeczywistych [160].

Dołączenie do obliczeń wymaga skierowania przeglądarki internetowej pod adres strony projektu *Comcute* i kliknięcia na wyróżniony przycisk. Skrypt startowy w przeglądarce internetowej przesyła informacje o obsługiwanych technologiach do serwera dystrybuującego dane i zadania. Wolontariusz otrzymuje zwrotnie zadanie, które najbardziej efektywnie wykorzysta zasoby jego urządzenia. Przykładowo, tablet otrzyma skrypt w języku *JavaScript*, a komputer wyposażony w maszynę wirtualną języka *Java* – aplet *Javy* [157]. Istnieje także możliwość prowadzenia obliczeń na smartfonach czy konsolach do gier [30].

W wyniku modernizacji systemu przeprowadzonej dla celów rozprawy dokonano implementacji środowiska agentowego w gridzie *Comcute*. Agenty klasy *W zarządzają obliczeniami* i organizują się w wirtualne grupy robocze dla realizacji zadań zleceniodawców. Z kolei agenty klasy *S dystrybuują zadania i dane* do urządzeń wolontariuszy. Każdy agent dystrybucyjny jest powiązany przynajmniej z jednym agentem zarządzającym. Zasadę działania gridu *Comcute* przedstawiono na rysunku 3.3. System cechuje się skalowalnością w każdej warstwie.



Rysunek 3.3: Zasada działania systemu *Comcute*
Źródło: opracowanie własne na podstawie [29]

Niech agenty typu W i S reprezentowane są jako elementy zbioru $\mathscr{A} = \{\alpha_1, \ldots, \alpha_v, \ldots, \alpha_V\}$. Agenty klasy W indeksuje się od 1 do V_W, a agenty S – od $V_W + 1$ do V. Zakłada się, że agent α_v może być uruchomiony na komputerach, których rodzaje należą do zbioru $\mathscr{B} = \{\beta_1, \ldots, \beta_j, \ldots, \beta_J\}$. Komputery mogą różnić się pod względem parametrów, takich jak: wydajność procesora, wielkość pamięci operacyjnej, wielkość pamięci dyskowej czy pobór mocy elektrycznej. W przypadku modernizacji systemu rozważa się wektor kosztów zakupu komputerów $\xi = [\xi_1, \ldots, \xi_j, \ldots, \xi_J]$, gdzie ξ_j jest kosztem zakupu komputera j-tego rodzaju [JM, JM – jednostka monetarna]. Można również rozpatrywać komputery już zakupione i zainstalowane w gridzie, przy czym wówczas w modelu przyjmuje się, że koszt zakupu ξ_j wynosi 0 (tabela nr 3.1).

Niech ustalona jest liczba zainstalowanych komputerów j-tego rodzaju ν_j, która nie może być większa od zadanego limitu $\widehat{\nu}_j$. Dopuszcza się sytuację, gdy zainstalowany komputer będzie zredukowany w nowej konfiguracji lub też przeniesiony do innego węzła. Jeśli ten sam rodzaj komputera jest dostępny na rynku oraz kilka jego egzemplarzy jest już zainstalowanych w gridzie, to w modelu traktuje się te komputery jako komputery różnych rodzajów ze względu na różne koszty, którymi się cechują.

Tabela 3.1: Parametry wybranych rodzajów serwerów dostępnych na rynku lub zainstalowanych w gridzie *Comcute*

j	Nazwa serwera	Procesory	ram_j	hdd_j	ε_j	ξ_j	ϑ_j	$\widehat{\nu}_j$
1	DELL R520 E5640 v1	2 x Intel 4-core Xeon E-5640 2,66GHz	24	3	900	0	11324	4
2	DELL R520 E5640 v2	2 x Intel 4-core Xeon E-5640 2,66GHz	48	3	900	0	11324	5
3	Infotronik ATX i5-4430	Intel 4-Core i5-4430 3,0GHz	8	2	400	4628	6284	$\geqslant I$
4	Infotronik ATX i7-4790	Intel 4-Core i7-4790 3,6GHz	16	2	550	4919	10110	$\geqslant I$
5	BizServer E5-2660v2	2 x Intel 6-core Xeon E5-2660v2 2,20GHz	48	24	1280	11751	23922	$\geqslant I$
6	DELL R520	2 x Intel 6-core Xeon E5-2420v2 2,20GHz	32	4	1500	12596	17194	$\geqslant I$
7	Fujitsu Primergy RX300S8	2 x Intel 6-core Xeon E5-2620v2 2,10GHz	32	1,6	900	20218	17384	$\geqslant I$
8	IBM x3650 M4	2 x Intel 6-core Xeon E5-2620v2 2,10GHz	48	4,6	675	21350	17384	$\geqslant I$
9	HP ProLiant DL380p gen8	2xIntel 10-core Xeon E5-2690v2 2.90 GHz	32	2	1500	33324	28720	$\geqslant I$
10	HP ProLiant E5-2695	2xIntel 14-core Xeon E5-2695v3 2.3 GHz	32	2	1500	37338	42246	$\geqslant I$
11	HP ProLiant E5-2697	2xIntel 14-core Xeon E5-2697v3 2.6 GHz	32	2	1500	39951	45112	$\geqslant I$
12	HP ProLiant E5-2699	2xIntel 18-core Xeon E5-2699v3 2,3 GHz	32	2	1500	47075	50384	$\geqslant I$

Oznaczenia:
ram_j – wielkość pamięci RAM (ang. *Random-Access Memory*) komputera j-tego rodzaju [GB],
hdd_j – wielkość pamięci dyskowej HDD (ang. *Hard Disk Drive*) komputera j-tego rodzaju [TB],
ε_j – pobór mocy elektrycznej przez komputer j-tego rodzaju [W],
ξ_j – koszt zakupu komputera j-tego rodzaju [zł], dla serwerów już zainstalowanych w gridzie *Comcute* koszt jest zerowy,
ϑ_j – indeks wydajności komputera j-tego rodzaju wg testu *CPU Mark* [212],
$\widehat{\nu}_j$ – limit dostępnych komputerów j-tego rodzaju.
Źródło: opracowanie własne na podstawie [209]

Wybrane komputery instalowane są w węzłach reprezentujących możliwe lokalizacje, np. sloty w szafie typu *rack* lub pomieszczenia, w których zainstalowano klaster obliczeniowy. Węzłem może być także obszar, w którym pracuje laptop, stacja robocza czy tablet. Zakłada się, że komputery, w których uruchamiane są agenty warstwy pośredniczącej, mogą być ulokowane tylko w zaufanych węzłach należących do zbioru $\Omega = \{\omega_1, \ldots, \omega_i, \ldots, \omega_I\}$. Ponadto zakłada się, że w każdym węźle może być zainstalowany dokładnie jeden komputer ze zbioru $\mathscr{B}$.

Wersja laboratoryjna gridu *Comcute* we wrześniu 2016 roku obejmowała dziewięć węzłów, w których można uruchamiać agenty klasy W lub S. Po zakupie komputerów i zainstalowaniu wymaganego oprogramowania grid można rozbudować o dodatkowe węzły. W tym wypadku istotne są ograniczenia finansowe i energetyczne. Administrator może również wyłączyć wybrane węzły

lub zredukować liczbę agentów w warstwie pośredniczącej w przypadku zmniejszonego zapotrzebowania na moc obliczeniową.

Zleceniodawca, uruchamiając zadanie obliczeniowe, specyfikuje dane wejściowe do przetworzenia oraz parametry wykonania dotyczące stopnia replikacji obliczeń, redundancji oraz wydajności. Niech zbiór zadań realizowanych przez grid *Comcute* definiuje się jako $\{z_1, \ldots, z_m, \ldots, z_M\}$. Wykonanie każdego z nich obejmuje następujące fazy: inicjację, dzielenie danych, propagację danych, wykonanie obliczeń na komputerach wolontariuszy, zbieranie wyników oraz scalanie wyników [32].

W pierwszej fazie, zadanie z_m zostaje przydzielone do inicjującego agenta zarządzającego W_m^0, który negocjuje z innymi agentami klasy W w celu zorganizowania grupy roboczej $\mathcal{W}_m$. Niech parametr wydajnościowy $P_W(z_m)$ oznacza liczebność grupy $\mathcal{W}_m$ na poziomie $P_W(z_m) = n$. Agent W_m^0 wysyła do pozostałych agentów W ofertę realizacji zadania i oczekuje na zgłoszenia gotowości do współpracy. Wykonanie zadania rozpocznie się, jeśli agent W_m^0 otrzyma przynajmniej $n-1$ zgłoszeń. W przeciwnym wypadku, zleceniodawca otrzymuje komunikat o braku wystarczających zasobów do wykonania zadania. Zleceniodawca może podjąć próbę uruchomienia zadania z innym parametrem $P_W(z_m)$.

W drugiej fazie komponent *partycjoner* przekształca dane wejściowe $DATA_m$ do zadania z_m na $L(z_m)$ paczek danych: $IN_m^{(1)}, \ldots, IN_m^{(l)}, \ldots, IN_m^{(L(z_m))}$. Niech parametr $P_R(z_m) = r$ specyfikuje stopień replikacji danych zadania z_m. Paczki danych $IN_m^{(l)}$, $l = \overline{1,L(z_m)}$ replikowane są r-krotnie przez agentów z grupy $\mathcal{W}_m$.

W fazie propagacji agenty klasy W wysyłają do agentów klasy S moduły obliczeniowe zadania wraz z informacją o gotowości paczek danych. Agenty klasy S posiadają bufor danych wejściowych wypełniany paczkami $IN_m^{(l)}$. Gdy wolontariusze zgłaszają agentom typu S gotowość do obliczeń, paczki danych z bufora są wysyłane wraz z modułem obliczeniowym, który realizuje ich przetwarzanie. Wyniki obliczeń zapisywane są w buforze zwrotnym agenta typu S, który odsyła je do agenta zarządzającego klasy W po zapełnieniu bufora.

Agenty z grupy $\mathcal{W}_m$ weryfikują zgodność wyników otrzymywanych dla tych samych paczek danych. Gdy wszystkie rezultaty zostaną skompletowane i zweryfikowane, następuje faza scalania wyników. Komponent *linker* analizuje rezultaty dla indywidualnych paczek danych $IN_m^{(l)}$ i oblicza na ich podstawie końcowy wynik dla zadania obliczeniowego, który może zostać pobrany przez zleceniodawcę.

Wybór agentów do grupy $\mathcal{W}_m$ uzależniony jest od powiązań pomiędzy agentami zarządzającymi. Administrator gridu *Comcute* określa, które agenty mogą ze sobą współpracować. Konfigurację powiązań między agentami zarządającymi typu W można przedstawić w postaci macierzy binarnej, jak niżej:

$$w = [w_{vu}]_{V_W \times V_W}, \tag{3.1}$$

gdzie $w_{vu} = \begin{cases} 1, & \text{gdy } \alpha_v \text{ i } \alpha_u \text{ mogą pracować w tej samej grupie roboczej lub } v = u, \\ 0, & \text{w przeciwnym razie.} \end{cases}$

Zbiór $\mathcal{W}^{(\alpha_v)}$ obejmujący agenty, które mogą wejść w skład grupy roboczej $\mathcal{W}_m$ formowanej przez agenta α_v, gdy pełni on rolę inicjatora W_m^0 dla zadania z_m, zdefiniowano następująco:

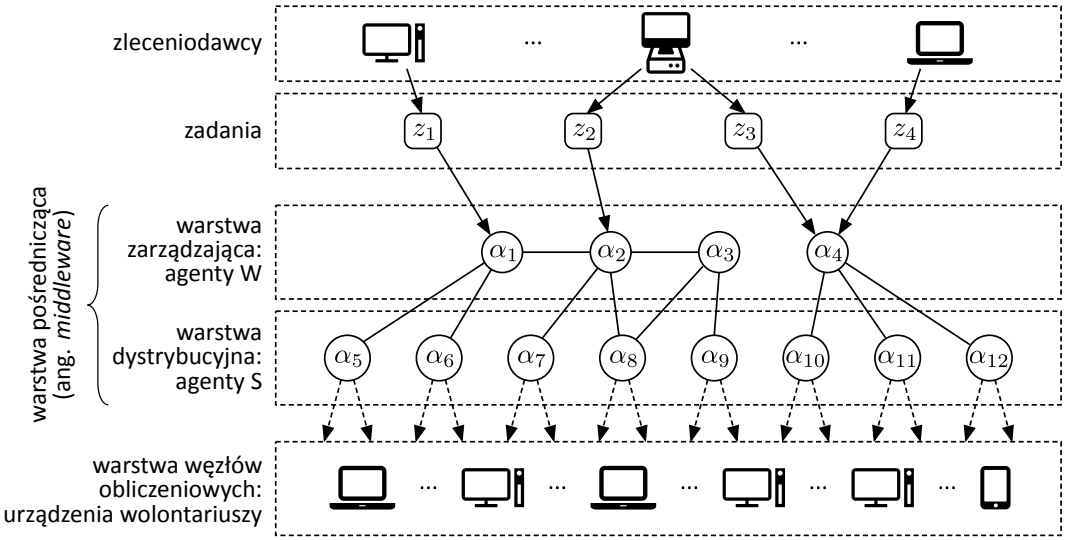
$$\mathcal{W}^{(\alpha_v)} = \{\alpha_u \in \{\alpha_1, \ldots, \alpha_{V_W}\} : w_{vu} = 1\},\ v = \overline{1, V_W}. \tag{3.2}$$

Każdy z agentów W propaguje zadania i dane do przypisanych agentów typu S. Relacje pomiędzy agentami W i S opisuje macierz:

$$s = [s_{vu}]_{V_W \times (V - V_W)}, \tag{3.3}$$

gdzie $s_{vu} = \begin{cases} 1, & \text{gdy agent } \alpha_v \text{ typu } W \text{ współpracuje z agentem } \alpha_u \text{ typu } S, \\ 0, & \text{w przeciwnym razie.} \end{cases}$

Przykłady powiązań między agentami typu W i S pokazano na rysunku 3.4. Agenty α_1, α_2 i α_3 typu W mogą wspólnie realizować zadania. Z kolei agent α_4 może wyłącznie wykonywać zadania o parametrze wydajnościowym $P_W(z_m) = 1$, gdyż nie jest powiązany z innymi agentami typu W. Ponadto agent α_4 współpracuje z trzema agentami typu S: α_{10}, α_{11}, α_{12}, które nie są wykorzystywane przez inne agenty zarządzające. Każdy spośród pozostałych agentów typu W współpracuje z dwoma agentami typu S, przy czym agent α_8 otrzymuje zadania od dwóch agentów zarządzających.



Rysunek 3.4: Przykłady powiązań między agentami warstwy pośredniczącej w gridzie *Comcute*
Źródło: opracowanie własne

Zbiór $\mathcal{S}^{(\alpha_v)}$ agentów dystrybucyjnych wykorzystywanych przez agenta α_v typu W definiuje się następująco:

$$\mathcal{S}^{(\alpha_v)} = \{\alpha_u \in \{\alpha_{V_W+1}, \ldots, \alpha_V\} : s_{vu} = 1\},\ v = \overline{1, V_W}. \tag{3.4}$$

Natomiast zbiór $\mathcal{S}_m$ agentów typu S biorących udział w dystrybucji zadania z_m można zdefiniować, jak niżej:

$$\mathcal{S}_m = \bigcup_{\alpha_v \in \mathcal{W}_m} \mathcal{S}^{(\alpha_v)},\ m = \overline{1,M}. \tag{3.5}$$

Liczba paczek danych zadania z_m przypadających na jednego agenta z grupy roboczej $\mathcal{W}_m$ wyznaczana jest według poniższej zależności:

$$L_W(z_m) = \frac{L(z_m)P_R(z_m)}{|\mathcal{W}_m|} = \frac{L(z_m)P_R(z_m)}{P_W(z_m)}. \tag{3.6}$$

Agenty klasy W wysyłają paczki danych do skojarzonych z nimi agentów dystrybucyjnych zgodnie ze zgłaszanym przez nie zapotrzebowaniem. Wynika ono z liczby żądań od węzłów obliczeniowych, jakie otrzymuje każdy z agentów typu S. Zastosowane mechanizmy równoważenia obciążenia pozwalają równomiernie rozłożyć strumień żądań wolontariuszy pomiędzy agenty dystrybucyjne. Przyjmuje się, że średnia liczba żądań jest taka sama dla wszystkich agentów klasy S. Przy założeniu, że każdy agent typu S przypisany jest do jednego agenta typu W, liczba paczek zadania z_m obsługiwanych przez agenta α_u typu S wyznaczana jest następująco:

$$L_S^{\alpha_u}(z_m) = \sum_{\alpha_v \in \mathcal{W}_m} \frac{L_W(z_m)}{\left|\mathcal{S}^{(\alpha_v)}\right|} s_{vu}. \tag{3.7}$$

Jeśli agent klasy S zostanie skojarzony z wieloma agentami typu W, konieczne jest uwzględnienie faktu, że agent dystrybucyjny przesyła wolontariuszom zadania od różnych agentów W zgodnie ze schematem *round robin* [147]. Dodatkowo niektóre agenty typu W mogą zakończyć przetwarzanie danych wcześniej niż inne. Jeśli jeden z agentów klasy W współpracujących z wybranym agentem typu S zakończy przetwarzanie, dalsze żądania węzłów obliczeniowych do rozpatrywanego agenta typu S odnoszą się do paczek z pozostałych, dysponujących jeszcze danymi, agentów typu W. Liczba przetwarzanych paczek zadania z_m może zostać wyznaczona za pomocą symulatora, który przedstawiono na diagramie algorytmu nr 3.1.

Każdy obieg pętli *while* w liniach 4-16 odpowiada przetworzeniu wszystkich danych przez agenta w^* typu W, którego paczki przesłano do największej liczby węzłów obliczeniowych, co wynikało z powiązań z agentami typu S. Jeśli dane dla innych agentów klasy W są przetwarzane z taką samą intensywnością, jak dane przetwarzane przez agenta w^*, to wszystkie te agenty otrzymują komplet odpowiedzi do danych w tym samym obiegu pętli. Agenty typu W, których dane przetworzono, trafiają do zbioru $\mathcal{W}^*$. W kolejnym przebiegu pętli agenty dysponujące jeszcze danymi – *agenty aktywne* – należą do zbioru $\mathcal{W}_m \setminus \mathcal{W}^*$ i tylko one są rozpatrywane.

Agenty typu S związane zarówno z agentami ze zbioru $\mathcal{W}^*$, jak i $\mathcal{W}_m \setminus \mathcal{W}^*$, przesyłają wolontariuszom tylko zadania i dane pochodzące od agentów aktywnych, co zmienia proporcje obsługi danych z różnych agentów klasy W w stosunku do poprzedniego obiegu pętli. Pętla symulująca działanie systemu kończy się, gdy dane wszystkich agentów typu W zostaną przetworzone. Zwracana tablica L_S zawiera liczbę paczek obsłużonych przez każdego z agentów klasy S i może być użyta do określenia wartości funkcji $L_S^{\alpha_v}(z_m)$.

Algorytm 3.1 Diagram symulatora przetwarzania paczek danych przez agenty z grupy $\mathcal{S}_m$

$\mathcal{W}^* := \varnothing$;
$L_S := [\ 0$ **for** $\alpha_u \in \mathcal{S}_m\ ]$;
$L_W := [\ L_W(z_m)$ **for** $\alpha_v \in \mathcal{W}_m\ ]$;
while $\mathcal{W}_m \neq \mathcal{W}^*$ **do**
  $w^* := \underset{v \in \mathcal{W}_m \setminus \mathcal{W}^*}{\arg\max} \sum_{\alpha_u \in \mathcal{S}^{(\alpha_v)}} \left| \mathcal{W}_m \cap \left(W^{(\alpha_u^S)} \setminus \mathcal{W}^* \right) \right|^{-1}$;
  $w^{\max} := \max_{v \in \mathcal{W}_m \setminus \mathcal{W}^*} \sum_{\alpha_u \in \mathcal{S}^{(\alpha_v)}} \left| \mathcal{W}_m \cap \left(W^{(\alpha_u^S)} \setminus \mathcal{W}^* \right) \right|^{-1}$;
  $w^{\text{suma}} := \sum_{v \in \mathcal{W}_m \setminus \mathcal{W}^*} \sum_{\alpha_u \in \mathcal{S}^{(\alpha_v)}} \left| \mathcal{W}_m \cap \left(W^{(\alpha_u^S)} \setminus \mathcal{W}^* \right) \right|^{-1}$;
  $req := \frac{w^{\text{suma}}}{w^{\max}} L_W[w^*]$;
  $S^{\text{aktywne}} := \bigcup_{\alpha_v \in \mathcal{W}_m \setminus \mathcal{W}^*} \mathcal{S}^{(\alpha_v)}$;
  $req_{1S} := \frac{req}{|S^{\text{aktywne}}|}$;
  for $\alpha_v \in \mathcal{W}_m \setminus \mathcal{W}^*$ **do**;
    $L_W[\alpha_v] := L_W[\alpha_v] - \frac{req}{w^{\text{suma}}} \sum_{\alpha_u \in \mathcal{S}^{(\alpha_v)}} \left| \mathcal{W}_m \cap \left(W^{(\alpha_u^S)} \setminus \mathcal{W}^* \right) \right|^{-1}$;
  end for
  for $\alpha_u \in S^{\text{aktywne}}$ **do**
    $L_S[\alpha_u] := L_S[\alpha_u] + req_{1S}$;
  end for
  $\mathcal{W}^* := \mathcal{W}^* \cup \{\alpha_v : L_W[\alpha_v] = 0\}$;
end while
return L_S;

Źródło: opracowanie własne

3.3 Eksperymentalne wyznaczenie wartości podstawowych parametrów agentów w gridzie *Comcute*

W celu wyznaczenia wartości podstawowych parametrów agentów w systemie *Comcute* wykonano serię pomiarów z wykorzystaniem różnych konfiguracji gridu i zadań użytkowników o różnych wielkościach danych. Istotne jest określenie wpływu poszczególnych elementów konfiguracji systemu na obciążenie agentów typu W i S. Pozwala to na opracowanie metod szacowania czasów przetwarzania i czasów komunikacji między agentami dla różnorodnych ustawień gridu. Zbadano wpływ następujących determinantów na obciążenie agentów:

- liczba agentów klasy W w systemie,
- liczba agentów klasy S w systemie,

- liczba agentów zarządzających klasy W w grupie roboczej $\mathcal{W}_m$,
- liczba agentów dystrybucyjnych klasy S przypisanych do jednego agenta klasy W,
- liczba paczek danych wejściowych zadania obliczeniowego,
- liczba paczek danych przypadająca na jednego agenta,
- liczba węzłów obliczeniowych (wolontariuszy) uczestniczących w obliczeniach,
- liczba żądań od węzłów obliczeniowych w jednostce czasu.

Na potrzeby testów wykorzystano węzły klastra *K2* Katedry Architektury Systemów Komputerowych Wydziału ETI Politechniki Gdańskiej oraz komputery typu *desktop* zainstalowane w katedralnym laboratorium. Każdy z węzłów klastra wyposażono w dwa 4-rdzeniowe procesory Intel Xeon E5345 taktowane zegarem o częstotliwości 2,33 GHz oraz w 8 GB pamięci operacyjnej. Na klaster składa się 106 węzłów, co pozwala na generowanie znacznej liczby żądań skierowanych do serwerów dystrybucyjnych grida *Comcute*. Pomiędzy klastrem a serwerami systemu *Comcute* dostępne jest łącze typu *Gigabit Ethernet*.

Niech laboratoryjne komputery klasy *desktop* reprezentują model konfiguracji sprzętowych, jakie udostępniają wolontariusze. Są one wyposażone w 4-rdzeniowe procesory Intel Core i7-2600K taktowane zegarem 3,40 GHz z obsługą technologii *Hyper-Threading* (8 rdzeni logicznych). Dostępna jest pamięć operacyjna o rozmiarze 8 GB i karty sieciowe typu *Gigabit Ethernet*.

Agenty typu W i S uruchomiono na komputerach klasy serwerowej. Agenty klasy S były przydzielane do węzłów wyposażonych w dwa 4-rdzeniowe procesory Intel Xeon E5640 2,67 GHz z obsługą technologii *Hyper-Threading* (łącznie 16 rdzeni logicznych) i 24 GB pamięci operacyjnej. Agenty klasy W wykorzystywały węzły o takiej samej konfiguracji z wyjątkiem pamięci operacyjnej, którą rozszerzono do 48 GB.

Przedstawione wyniki stanowią uśrednienie pomiarów z wielokrotnych powtórzeń scenariuszy testowych. Pierwszy pomiar po rekonfiguracji systemu odrzucano ze względu na model wykonania aplikacji przez maszynę wirtualną *Javy* oraz konieczność zainicjowania kanałów komunikacyjnych pomiędzy agentami. Aplikacje w języku *Java* są tłumaczone z kodu pośredniego na kod natywny. Uzyskane zestawy instrukcji procesora są utrzymywane w pamięci, co przyspiesza ponowne wywołanie uruchomionych już wcześniej metod. Podczas pierwszego wykonania testu, system cechuje się zatem gorszą wydajnością niż w kolejnych instancjach.

Prezentowane charakterystyki odzwierciedlają kilkaset godzin pracy systemu *Comcute*, który przetworzył w tym czasie ok. 20 milionów paczek danych do zadań obliczeniowych. Warty podkreślenia jest fakt, że były to pierwsze testy gridu *Comcute* w tak dużej skali i z tak dużą różnorodnością konfiguracji. Wykorzystane zestawy danych przewyższały o rząd wielkości te, które stosowano we wcześniejszych pomiarach [32, 209].

Pierwsza seria pomiarów miała na celu wyznaczenie czasu przetwarzania, a także czasu komunikacji agenta klasy S w zależności od liczby paczek danych dystrybuowanych do węzłów obliczeniowych. Testowane były scenariusze, w których agent typu S przetwarzał: 16 tys., 32 tys. oraz 64 tys. paczek przy ustalonej liczbie żądań na sekundę od wolontariuszy. Następnie dla każdego ze scenariuszy dodawano kolejne agenty klasy S przy zachowaniu stałego obciążenia i stałej liczby paczek danych przypadających na jednego agenta klasy S. W ostatniej serii testów dla 16 agentów

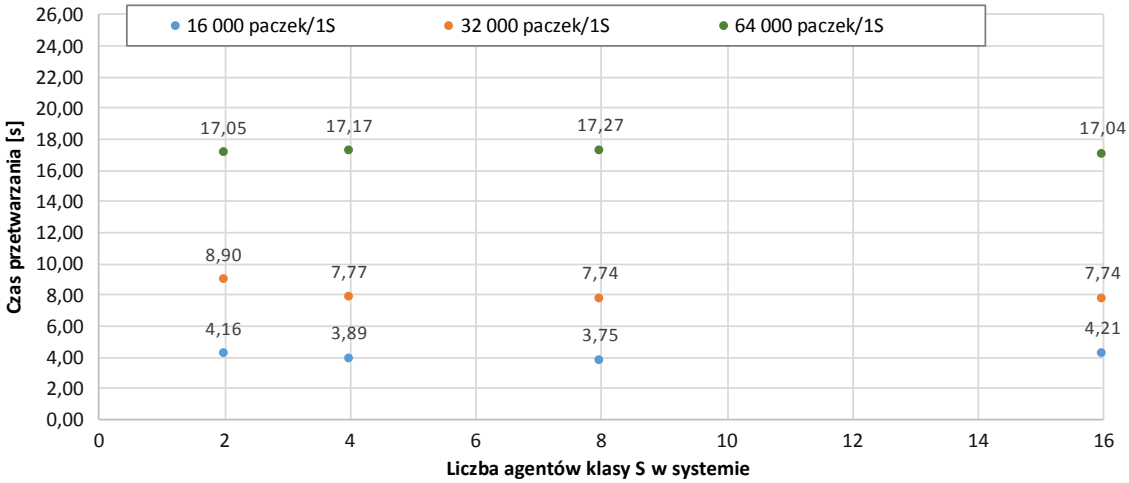
klasy S i 64 tys. paczek na każdego agenta, grid *Comcute* przetwarzał ponad milion paczek danych dla każdego z wykonanych zadań obliczeniowych. Zmierzone czasy przedstawiono w tabeli nr 3.2.

Tabela 3.2: Obciążenie agenta klasy S w zależności od liczby paczek danych do przetworzenia oraz od łącznej liczby agentów klasy S w gridzie

Liczba agentów S	Liczba paczek danych na agenta klasy S	Średni czas przetwarzania przez agenta klasy S [s]	Średni czas komunikacji agenta klasy S z agentem klasy W [s]
2	16 000	4,16	1,67
	32 000	8,90	3,46
	64 000	17,05	6,88
4	16 000	3,89	1,52
	32 000	7,77	3,20
	64 000	17,17	6,54
8	16 000	3,75	1,57
	32 000	7,74	3,46
	64 000	17,27	6,70
16	16 000	4,21	1,82
	32 000	7,74	3,52
	64 000	17,04	7,48

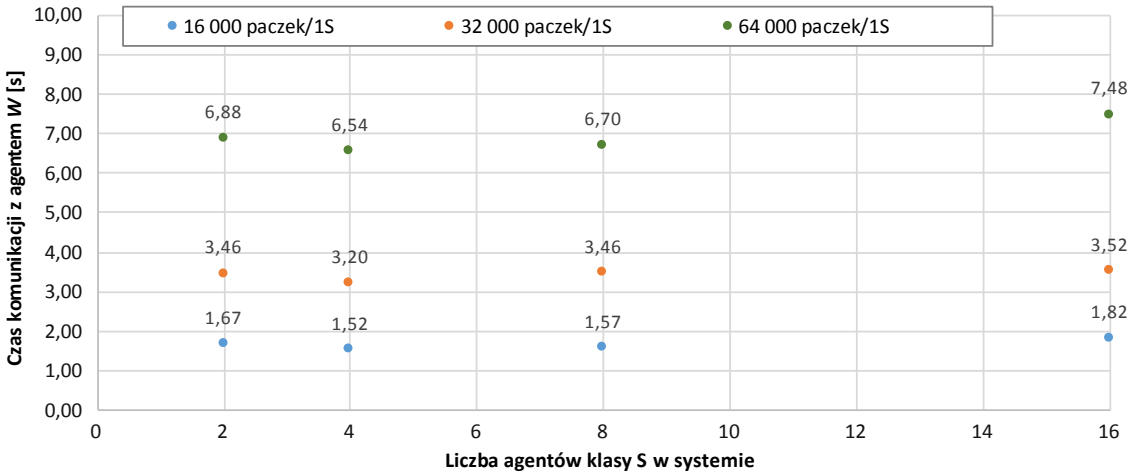
Źródło: opracowanie własne

Na rysunku 3.5 zaprezentowano czasy przetwarzania, a na rysunku 3.6 – czasy komunikacji. Każdy z oznaczonych punktów pomiarowych stanowi średnią z 5 testów powtórzonych dla każdej spośród badanych konfiguracji systemu i rozpatrywanych wielkości danych.



Rysunek 3.5: Czas przetwarzania wybranej liczby paczek danych przez agenta klasy S w zależności od liczby paczek oraz od łącznej liczby agentów klasy S przetwarzających taką samą liczbę paczek w ramach zadania obliczeniowego
Źródło: opracowanie własne

Zauważa się, że przy stałym obciążeniu czasy zależą od liczby paczek do przetworzenia. Obecność innych agentów klasy S nie wpływa na lokalne działanie pojedynczego agenta tego typu.



Rysunek 3.6: Obciążenie komunikacyjne agenta klasy S w zależności od liczby obsługiwanych paczek danych i liczby agentów klasy S przetwarzających taką samą liczbę paczek w ramach zadania obliczeniowego
Źródło: opracowanie własne

Równocześnie, czasy przetwarzania rosną wprost proporcjonalnie do liczby paczek danych. Zastosowano metodę regresji liniowej w celu wyznaczenia funkcji pozwalającej na oszacowanie czasu przetwarzania paczek danych przez agenta typu S. Uzyskano funkcję w następującej postaci:

$$t_v^{(S)}(z_m) = 3 \cdot 10^{-4} L_S^{\alpha_v}(z_m) - 0{,}545, \tag{3.8}$$

gdzie $t_v^{(S)}(z_m)$ – oszacowanie czasu przetwarzania danych zadania z_m przez agenta klasy S o numerze v [s].

Wyznaczony za pomocą analizy regresji współczynnik determinacji [244] wynosi $R^2 = 0{,}996$, co oznacza precyzyjne odwzorowanie rzeczywistych pomiarów. W analogiczny sposób wyznaczono funkcję oszacowania dla czasów komunikacji:

$$\tau_{u,v}^{(SW)}(z_m) = 10^{-4} L_S^{\alpha_v}(z_m) - 0{,}1, \tag{3.9}$$

gdzie $\tau_{u,v}^{(SW)}(z_m)$ – oszacowanie czasu komunikacji agenta klasy S o numerze u z agentem klasy W o numerze v w czasie obsługi zadania z_m [s].

Uzyskane oszacowanie cechuje się współczynnikiem determinacji $R^2 = 0{,}989$. Ponieważ komunikacja angażuje oba agenty, które w niej uczestniczą, przyjmuje się, że $\tau_{v,u}^{(WS)} = \tau_{u,v}^{(SW)}$.

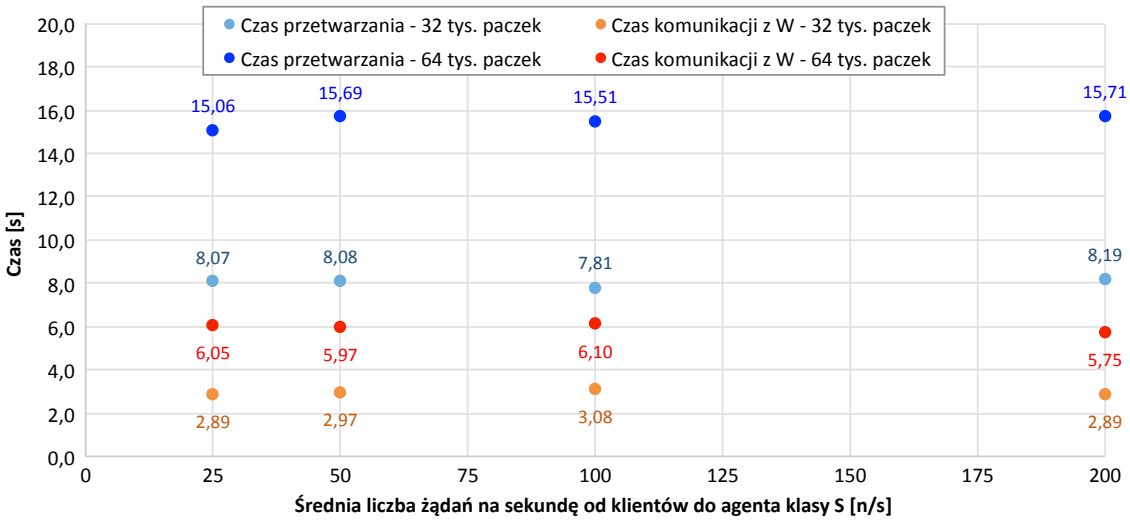
Kolejnym badanym determinantem była liczba żądań na sekundę kierowanych do agentów dystrybucyjnych typu S przez węzły obliczeniowe. Przyjęto stałą liczbę agentów w warstwie pośredniczącej gridu (2 agenty typu W i 2 agenty typu S, po jednym agencie dystrybucyjnym przypisanym do każdego agenta zarządzającego) oraz zadanie przetwarzające dane o zadanej wielkości (32 tys. paczek danych dla każdego agenta S), które to zadanie było realizowane z intensywnością od 25 do 200 żądań na sekundę przypadających na agenta klasy S. Następnie testy powtórzono dla danych

o dwukrotnie większej wielkości. Czasy zmierzone dla agentów klasy S zaprezentowano w tabeli nr 3.3. Wyniki przedstawiono również na rysunku 3.7.

Tabela 3.3: Obciążenie agenta klasy S w zależności od liczby żądań na sekundę od węzłów obliczeniowych w czasie przetwarzania 32 tys. paczek danych

Liczba wolontariuszy/1S	Liczba żądań/ 1S/sekundę	Agent klasy S		Czas realizacji zadania [s]
		Czas przetwarzania [s]	Czas komunikacji [s]	
125	25	8,07	2,89	1349
250	50	8,08	2,97	706
500	100	7,81	3,08	385
1000	200	8,19	2,89	222

Źródło: opracowanie własne



Rysunek 3.7: Obciążenie agenta klasy S w zależności od liczby żądań na sekundę od węzłów obliczeniowych
Źródło: opracowanie własne

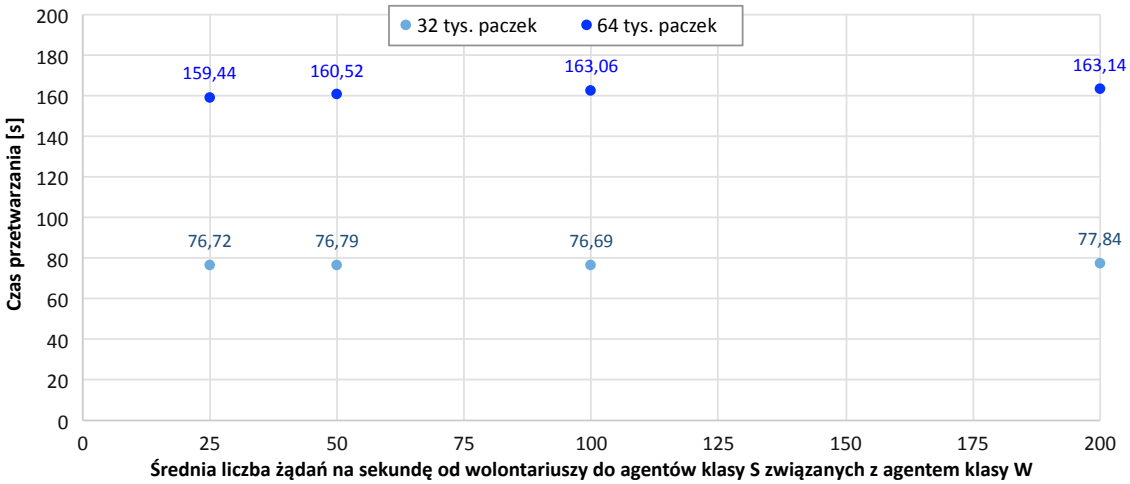
Dla testowanego zadania przetworzenie paczki danych wymagało średnio 5 s pracy procesora w wykorzystywanych węzłach obliczeniowych klastra *K2*. Wygenerowanie strumienia żądań o większej intensywności wymagało zaangażowania kolejnych węzłów symulujących wolontariuszy, którzy podłączają komputery do systemu. Większa liczba węzłów obliczeniowych przekłada się bezpośrednio na krótszy czas realizacji zadania z punktu widzenia zleceniodawcy obliczeń, co pokazano w ostatniej kolumnie tabeli nr 3.3.

Równocześnie, czas przetwarzania danych przez agenty klasy S oscylował wokół wartości stałej przy zadanej wielkości danych. Wynika to z faktu, że agent klasy S w każdym ze scenariuszy miał do wykonania te same operacje na takiej samej liczbie paczek danych. Przy większej liczbie klientów, okresy aktywności agenta klasy S występowały częściej – wraz ze wzrostem liczby żądań na sekundę. W czasie, gdy komputery wolontariuszy przetwarzają otrzymane dane, agenty typu S oczekują na zgłoszenia, nie obciążając procesora. Skoro dla agenta klasy S czas przetwarzania

paczek nie zależy od tempa, w jakim są one pobierane przez węzły obliczeniowe, a jedynie od liczby paczek, to zależność (3.8) pozostaje w mocy również dla zmiennych obciążeń ze strony wolontariuszy.

Czasy komunikacji agenta typu S z agentem typu W również utrzymują się na stałym poziomie. Czas komunikacji wynika z rozmiaru danych do przesłania i przepustowości łącza. Dla tego samego zadania agenty klasy W i S wymieniają się danymi o stałym rozmiarze, które obejmują paczki wejściowe dla wolontariuszy i odebrane od nich wyniki. W tym przypadku również nie jest istotne tempo przetwarzania paczek danych, a jedynie ich liczba.

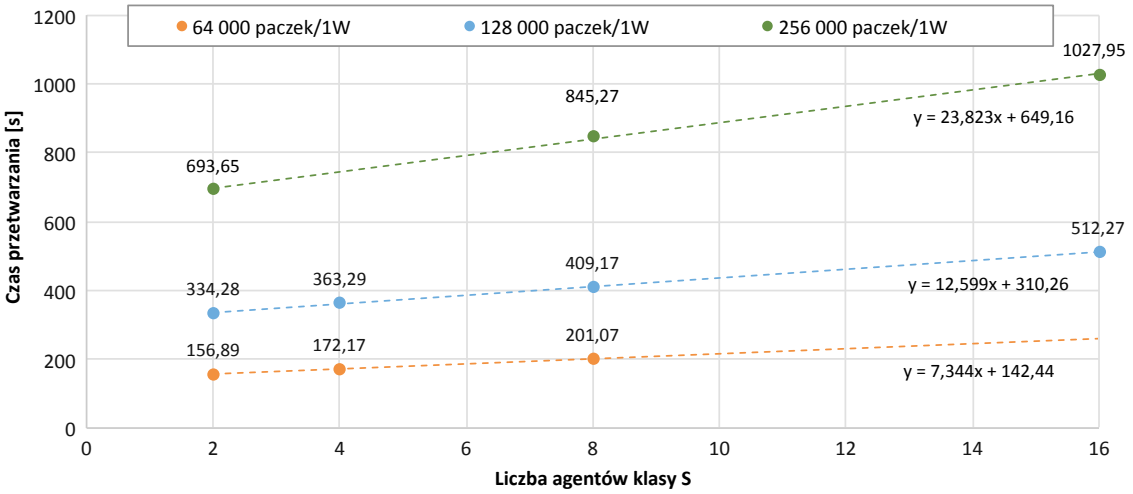
Agenty klasy W przetestowano pod kątem czasów przetwarzania przy zmiennej liczbie żądań napływających do systemu. Agenty te nie komunikują się bezpośrednio z węzłami obliczeniowymi. Badaniom podlegała zatem liczba żądań, jaką otrzymują agenty dystrybucyjne przypisane do monitorowanych agentów typu W. Pozwoliło to na określenie, czy obciążenie agentów klasy S wpływa pośrednio na czasy przetwarzania agentów typu W. Wyniki testów zaprezentowano na rysunku 3.8.



Rysunek 3.8: Czas przetwarzania agenta zarządzającego klasy W w zależności od liczby żądań napływających do powiązanych agentów dystrybucyjnych typu S
Źródło: opracowanie własne

W pomiarach wykorzystano dwa agenty klasy W oraz dwa agenty klasy S, przy czym każdy z agentów typu W współpracował z agentem klasy S. Dla strumieni od 25 do 200 żądań na sekundę do każdego agenta klasy S testowane były zadania przetwarzające dane o dwóch różnych wielkościach: 32 tys. i 64 tys. paczek na agenta klasy W. Skumulowane czasy przetwarzania pozostają na stałym poziomie, a więc są niezależne od liczby żądań napływających do agentów dystrybucyjnych.

Przeprowadzono również testy uwzględniające różne liczby paczek danych do przygotowania i obsłużenia przez agenta klasy W. Dodatkowo zbadano, w jaki sposób liczba powiązanych agentów typu S wpływa na czasy przetwarzania agenta klasy W, przy zadaniach wymagających przetworzenia danych o takich samych rozmiarach. Wyniki przedstawiono na rysunku 3.9.



Rysunek 3.9: Czasy przetwarzania wybranej liczby paczek przez agenta klasy W w zależności od liczby powiązanych agentów typu S oraz od liczby paczek
Źródło: opracowanie własne

Zwiększenie liczby paczek danych przekłada się liniowo na czas ich przetwarzania przez agenta klasy W. Natomiast przy stałej wielkości danych, liczba skojarzonych agentów klasy S istotnie wpływa na czasy pracy agenta typu W. Na rysunku 3.9 zobrazowano liniową zależność dla testowanych wielkości danych. Dodatkowe agenty klasy S pozwalają na obsłużenie większej liczby żądań od klientów, ale równocześnie stanowią dodatkowe obciążenie dla agenta klasy W. Wynika to z faktu, że agent typu W musi koordynować pracę większej liczby agentów dystrybucyjnych.

Wyznaczenie czasu przetwarzania danych przez agenta klasy W wymaga uwzględnienia obu powyższych własności. Postać funkcji zwracającej oszacowanie jest następująca:

$$t_v^{(W)}(z_m) = \frac{L_W(z_m)}{L_W^{\text{ref}}} \left(a|\mathcal{S}^{(\alpha_v)}| + b \right), \tag{3.10}$$

gdzie:
$t_v^{(W)}(z_m)$ – oszacowanie czasu przetwarzania danych dla zadania z_m przez agenta α_v typu W [s],
L_W^{ref} – liczba paczek danych przypadających na agenta typu W dla zadania referencyjnego, w eksperymencie $L_W^{\text{ref}} = 128000$,
$|\mathcal{S}^{(\alpha_v)}|$ – liczba agentów dystrybucyjnych przypisanych do agenta α_v typu W,
a – czas obsługi agenta typu S przez agenta klasy W podczas wykonania zadania referencyjnego [s],
b – czas przetwarzania danych przez agenta typu W, który jest niezależny od liczby skojarzonych agentów klasy S (m.in. czas partycjonowania danych i scalania rezultatów obliczeń) [s].

Zastosowanie metody najmniejszych kwadratów dla funkcji szacującej (3.10) i zbioru punktów pomiarowych z przeprowadzonych testów, umożliwia wyznaczenie wartości współczynników prostej $a = 6{,}528$ oraz $b = 152{,}49$, przy współczynniku determinacji $R^2 = 0{,}977$. Jest to rezultat rekomendujący wykorzystanie tego oszacowania w modelu optymalizacji strategii zespołu inteligentnych agentów w gridzie *Comcute*.

W testach monitorowano również czasy komunikacji pomiędzy agentami klasy W. Z uzyskanych pomiarów wynika, że zależą one jedynie od czasu realizacji zadania. Ma to uzasadnienie w kilku aspektach działania sytemu *Comcute*. Po pierwsze, zadania użytkownika są jednorazowo propagowane pomiędzy agentami klasy W przed rozpoczęciem obliczeń. Po tej operacji uruchamianych jest wiele zadań z różnymi danymi wejściowymi. W związku z tym czas propagacji modułu obliczeniowego pomiędzy agentami W nie jest doliczany do czasów komunikacji związanych z wykonaniem zadania. Natomiast uwzględniane są czasy propagacji danych, które umożliwiają wygenerowanie paczek wejściowych różniących się dla każdego zadania.

Po drugie, agenty klasy W nie wymieniają się paczkami $IN_m^{(1)}, \ldots, IN_m^{(l)}, \ldots, IN_m^{(L(m))}$, lecz metadanymi zadania, które obejmują parametry dla partycjonera generującego właściwe dane wejściowe. Rozmiar metadanych dla istniejących w systemie klas zadań jest stały. Ponadto w fazie konsolidacji agenty typu W nie przesyłają między sobą odpowiedzi $OUT_m^{(1)}, \ldots, OUT_m^{(l)}, \ldots, OUT_m^{(L(m))}$. Zamiast tego każdy z nich scala zbiór własnych wyników i rezultat tej operacji przesyła do pozostałych agentów klasy W, co redukuje rozmiar wymienianych danych do wartości stałej, niezależnej od liczby paczek danych zadania. Przykładowo, dla zadania symulacji rozprzestrzeniania się pożaru każdy z agentów klasy W wysyła do pozostałych ranking pięciu najlepszych lokalizacji jednostek straży pożarnej niezależnie od tego, ile konfiguracji przetestowali wolontariusze podłączeni do agentów klasy S współpracujących z wybranym agentem klasy W. Po otrzymaniu częściowych wyników od pozostałych agentów, każdy agent typu W scala je, aby uzyskać końcowy rezultat zadania.

Dominujący udział w czasach komunikacji agentów klasy W posiadają powiadomienia przesyłane w trakcie realizacji zadania pomiędzy członkami grupy $\mathcal{W}_m$. Zawierają one informacje na temat stanu agentów i postępów przetwarzania zadania. Pozwala to agentom typu W na reagowanie na awarie, jakie mogą wystąpić w systemie. Komunikacja w grupie $\mathcal{W}_m$ odbywa się według reguły *każdy z każdym*, więc wnosi takie samo obciążenie komunikacyjne dla agentów klasy W przetwarzających zadanie. Rozmiar przekazywanych metadanych jest stały, a powiadomienia przesyłane są cyklicznie w stałych odstępach czasu.

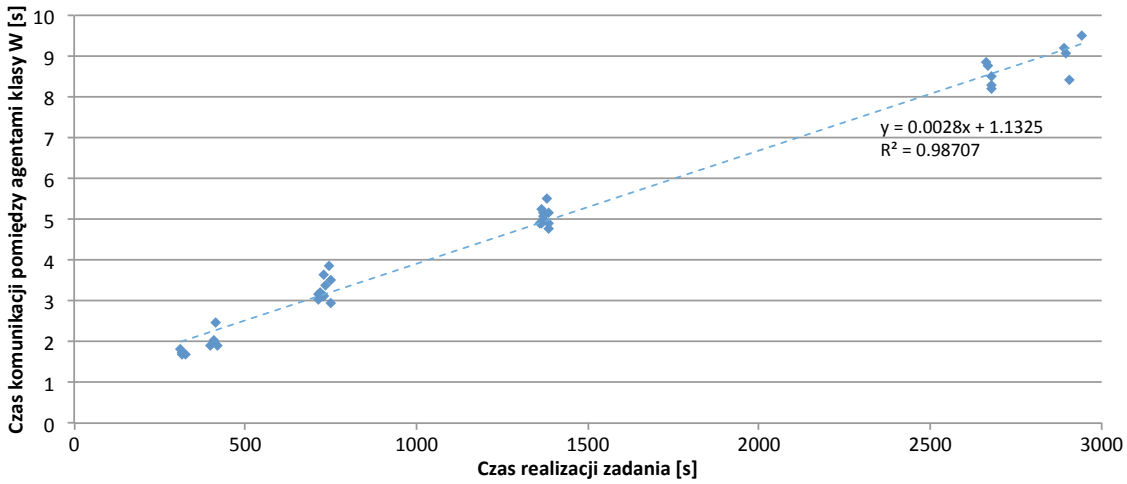
Na etapie formowania grupy roboczej $\mathcal{W}_m$, agent inicjujący $W_m^0 = \alpha_v$ komunikuje się z każdym agentem ze zbioru $\mathcal{W}^{(\alpha_v)}$. Oznacza to, że agent W_m^0 doświadcza większego obciążenia komunikacyjnego niż pozostali członkowie grupy $\mathcal{W}_m$. Formowanie grupy roboczej odbywa się jednak jednorazowo dla zadania, a protokół negocjacji udziału w niej składa się ze stałej liczby kroków. Negocjacja z agentem typu W wprowadza obciążenie komunikacyjnej nieprzekraczające 150 ms. Jest ono istotne jedynie dla zadań wymagających licznych grup $\mathcal{W}_m$. Czasy komunikacji realizowanej w trakcie wykonania zadania przedstawiono na rysunku 3.10. Wykres obejmuje pomiary dla 38 scenariuszy cechujących się różnymi czasami realizacji zadania.

Oszacowanie łącznego czasu komunikacji pomiędzy agentami klasy W wymaga uwzględnienia czasów przypadających na formowanie grupy roboczej $\mathcal{W}_m$ i czasów wymiany komunikatów

w trakcie realizacji zadania. Posługując się współczynnikami linii trendu oznaczonej na rysunku 3.10, otrzymujemy funkcję w następującej postaci:

$$\tau_{v,u}^{(WW)}(z_m) = \begin{cases} 0{,}15 \cdot |\mathcal{W}_m| + 2{,}8 \cdot 10^{-3}\ \widehat{T} + 1{,}13, & \text{gdy } \alpha_v = W_m^0, \\ 0{,}15 + 2{,}8 \cdot 10^{-3}\ \widehat{T} + 1{,}13, & \text{gdy } \alpha_v \in \mathcal{W}_m \ \wedge\ \alpha_v \neq W_m^0, \\ 0, & \text{gdy } \alpha_v \notin \mathcal{W}_m, \end{cases} \tag{3.11}$$

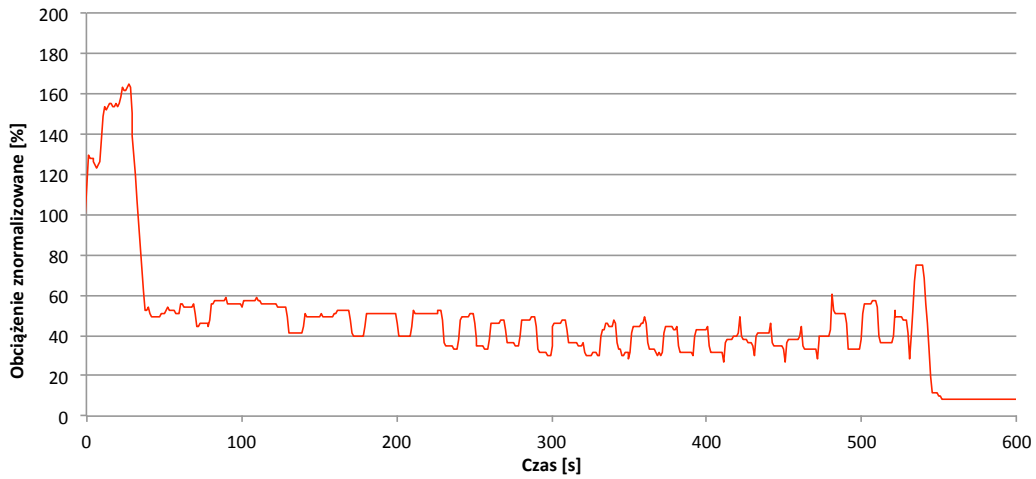
gdzie:
$\tau_{v,u}^{(WW)}(z_m)$ – czas komunikacji pomiędzy agentami α_v oraz α_u typu W w trakcie obsługi zadania z_m [s],
$\widehat{T}$ – czas realizacji zadania [s].



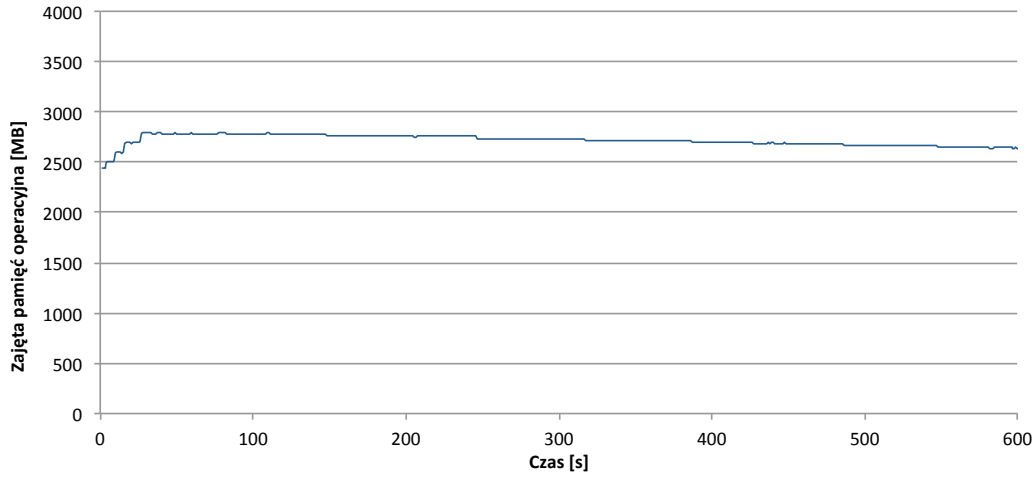
Rysunek 3.10: Czasy komunikacji pomiędzy agentami klasy W w zależności od czasu realizacji zadania
Źródło: opracowanie własne

Dokonano pomiarów obciążenia procesorów, pamięci operacyjnej i kart sieciowych na poszczególnych węzłach warstwy pośredniczącej systemu *Comcute*. Na rysunku 3.11 przedstawiono wykresy obciążenia dla węzła, w którym działa agent klasy W, a na rysunku 3.12 – dla węzła, w którym działa agent klasy S. Obciążenie procesorów (rys. 3.11a i 3.12a) znormalizowano na jeden rdzeń CPU. Wartości przekraczające 100% oznaczają, że agenty wykorzystywały kilka rdzeni równocześnie pracując wielowątkowo. Pomiary wykonano przy użyciu oprogramowania *Dstat* [267] do monitorowania obciążenia systemów z jądrem *Linux*.

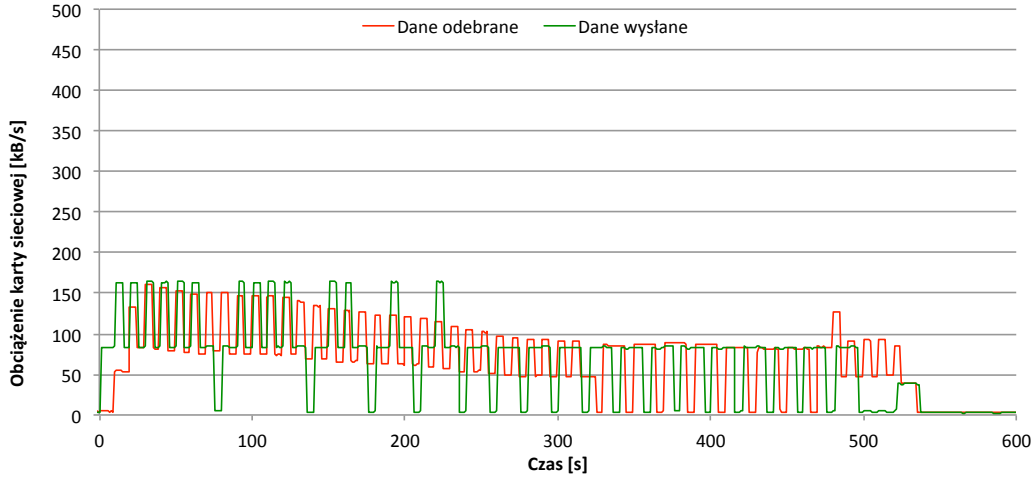
W przypadku agenta klasy W największe obciążenie procesora występuje w początkowej fazie obsługi zadania, która obejmuje partycjonowanie i przygotowanie paczek danych dla węzłów obliczeniowych. Towarzyszy temu wzrost zajętości pamięci operacyjnej o ok. 400 MB dla zadania obejmującego 128 tys. paczek danych. Serwer *Glassfish* wspomagający agenta klasy W rezerwuje dodatkowo 512 MB pamięci RAM. Pozostała pamięć operacyjna węzła jest wykorzystywana przez inne aplikacje. Wzrost obciążenia CPU zauważalny jest również w fazie konsolidacji, gdy scalane są wyniki otrzymane od agentów typu S.



(a) Obciążenie procesora

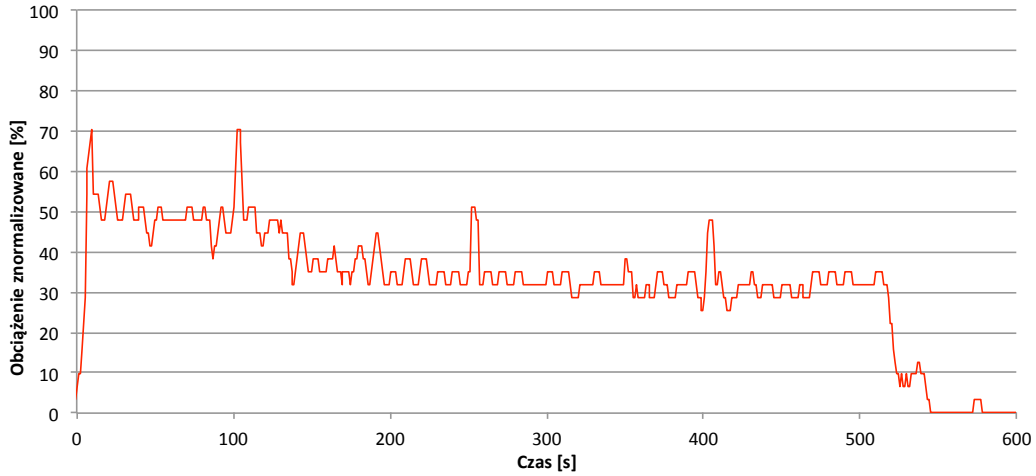


(b) Obciążenie pamięci operacyjnej

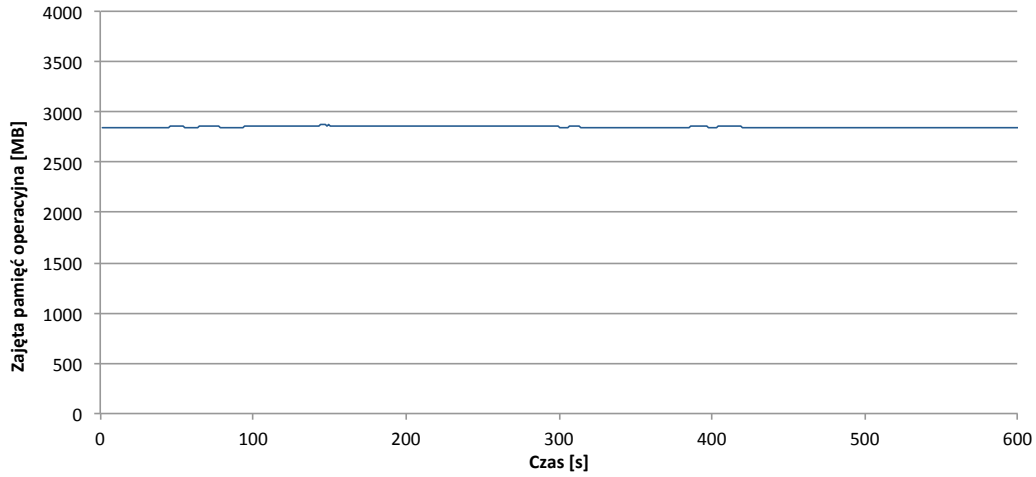


(c) Obciążenie karty sieciowej

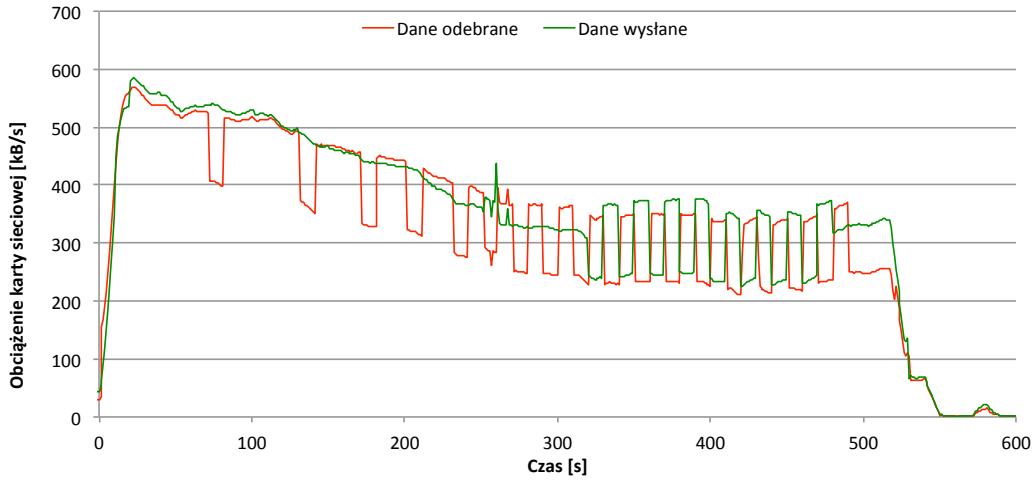
Rysunek 3.11: Obciążenie węzła, w którym działa agent klasy W
Źródło: opracowanie własne



(a) Obciążenie procesora



(b) Obciążenie pamięci operacyjnej



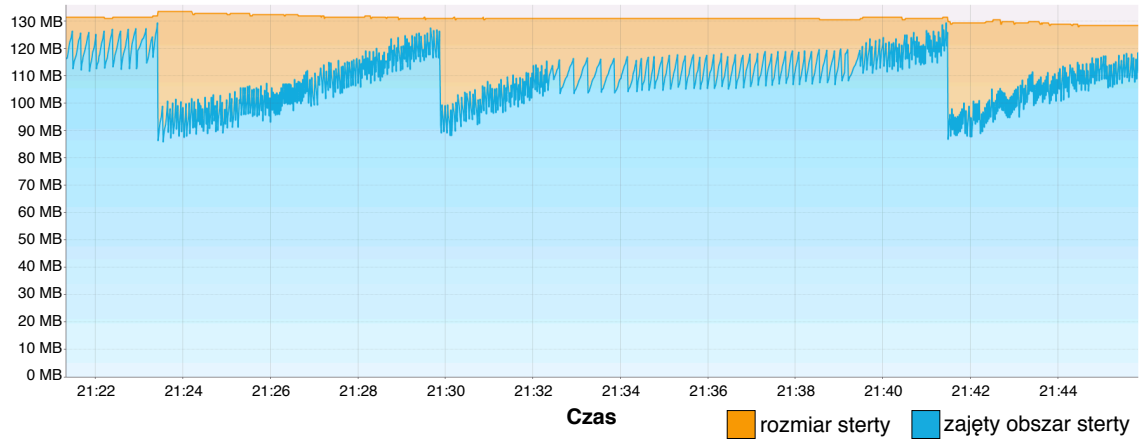
(c) Obciążenie karty sieciowej

Rysunek 3.12: Obciążenie węzła, w którym działa agent klasy S
Źródło: opracowanie własne

W wypadku agentów klasy S największe obciążenie procesora występuje w początkowej fazie obsługi zadania. Zajętość pamięci operacyjnej utrzymuje się natomiast na stałym poziomie. Zaobserwowane charakterystyki wynikają z faktu, że agent typu S zarządza buforem paczek wejściowych zawierającym jedynie część danych zadania (w produkcyjnej instalacji systemu – 2000 paczek). Jeśli bufor agenta typu S jest bliski wyczerpania, kolejne paczki są pobierane od agenta typu W. Z kolei odpowiedzi otrzymane od węzłów obliczeniowych umieszczane są w buforze zwrotnym (w wersji produkcyjnej o rozmiarze 1000 odpowiedzi). Jeśli bufor zwrotny zostanie wypełniony, odpowiedzi są odsyłane do agenta klasy W. Oznacza to, że niezależnie od liczby paczek danych dla zadania, rozmiar danych utrzymywanych przez agenta klasy S ograniczony jest wartością stałą.

Na wykresie zajętości pamięci operacyjnej nie są widoczne wahania wynikające z opróżniania buforów agenta typu S. Wynika to ze sposobu zarządzania pamięcią przez *JVM* – maszynę wirtualną *Javy*. *JVM* dla serwera *Glassfish* rezerwuje 512 MB pamięci RAM. Następnie, w czasie pracy agenta maszyna wirtualna przydziela pamięć dla konstruowanych obiektów w ramach zarezerwowanej puli. Zmienia się zatem zajętość sterty w obrębie *JVM*, ale z punktu widzenia systemu operacyjnego, agent cały czas wykorzystuje taką samą wielkość pamięci RAM.

Aby zbadać obciążenie pamięci przez maszynę wirtualną *Javy*, wykorzystano narzędzie *JVisualVM*. Na rysunku 3.13 przedstawiono zajętość sterty, którą zarządza *JVM*. W czasie realizacji zadania, rozmiar zajętej pamięci waha się w przedziale 85-130 MB. Oznacza to, że domyślnie rezerwowane 512 MB pozostawia duży zapas na obsługę kolejnych zadań. Obciążenie pamięci RAM przez agenta typu S można oszacować przez wartość stałą na poziomie 512 MB. Mniejsze rozmiary pamięci początkowej nie są zalecane przez producenta serwera *Glassfish* [208].



Rysunek 3.13: Obciążenie sterty maszyny wirtualnej *Javy* przez agenta klasy S
Źródło: opracowanie własne

Podsumowując, czasy przetwarzania danych przez agenty klasy S zależą od liczby paczek danych, które podlegają dystrybucji, podobnie jak czasy komunikacji między agentami klasy S i W. Czasy komunikacji w obrębie grupy roboczej $\mathcal{W}_m$ związane są z czasem realizacji zadania. Czasy przetwarzania danych przez agenty klasy W zależą od wielkości danych i od liczby współpracujących agentów klasy S. Zajętość pamięci operacyjnej dla agentów klasy S utrzymuje

się na stałym poziomie. Zajętość pamięci RAM przez agenta typu W zależy od wielkości przetwarzanych danych.

3.4 Model współpracy agentów w gridzie *Comcute*

Dla gridu *Comcute* opracowano w rozprawie model współpracy agentów uwzględniający parametry istotne dla działania systemu. Oprócz agentów warstwy pośredniczącej, wprowadzono agenty oferujące dodatkowe funkcjonalności. Na model składają się następujące elementy:

- $\mathscr{A} = \{\alpha_1, \ldots, \alpha_v, \ldots, \alpha_V\}$ – zbiór agentów typu W i S,
- $\mathscr{B} = \{\beta_1, \ldots, \beta_j, \ldots, \beta_J\}$ – zbiór typów komputerów do wykorzystania w gridzie,
- $\xi = [\xi_1, \ldots, \xi_j, \ldots, \xi_J]$ – wektor kosztów zakupu komputerów ze zbioru $\mathscr{B}$,
- $\Omega = \{\omega_1, \ldots, \omega_i, \ldots, \omega_I\}$ – zbiór węzłów, w których rozlokowane są agenty ze zbioru $\mathscr{A}$,
- w – macierz powiązań między agentami typu W,
- s – macierz powiązań między agentami typu W i S,
- $\{z_1, \ldots, z_m, \ldots, z_M\}$ – zbiór zadań obliczeniowych realizowanych w systemie,
- $GP = \{gp_1, \cdots, gp_l, \cdots, gp_L\}$ – zbiór inteligentnych agentów wykorzystujących programowanie genetyczne do optymalizacji strategii zespołu agentów warstwy pośredniczącej.

Każdy j-ty rodzaj komputera ze zbioru $\mathscr{B}$ scharakteryzowany jest przez poniższe parametry:

- ϑ_j – wydajność wg zadanego benchmarku, zazwyczaj wyrażona w [FLOPS],
- ram_j – wielkość pamięci RAM [GB],
- hdd_j – wielkość pamięci dyskowej HDD [TB],
- ssd_j – wielkość pamięci na dysku półprzewodnikowym SSD [TB],
- ξ_j – koszt zakupu [JM],
- ε_j – pobór mocy elektrycznej [W],
- γ_j – parametr niezawodności,
- ν_j – liczba zainstalowanych komputerów danego typu.

Zadanie obliczeniowe z_m opisywane jest przez:

- $L(z_m)$ – liczbę paczek danych wejściowych,
- $P_W(z_m)$ – parametr wydajnościowy określający liczbę agentów klasy W przetwarzających dane,
- $P_R(z_m)$ – stopień replikacji danych zadania,
- $P_T(z_m)$ – średni czas przetwarzania pojedynczej paczki danych na referencyjnym węźle obliczeniowym [s].

Podstawowe kryteria oceny jakości systemu rozproszonego dotyczą:

- obciążenia procesorów newralgicznego komputera,
- obciążenia komunikacyjnego newralgicznego komputera,
- łącznego obciążenia procesorów w gridzie,
- łącznego obciążenia komunikacyjnego w gridzie,
- łącznej wydajności gridu,

- kosztu zakupu hostów,
- kosztu przetwarzania danych i komunikacji,
- stopnia rozproszenia agentów,
- wielkości dostępnej pamięci RAM w newralgicznym hoście,
- wielkości dostępnej pamięci HDD w newralgicznym hoście,
- poboru mocy elektrycznej,
- dostępności gridu.

Obciążenia węzłów systemu generowane przez przypisane do nich agenty typu W i S, mogą być szacowane na podstawie wyprowadzonych zależności, które zebrano w tabeli nr 3.4. Model może zostać zaadaptowany również do innych systemów typu grid po wyznaczeniu dla nich oszacowań obciążeń węzłów.

Tabela 3.4: Metody szacowania obciążeń agentów typu W i S w gridzie *Comcute*

Agent	Parametr	Oszacowanie	R^2
S	Czas przetwarzania [s]	$t_v^{(S)}(z_m) = 3 \cdot 10^{-4} \cdot L_S^{\alpha_v}(z_m) - 0{,}545$	99,6%
	Czas komunikacji z agentem klasy W [s]	$\tau_{u,v}^{(SW)}(z_m) = 10^{-4} L_S^{\alpha_u}(z_m) - 0{,}1$	98,9%
	Pamięć operacyjna [MB]	512	–
W	Czas przetwarzania [s]	$t_v^{(W)}(z_m) = \frac{L_W(z_m)}{L_W^{\text{ref}}}\left(6{,}528\left\|\mathcal{S}^{(\alpha_v)}\right\| + 152{,}49\right)$	97,7%
	Czas komunikacji z agentami klasy S [s]	$\tau_{v,u}^{(WS)}(z_m) = \tau_{u,v}^{(SW)}(z_m)$	98,9%
	Czas komunikacji z agentami klasy W dla W_m^0 [s]	$\tau_{v,u}^{(WW)}(z_m) = 0{,}15 \cdot \lvert\mathcal{W}_m\rvert + 2{,}8 \cdot 10^{-3}\ \widehat{T} + 1{,}13$	98,7%
	Czas komunikacji z agentami klasy W (dla pozostałych agentów z grupy $\mathcal{W}_m$) [s]	$\tau_{v,u}^{(WW)}(z_m) = 0{,}15 + 2{,}8 \cdot 10^{-3}\ \widehat{T} + 1{,}13$	98,7%
	Pamięć operacyjna [MB]	$512 + 400 \cdot L_W(z_m)/L_W^{\text{ref}}$	–

Źródło: opracowanie własne

3.5 Wnioski i uwagi

Gridy komputerowe wykorzystuje się do współdzielenia i koordynacji rozproszonych zasobów należących do niezależnych podmiotów, które zrzeszają się w wirtualne organizacje dla realizacji wspólnych celów. Warto zauważyć, że zasoby obejmować mogą nie tylko moc obliczeniową, ale również pamięć dyskową i operacyjną, specjalistyczne urządzenia, czujniki, aplikacje czy usługi. Systemy gridowe pełnią istotną rolę w badaniach i edukacji.

Pakiet *Globus Toolkit* [104] jest szeroko stosowanym oprogramowaniem warstwy pośredniczącej dla gridów komputerowych. Obejmuje on moduły odpowiedzialne za wykonanie zadań obliczeniowych, dostęp do danych, wyszukiwanie i monitorowanie usług oraz bezpieczeństwo systemu. Pakiet zawiera mechanizmy poufnej wymiany informacji oraz mechanizmy uwierzytelniania i autoryzacji. Oprogramowanie udostępnia również narzędzia ułatwiające implementację aplikacji rozproszonych.

Ważną grupę wśród gridów komputerowych stanowią platformy obliczeń wolontariackich. W tym przypadku podmiotami budującymi wirtualną organizację są wolontariusze oferujący moc obliczeniową swoich komputerów oraz koordynator zadań, który zarządza modułami obliczeniowymi i danymi do przetworzenia. Systemy tego typu gromadzą wolontariuszy, oferując interesujące zadania obliczeniowe, np. wyszukiwanie pulsarów w przestrzeni kosmicznej czy badanie zmian zachodzących w białkach organizmów żywych.

W wielu gridach wolontariackich wykorzystuje się infrastrukturę *BOINC*. Posiada ona mechanizmy dystrybucji zadań i negocjowania parametrów ich wykonania na komputerze wolontariusza. Architektura gridu zorientowana jest na skalowalność i wysoki poziom rozproszenia komponentów systemu. Uwzględnia ona również mechanizmy tolerancji awarii i wznawiania obliczeń w wypadku ich przerwania. Czas realizacji zadań w gridach wolontariackich zależy od rodzaju projektu i może wahać się od kilkunastu sekund do kilkuset godzin.

Opracowany na Politechnice Gdańskiej grid *Comcute* cechuje się możliwością zwiększenia mocy obliczeniowej poprzez przyłączenie komputerów wolontariuszy, które wyposażono w przeglądarkę internetową. Nie występuje konieczność instalacji dedykowanego oprogramowania klienckiego. Jest to szczególnie istotne w sytuacjach kryzysowych, gdy wykorzystana może zostać moc komputerów instytucji administracji państwowej bez konieczności interwencji administratora grida. W takim wypadku udziałowcami wirtualnej organizacji są Politechnika Gdańska, zaangażowane instytucje oraz anonimowi wolontariusze.

W systemie *Comcute* wykonywane są zadania, takie jak: symulacja rozprzestrzeniania się pożaru w oparciu o model automatu komórkowego [10, 148], analiza skutków awarii elektrowni atomowej, weryfikacja hipotezy *Collatza* [152] czy wyznaczanie liczb pierwszych *Mersenne'a* [150]. Możliwe są również zastosowania w dziedzinie audytów bezpieczeństwa [151]. Każde zadanie posiada moduł obliczeniowy uruchamiany na komputerach wolontariuszy, *partycjoner* odpowiedzialny za wygenerowanie paczek danych do przetworzenia oraz *linker* obliczający końcowy wynik na podstawie wyników cząstkowych dla paczek danych.

Przeprowadzone pomiary dla gridu *Comcute* pokazują, że jego zmodernizowana wersja może obsłużyć znaczne obciążenie ze strony wolontariuszy. Równocześnie, uzyskane wyniki pozwalają na określenie metod szacowania obciążeń agentów systemu. Dzięki uzyskanym oszacowaniom w trakcie optymalizacji gridu możliwe jest rozważanie strategii zespołu agentów warstwy pośredniczącej, które nie były testowane w systemie. Zdefiniowany model współpracy agentów w gridzie *Comcute* jest podstawą do sformułowania problemów optymalizacji dla rozpatrywanego systemu.

Rozdział 4

Problemy optymalizacji strategii zespołu inteligentnych agentów w środowisku wirtualnym

4.1 Zbiór strategii dopuszczalnych w podstawowym modelu gridu

Odpowiedni dobór strategii zespołu agentów warstwy pośredniczącej jest kluczowy dla efektywnego wykorzystania dostępnych zasobów systemu. Wybrana strategia powinna odzwierciedlać wskazane przez interesariuszy kryteria oceny jakości i zadane ograniczenia. Dobór typów komputerów do wykorzystania w węzłach, odpowiednie rozmieszczenie agentów oraz właściwy przydział zadań obliczeniowych pozwalają na optymalizację strategii pod kątem wydajności systemu, kosztu zakupu serwerów, zużycia energii, dostępności gridu lub równoważenia obciążeń.

Wdrażana strategia działania inteligentnych agentów musi uwzględniać obciążenia występujące w gridzie. Niech obciążenie CPU powodowane przez agenty klasy W i S zadane jest w postaci macierzy skumulowanych czasów przetwarzania danych: $T = [t_{vj}]_{V \times J}$, gdzie t_{vj} oznacza skumulowany czas pracy agenta α_v na komputerze rodzaju β_j [209]. Pod pojęciem czasu cząstkowego obsługi agenta rozumiany jest czas jego obsługi przez CPU od chwili rozpoczęcia lub wznowienia do momentu zakończenia wykonania lub przełączenia na obsługę innego procesu. Czasem skumulowanym nazywana jest suma czasów cząstkowych obsługi agenta [40]. Z kolei łączny czas pracy agenta jest sumą czasów skumulowanych obsługi zadań, które zostały do niego przypisane.

Obciążenia agentów można wyznaczyć w oparciu o zależności (3.8) i (3.10), jak niżej:

$$t_{vj} = \begin{cases} \dfrac{\vartheta_{\text{ref}}}{\vartheta_j} \displaystyle\sum_{z_m \in \mathscr{Z}} t_v^{(W)}(z_m) \cdot |\mathcal{W}_m \cap \{\alpha_v\}|, & \text{dla } v = \overline{1, V_W}, \\ \dfrac{\vartheta_{\text{ref}}}{\vartheta_j} \displaystyle\sum_{z_m \in \mathscr{Z}} t_v^{(S)}(z_m) \cdot |\mathcal{S}_m \cap \{\alpha_v\}|, & \text{dla } v = \overline{V_W + 1, V}, \end{cases} \tag{4.1}$$

gdzie:
ϑ_{ref} – miara wydajności komputera referencyjnego wykorzystanego do wyznaczenia charakterystyk systemu *Comcute* [w jednostkach benchmarku],
ϑ_j – miara wydajności komputera typu β_j [w jednostkach benchmarku].

Niech $\tau = [\tau_{vu}]_{V \times V}$ jest macierzą skumulowanych czasów komunikacji, gdzie τ_{vu} oznacza czas transmisji między agentami α_v i α_u [s]. Zakłada się, że $\tau_{vv} = 0$ dla $v = \overline{1,V}$, gdyż komunikacja wewnątrz agenta nie wymaga wykorzystania sieci komputerowej. Dodatkowo przyjmuje się, że pomiędzy agentami pracującymi w tym samym węźle nie występują opóźnienia komunikacyjne. W gridzie *Comcute* komunikacja odbywa się pomiędzy agentami typu W, a także między agentem klasy W a przypisanymi do niego agentami typu S. Agenty typu S nie komunikują się ze sobą bezpośrednio, więc $\tau_{vu} = 0$, gdy $v = \overline{V_W + 1, V}$ i $u = \overline{V_W + 1, V}$. Wartości elementów macierzy τ można wyznaczyć w oparciu o opisane powyżej charakterystyki oraz zależności (3.9) i (3.11):

$$\tau_{vu} = \begin{cases} 0, & \text{gdy } v = \overline{V_W + 1, V}, u = \overline{V_W + 1, V}, \\ 0, & \text{gdy } \alpha_v \text{ i } \alpha_u \text{ działają w tym samym węźle}, \\ 0, & \text{gdy } v = \overline{1, V_W}, u = \overline{V_W + 1, V}, u \notin S^{(\alpha_v)}, \\ \displaystyle\sum_{m=1}^{M} \tau_{v,u}^{(WW)}(z_m), & \text{gdy } v = \overline{1, V_W}, u = \overline{1, V_W}, \\ \displaystyle\sum_{m=1}^{M} \tau_{v,u}^{(WS)}(z_m), & \text{gdy } v = \overline{1, V_W}, u = \overline{V_W + 1, V}, u \in S^{(\alpha_v)}. \end{cases} \tag{4.2}$$

Czasy przetwarzania i komunikacji agentów zależą od obciążeń wnoszonych przez zadania użytkownika. Kluczowy jest zatem przydział zadań ze zbioru $\{z_1, \ldots, z_m, \ldots, z_M\}$ do agentów W_m^0, które zajmują się ich dalszą obsługą w systemie. Przyporządkowanie to definiowane jest za pomocą następującej macierzy binarnej x^z:

$$x^z = [x_{mv}^z]_{M \times V_W}, \tag{4.3}$$

gdzie $x_{mv}^z = \begin{cases} 1, & \text{gdy agent } \alpha_v \text{ rozpoczyna obsługę zadania } z_m, \\ 0, & \text{w przeciwnym razie,} \end{cases}$ dla $m = \overline{1, M}, v = \overline{1, V_W}$.

Każde zadanie z_m przypisywane jest do jednego agenta W_m^0, co wyraża się w postaci ograniczenia:

$$\sum_{v=1}^{V_W} x_{mv}^z = 1, \text{ dla } m = \overline{1, M}. \tag{4.4}$$

Ograniczenie (4.4) może zostać spełnione poprzez specyfikację przydziału zadań do agentów W_m^0 za pomocą wektora całkowitoliczbowego $X^z = [X_1^z, \ldots, X_m^z, \ldots, X_M^z]^T$, gdzie X_m^z odpowiada indeksowi v agenta $\alpha_v = W_m^0$ dla zadania z_m. Gdy $x_{mv}^z = 1$, to $X_m^z = v$, przy zachowaniu ograniczeń: $1 \leqslant X_m^z \leqslant V_W$, $m = \overline{1,M}$.

Rozmieszczenie agentów w węzłach gridu opisać można za pomocą binarnej macierzy, jak niżej:

$$x^\alpha = [x_{vi}^\alpha]_{V \times I} \,, \tag{4.5}$$

gdzie $x_{vi}^\alpha = \begin{cases} 1, & \text{gdy agent } \alpha_v \text{ migruje do węzła } \omega_i, \\ 0, & \text{w przeciwnym razie,} \end{cases}$ dla $v = \overline{1,V}, i = \overline{1,I}$.

W każdym węźle może działać kilka agentów, przy czym pojedynczy agent pracuje w jednym węźle, co wyraża poniższe ograniczenie:

$$\sum_{i=1}^{I} x_{vi}^\alpha = 1, \text{ dla } v = \overline{1,V}. \tag{4.6}$$

Podobnie jak w wypadku macierzy przydziału zadań (4.3) i ograniczenia (4.4), zastosować można wektorowy opis migracji agentów do węzłów w postaci: $X^\alpha = [X_1^\alpha, \ldots, X_v^\alpha, \ldots, X_V^\alpha]^T$, gdzie X_v^α to indeks i węzła ω_i, do którego powinien przemieścić się agent α_v. Jeśli $x_{vi}^\alpha = 1$, to $X_v^\alpha = i$, przy ograniczeniach: $1 \leqslant X_v^\alpha \leqslant I$, $v = \overline{1,V}$. Przydział opisany wektorem X^α spełnia ograniczenie (4.6).

Z kolei przypisanie typów komputerów ze zbioru $\mathscr{B} = \{\beta_1, \ldots, \beta_j, \ldots, \beta_J\}$ do węzłów systemu przedstawia binarna macierz:

$$x^\beta = [x_{ij}^\beta]_{I \times J}, \tag{4.7}$$

gdzie $x_{ij}^\beta = \begin{cases} 1, & \text{gdy do węzła } \omega_i \text{ przypisano komputer typu } \beta_j, \\ 0, & \text{w przeciwnym razie,} \end{cases}$ dla $i = \overline{1,I}, j = \overline{1,J}$.

Zakłada się, że w węźle (np. w slocie szafy rackowej) umieszczony jest pojedynczy komputer, co można przedstawić jako ograniczenie:

$$\sum_{j=1}^{J} x_{ij}^\beta = 1, \text{ dla } i = \overline{1,I}. \tag{4.8}$$

Spełnienie ograniczenia (4.7) jest możliwe za pomocą wektora przydziału typów komputerów do węzłów: $X^\beta = [X_1^\beta, \ldots, X_i^\beta, \ldots, X_I^\beta]^T$, gdzie X_i^β to indeks j typu komputera β_j w węźle ω_i. Jeżeli $x_{ij}^\beta = 1$, to $X_i^\beta = j$, przy ograniczeniach: $1 \leqslant X_i^\beta \leqslant J$, $i = \overline{1,I}$. Warto zwrócić uwagę, że ograniczenia (4.4), (4.6) oraz (4.8) stanowią układ $M + V + I$ równań.

Przez *strategię zespołu inteligentnych agentów w gridzie* rozumiana jest uporządkowana trójka macierzy binarnych określających przypisanie zadań obliczeniowych do agentów, migrację agentów do węzłów oraz przyporządkowanie typów komputerów do węzłów:

$$x = (x^z, x^\alpha, x^\beta). \tag{4.9}$$

Liczba binarnych zmiennych decyzyjnych w trzech macierzach wynosi $MV_W + I(V + J)$. Rozwiązania należą zatem do przestrzeni $\mathbb{B}^{MV_W+I(V+J)}$, gdzie $\mathbb{B} = \{0,1\}$. Z tego powodu binarna przestrzeń przeszukiwań zawiera $2^{MV_W+I(V+J)}$ elementów. Podstawowy *zbiór możliwych strategii* $\boldsymbol{\mathcal{X}}$ obejmuje rozwiązania spełniające ograniczenia formalne (4.4), (4.6) oraz (4.8):

$$\begin{aligned}\boldsymbol{\mathcal{X}} = \{x \in \mathbb{B}^{MV_W+I(V+J)} : & \sum_{v=1}^{V_W} x^z_{mv} = 1, \text{ dla } m = \overline{1,M}; \\ & \sum_{i=1}^{I} x^\alpha_{vi} = 1, \text{ dla } v = \overline{1,V}; \\ & \sum_{j=1}^{J} x^\beta_{ij} = 1, \text{ dla } i = \overline{1,I}\}.\end{aligned} \tag{4.10}$$

Twierdzenie 4.1.1. *Jeżeli $V_W \geqslant 1$, $V \geqslant 2$, $V_W < V$, $M \geqslant 1$, $I \geqslant 1$ oraz $J \geqslant 1$, to zbiór rozwiązań dopuszczalnych $\boldsymbol{\mathcal{X}}$ opisany zależnością (4.10) nie jest zbiorem pustym i zawiera ${V_W}^M I^V J^I$ strategii zespołu agentów warstwy pośredniczącej gridu.*

Dowód. Zgodnie z ograniczeniem formalnym (4.4) pojedynczy wiersz macierzy x^z może zawierać wartość 1 tylko w wybranej kolumnie. Równocześnie nie jest możliwe pozostawienie nieprzypisanego zadania, więc wartość 1 musi wystąpić dokładnie raz. Przykładowo, jeśli zadanie z_m przydzielono do agenta $\alpha_v = W^0_m$, macierz x^z ma następującą postać:

$$x^z = \begin{bmatrix} x^z_{1,1} & \cdots & x^z_{1,v-1} & x^z_{1,v} & x^z_{1,v+1} & \cdots & x^z_{1,V_W} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x^z_{m-1,1} & \cdots & x^z_{m-1,v-1} & x^z_{m-1,v} & x^z_{m-1,v+1} & \cdots & x^z_{m-1,V_W} \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ x^z_{m+1,1} & \cdots & x^z_{m+1,v-1} & x^z_{m+1,v} & x^z_{m+1,v+1} & \cdots & x^z_{m+1,V_W} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x^z_{M,1} & \cdots & x^z_{M,v-1} & x^z_{M,v} & x^z_{M,v+1} & \cdots & x^z_{M,V_W} \end{bmatrix}_{M \times V_W} \tag{4.11}$$

Oznacza to, że w macierzy x^z istnieje V_W dopuszczalnych wariantów dla każdego z M wierszy, czyli ${V_W}^M$ dopuszczalnych wariantów macierzy, gdyż nie ma ograniczenia na liczbę zadań przypisanych do pojedynczego agenta. Skoro $V_W \geqslant 1$, to możliwe jest wybranie przynajmniej jednego agenta dla każdego z zadań. Z kolei warunek $M \geqslant 1$ gwarantuje, że istnieje przynajmniej jedno zadanie do przydzielenia.

Analogiczne rozumowanie zastosowane do macierzy migracji agentów do węzłów x^α i ograniczenia (4.6) oraz macierzy przydziału typów komputerów do węzłów x^β i ograniczenia (4.8) prowadzi do wniosku, że istnieje I^V macierzy x^α oraz J^I macierzy x^β, które reprezentują dopuszczalne przypisania. Każde z przyporządkowań odbywa się niezależnie od pozostałych, więc liczba dopuszczalnych strategii agentów w zbiorze $\boldsymbol{\mathcal{X}}$ wynosi ${V_W}^M I^V J^I$, co kończy dowód. □

Binarne macierze x^z, x^α i x^β pozwalają na wygodne formułowanie ograniczeń, funkcji kryterialnych i zagadnień optymalizacji. Jednak w metodach optymalizacji, w tym w programowaniu genetycznym, preferowane są reprezentacje wektorowe X^z, X^α i X^β. Ograniczają one przestrzeń przeszukiwań poprzez redukcję zbioru rozwiązań możliwych do zakodowania do tych, które spełniają wymagania formalne (4.4), (4.6) oraz (4.8). Trójka $x = (x^z, x^\alpha, x^\beta)$ kodować może $2^{MV_W+I(V+J)}$ rozwiązań, podczas gdy tylko ${V_W}^M I^V J^I$ z nich należy do zbioru strategii dopuszczalnych. Liczba kroków wymaganych do weryfikacji ograniczeń formalnych dla kodowania rozwiązań w postaci trójki x jest proporcjonalna do $MV_W + VI + IJ$.

W metodzie programowania genetycznego proponuje się wykorzystanie reprezentacji strategii agentów w postaci wektora X, który zawiera składowe zdefiniowanych wcześniej całkowitoliczbowych wektorów X^z, X^α i X^β:

$$X = [X_1^z, \ldots, X_m^z, \ldots, X_M^z, X_1^\alpha, \ldots, X_v^\alpha, \ldots, X_V^\alpha, X_1^\beta, \ldots, X_i^\beta, \ldots, X_I^\beta]^T, \tag{4.12}$$

przy ograniczeniach:

$$1 \leqslant X_m^z \leqslant V_W, \tag{4.13}$$

$$1 \leqslant X_v^\alpha \leqslant I, \tag{4.14}$$

$$1 \leqslant X_i^\beta \leqslant J. \tag{4.15}$$

Przy spełnieniu ograniczeń (4.13)-(4.15) za pomocą wektora X można kodować ${V_W}^M I^V J^I$ rozwiązań. Liczba kroków wymaganych do weryfikacji spełnienia ograniczeń formalnych dla wektorowej reprezentacji strategii agentów jest proporcjonalna do $M + V + I$. Zysk z kodowania całkowitoliczbowego rośnie wraz ze wzrostem liczby zadań, agentów i węzłów w rozpatrywanym modelu.

Przy formułowaniu zagadnień optymalizacji strategii zespołu agentów warstwy pośredniczącej gridu, poza ograniczeniami formalnymi, brane są pod uwagę również ograniczenia w zakresie oferowanej wydajności systemu, kosztów pozyskania infrastruktury i jej działania, czy stopnia rozproszenia agentów. Uwzględnia się również wymagania odnoszące się do wielkości dostępnych zasobów (np. pamięci operacyjnej i dyskowej) czy ograniczenia energetyczne [116]. Jednym z kluczowych kryteriów jest *łączna moc obliczeniowa* węzłów systemu według wybranego benchmarku wydajności.

W przypadku aplikacji rozproszonych wydajność może być rozumiana jako liczba żądań obsługi, jaką system przetwarza w ciągu sekundy. Testy pozwalające na ocenę systemu pod tym kątem można zaprojektować posługując się narzędziami takimi jak *JMeter* [18], *SoapUI* [243], czy *Postman* [220]. Wydajność superkomputerów oceniana jest zazwyczaj przy użyciu oprogramowania *SPEC* [246] lub – stosowanego w projekcie *TOP500* – testu *Linpack* [251]. Testy te odnoszą się do wydajności CPU i GPU badanych systemów.

W systemach typu grid badana może być wydajność CPU poszczególnych komputerów. Wykorzystywane są do tego narzędzia, takie jak *CINEBENCH* [189] czy *Linpack*. Jeśli w gridach występują również urządzenia mobilne, mogą być testowane przy użyciu aplikacji *AnTuTu*

lub *Quadrant*. Wydajność gridu scharakteryzowana może być jako suma wydajności jego składowych, wartość uśredniona lub wydajność newralgicznego węzła.

Dla gridów komputerowych bardziej reprezentatywne są benchmarki, które uwzględniają interakcje pomiędzy komponentami, a także typowe zadania realizowane w systemie. Przykładem tego typu benchmarka jest test zaprojektowany dla systemu *SAS Grid Manager* [122]. Badana była konfiguracja obejmująca serwery *HP ProLiant BL460c G7 Blades* oraz macierz dyskową *HP 3PAR T800*. W zbiorze zadań testowych 50% operacji generowało obciążenie CPU, a pozostałe 50% – obciążenie pamięci masowej. Wykonywane operacje obejmowały czynności typowe dla zadań realizowanych w systemie *SAS Grid Manager*, w tym zapytania SQL, algorytmy analizy ryzyka, algorytmy logistyczne, sortowanie zbiorów danych, czy operacje agregujące.

Badano przepustowość macierzy dyskowej oraz czas realizacji scenariusza dla różnych konfiguracji zadań i serwerów. Dla 2160 zadań uruchomionych na 15 serwerach uzyskano przepustowość macierzy na poziomie 5,01 GB/s, co stanowi blisko 95% teoretycznej przepustowości określanej przez producenta na poziomie 5,3 GB/s. Zwiększenie liczby zadań skutkowało pogorszeniem wydajności macierzy dyskowej [122].

Ważnym benchmarkiem ukierunkowanym na badanie gridów obliczeniowych jest opracowany przez agencję *NASA* zestaw testów *NAS Parallel Benchmarks* [259]. Uwzględnia on realizację komunikujących się ze sobą, rozproszonych zadań. Podstawowym wynikiem zwracanym przez benchmark jest czas realizacji scenariusza testowego. Dodatkowo, wyznaczane są wartości dotyczące łącznego wykorzystania zasobów: czas pracy CPU, zajętość pamięci RAM i HDD, a także przepustowość kanałów komunikacyjnych. Raport z testu obejmuje łączne czasy komunikacji między parami zadań, a także rzeczywiste czasy wykonania (ang. *wall clock time*) każdego zadania z osobna w warunkach obciążenia komputerów innymi zadaniami oraz pracą oprogramowania gridowego. W takim przypadku uwzględniane są czasy oczekiwania zadania na dostępność procesora. Pomiar obejmuje cały okres od momentu rozpoczęcia od zakończenia zadania.

Czas zajętości procesora w związku z obsługą zadania (ang. *CPU time*) może być krótszy, gdyż obejmuje wykonanie kodu zadania oraz czas realizacji wywołań systemowych inicjowanych przez zadanie (np. odwołania do urządzeń wejścia/wyjścia). Natomiast najkrótszy z mierzonych czasów to użytkowy czas wykonania zadania (ang. *user time*), który obejmuje jedynie wykonanie kodu zadania bez uwzględnienia czasu realizacji wywołań systemowych.

Zestaw testów *NAS Parallel Benchmarks* obejmuje cztery kategorie scenariuszy testowych do oceny wydajności systemu:

- ED (ang. *Embarrassingly Distributed*) – wykonywanie w różnych węzłach tego samego zadania *Scalar Penta-diagonal solver* (SP) polegającego na rozwiązywaniu liniowego układu równań z różnymi parametrami. Elementy niezerowe w macierzy współczynników występują jedynie na pięciu przekątnych: przekątnej głównej oraz dwóch przekątnych nad i pod przekątną główną;
- HC (ang. *Helical Chain*) – łańcuch zadań składający się z wielokrotnego powtarzania sekwencji trzech modułów: BT (ang. *Block Tridiagonal solver* – rozwiązywanie blokowego układu

równań dla macierzy, w której niezerowe elementy występują jedynie na przekątnej głównej oraz nad i pod nią), SP oraz LU (rozwiązywanie układu równań metodą *Gaussa-Seidla*);

- VP (ang. *Visualization Pipeline*) – wielokrotne wykonanie sekwencji trzech zadań: SP, MG (*post-processing* wyników zadania SP) i FT (wizualizacja wyników zadania MG), przy czym wyniki zadania SP przekazywane są do kolejnej instancji zadań SP i MG. Dane do wizualizacji przekazywane są między kolejnymi instancjami zadania FT;
- MB (ang. *Mixed Bag*) – trzyetapowe, równoległe i synchroniczne wykonanie pewnej liczby zadań. W pierwszym etapie liczone są równolegle zadania klasy LU dla różnych danych wejściowych. Po synchronizacji wykonywany jest *post-processing* z wykorzystaniem zadań MG, a następnie w trzecim etapie równoległa wizualizacja FT.

Referencyjne implementacje zestawu testów *NAS Parallel Benchmarks* obejmują sekwencyjne i współbieżne odmiany zadań obliczeniowych i są dostępne w językach: *C*, *Java* oraz *Fortran*. Mogą być wykorzystywane w gridach zarządzanych przez oprogramowanie *Globus Toolkit*, a także w środowiskach klastrów obliczeniowych opartych o rozwiązanie *MPI*.

Łączna moc obliczeniowa gridu Θ może zostać wyznaczona w oparciu o wyniki wybranego benchmarku dla rozpatrywanej strategii x, jak niżej [209]:

$$\Theta(x) = \sum_{i=1}^{I} \sum_{j=1}^{J} \vartheta_j x_{ij}^{\beta}, \tag{4.16}$$

gdzie ϑ_j – wydajność komputera typu β_j oszacowana według wybranego benchmarku [w jednostkach testu].

Sformułować można ograniczenie na *minimalną wymaganą wydajność systemu* $\vartheta_{\min}$ [w jednostkach benchmarku], zgodnie z nierównością, jak niżej [22]:

$$\Theta(x) \geqslant \vartheta_{\min}. \tag{4.17}$$

Podobne ograniczenia można narzucić na wartość średnią wydajności komputerów w gridzie lub też na wydajność newralgicznego węzła.

Ważne wymaganie odnosi się do *kosztu zakupu komputerów gridu* $\Xi(x)$ [JM], który nie powinien przekroczyć limitu finansowego $\xi_{\max}$ [JM]. Koszt szacowany jest na podstawie wektora cen dostępnych typów komputerów $\xi = [\xi_1, \ldots, \xi_j, \ldots, \xi_J]$. Ograniczenie definiuje się następująco:

$$\Xi(x) \leqslant \xi_{\max}, \tag{4.18}$$

gdzie $\Xi(x) = \sum_{i=1}^{I} \sum_{j=1}^{J} \xi_j x_{ij}^{\beta}$.

Kolejne ograniczenie związane jest z zagwarantowaniem minimalnego stopnia rozproszenia agentów, który jest istotny z punktu widzenia poziomu zrównoleglania obliczeń oraz obsługi awarii węzłów. Jeśli wszystkie agenty przemieszczą się do jednego węzła, to jego awaria sparaliżuje działanie całego systemu. W przypadku rozproszenia agentów możliwa jest zmiana przypisania zadań

w celu zapewnienia ciągłości pracy gridu. Zakłada się, że powinno istnieć co najmniej $\eta_{\min}$ par agentów działających w różnych węzłach, przy czym $1 \leqslant \eta_{\min} \leqslant \frac{1}{2}V(V-1)$. Odpowiednie ograniczenie przedstawia poniższa nierówność:

$$\eta(x) \geqslant \eta_{\min}, \tag{4.19}$$

gdzie $\eta(x) = \sum_{v=1}^{V} \sum_{\substack{u=1 \\ u>v}}^{V} \sum_{i=1}^{I} \sum_{\substack{k=1 \\ k\neq i}}^{I} x_{vi}^{\alpha} x_{uk}^{\alpha}$.

Wymagania dotyczą również zasobów komputerów w postaci pamięci operacyjnej i pamięci trwałej. Niech ram_j oznacza wielkość pamięci RAM w komputerze j-tego typu [GB]. Przyjmuje się, że agent α_v rezerwuje r_v [GB] pamięci RAM, $r_v \geqslant 0$ dla $v = \overline{1,V}$. Przy zadanej strategii współdziałania agentów, wielkość pamięci r_v wyznaczyć można w oparciu o oszacowania poczynione w tabeli nr 3.4. Przyjmuje się, że $r_j^{\min}$ oznacza wielkość pamięci przeznaczonej na niezbędne procesy użytkowników, procesy systemu operacyjnego i sterowniki na komputerze j-tego rodzaju.

Niech hdd_j oznacza wielkość pamięci dyskowej w komputerze j-tego typu [TB]. Przyjmuje się, że agent α_v rezerwuje h_v [TB] pamięci HDD, $h_v \geqslant 0$ dla $v = \overline{1,V}$. Niech $h_j^{\min}$ oznacza wielkość pamięci dyskowej przeznaczonej na system operacyjny i plik stronicowania w komputerze j-tego rodzaju. Ograniczenia na pamięć RAM i HDD można zapisać następująco:

$$\sum_{v=1}^{V} r_v x_{vi}^{\alpha} \leqslant \sum_{j=1}^{J} \left(ram_j - r_j^{\min}\right) x_{ij}^{\beta}, \ i = \overline{1,I}, \tag{4.20}$$

$$\sum_{v=1}^{V} h_v x_{vi}^{\alpha} \leqslant \sum_{j=1}^{J} \left(hdd_j - h_j^{\min}\right) x_{ij}^{\beta}, \ i = \overline{1,I}. \tag{4.21}$$

Niech $\kappa_i^{\mathrm{RAM}}(x)$ [GB] oznacza wielkość rezerwy pamięci RAM w komputerze przypisanym do i-tego węzła. Wymaganie dostępności pamięci RAM o wystarczającej wielkości we wszystkich węzłach może być przedstawione jako I nierówności:

$$\kappa_i^{\mathrm{RAM}}(x) \geqslant 0, \ i = \overline{1,I}, \tag{4.22}$$

gdzie $\kappa_i^{\mathrm{RAM}}(x) = \sum_{j=1}^{J} \left(ram_j - r_j^{\min}\right) x_{ij}^{\beta} - \sum_{v=1}^{V} r_v x_{vi}^{\alpha}, \ i = \overline{1,I}$.

Komputerem newralgicznym ze względu na pamięć RAM nazywamy komputer usytuowany w węźle i^* takim, że:

$$\kappa_{\min}^{\mathrm{RAM}}(x) = \kappa_{i^*}^{\mathrm{RAM}}(x) = \min_{i=\overline{1,I}} \left\{\kappa_i^{\mathrm{RAM}}(x)\right\}. \tag{4.23}$$

Wówczas ograniczenie (4.22) można zapisać, jak następuje:

$$\kappa_{\min}^{\mathrm{RAM}}(x) \geqslant 0. \tag{4.24}$$

Niech $\kappa_i^{\mathrm{HDD}}(x)$ [TB] oznacza wielkość rezerwy pamięci HDD w komputerze usytuowanym w i-tym węźle. Analogicznie do rozważań odnośnie pamięci RAM otrzymujemy, jak niżej:

$$\kappa_i^{\mathrm{HDD}}(x) \geqslant 0,\ i = \overline{1,I}, \tag{4.25}$$

gdzie $\kappa_i^{\mathrm{HDD}}(x) = \sum_{j=1}^{J} \left(hdd_j - h_j^{\min}\right) x_{ij}^{\beta} - \sum_{v=1}^{V} h_v x_{vi}^{\alpha},\ i = \overline{1,I}.$

Dla węzła newralgicznego ze względu na pamięć dyskową zachodzą następujące zależności:

$$\kappa_{\min}^{\mathrm{HDD}}(x) = \kappa_{i^*}^{\mathrm{HDD}}(x) = \min_{i=\overline{1,I}} \left\{\kappa_i^{\mathrm{HDD}}(x)\right\}, \tag{4.26}$$

$$\kappa_{\min}^{\mathrm{HDD}}(x) \geqslant 0. \tag{4.27}$$

W pewnych zastosowaniach istotny jest dostęp do szybkiej pamięci masowej. Dla takich scenariuszy w węzłach gridu ulokowane mogą zostać komputery wyposażone w dysk półprzewodnikowy SSD. Niech ssd_j oznacza wielkość pamięci SSD w komputerze j-tego rodzaju [TB], a $\hat{h}_v$ – rozmiar pamięci tego typu, którą rezerwuje agent α_v [TB], przy czym $\hat{h}_v \geqslant 0$ dla $v = \overline{1,V}$. Ponadto, niech $\hat{h}_j^{\min}$ oznacza wielkość pamięci SSD przeznaczonej na system operacyjny i dodatkowo niech $\kappa_i^{\mathrm{SSD}}(x)$ oznacza wielkość rezerwy pamięci SSD w komputerze ulokowanym w i-tym węźle. Zagwarantowanie wystarczającego rozmiaru pamięci SSD dla działających w węzłach agentów wyrażone jest za pomocą I ograniczeń, jak niżej:

$$\kappa_i^{\mathrm{SSD}}(x) \geqslant 0,\ i = \overline{1,I}, \tag{4.28}$$

gdzie $\kappa_i^{\mathrm{SSD}}(x) = \sum_{j=1}^{J} \left(ssd_j - \hat{h}_j^{\min}\right) x_{ij}^{\beta} - \sum_{v=1}^{V} \hat{h}_v x_{vi}^{\alpha},\ i = \overline{1,I}.$

W przypadku węzła i^* newralgicznego ze względu na pamięć SSD zachodzi dodatkowo:

$$\kappa_{\min}^{\mathrm{SSD}}(x) = \kappa_{i^*}^{\mathrm{SSD}}(x) = \min_{i=\overline{1,I}} \left\{\kappa_i^{\mathrm{SSD}}(x)\right\}, \tag{4.29}$$

$$\kappa_{\min}^{\mathrm{SSD}}(x) \geqslant 0. \tag{4.30}$$

Warto podkreślić, że rezerwa pamięci RAM oraz rezerwy pamięci masowej HDD i SSD w komputerze newralgicznym mogą być wykorzystane nie tylko w ograniczeniach, ale także mogą być stosowane jako kryteria jakości.

Istotne ograniczenie odnosi się do *łącznego poboru mocy elektrycznej* $E(x)$ [W], który nie powinien przekraczać założonego limitu $\varepsilon_{\max}$ [W]. Wymaganie tego typu może wynikać z ograniczeń infrastruktury energetycznej w obszarze pracy gridu, która pozwala na zapewnienie zasilania nieprzekraczającego poziomu $\varepsilon_{\max}$. Pobór mocy elektrycznej przez komputery scharakteryzować można dwoma wektorami: $\varepsilon^{\mathrm{idle}} = [\varepsilon_1^{\mathrm{idle}}, \ldots, \varepsilon_j^{\mathrm{idle}}, \ldots, \varepsilon_J^{\mathrm{idle}}]$ oraz $\varepsilon^{\mathrm{load}} = [\varepsilon_1^{\mathrm{load}}, \ldots, \varepsilon_j^{\mathrm{load}}, \ldots, \varepsilon_J^{\mathrm{load}}]$, przy czym $\varepsilon_j^{\mathrm{idle}}$ [W] wskazuje zapotrzebowanie uruchomionego, ale nie obciążonego komputera j-tego rodzaju, a $\varepsilon_j^{\mathrm{load}}$ [W] opisuje zapotrzebowanie energetyczne obciążonego komputera j-tego rodzaju.

Ograniczenie na łączny pobór mocy elektrycznej gridu zdefiniowano następująco:

$$E(x) \leqslant \varepsilon_{\max}, \tag{4.31}$$

gdzie $E(x) = \sum_{i=1}^{I}\sum_{j=1}^{J} \varepsilon_j^{\text{load}} x_{ij}^{\beta}$ – pobór mocy elektrycznej przez obciążone komputery gridu dla strategii x.

W wypadku modernizacji istniejącego gridu należy uwzględnić ograniczenia na liczbę dostępnych komputerów. W projektowanej konfiguracji interesariusze mogą żądać wycofania niektórych komputerów ze względu na zbyt długi czas ich eksploatacji. Wówczas przyjmuje się, aby $\nu_j = 0$ w wypadku zamiaru wycofania komputerów j-tego rodzaju. Natomiast ograniczenie na liczbę komputerów zadanych typów w gridzie formułuje się, jak niżej:

$$\sum_{i=1}^{I} x_{i,j}^{\beta} \leqslant \nu_j,\ j \in \{\mathcal{J} \setminus \mathcal{J}^*\}, \tag{4.32}$$

gdzie:
$\mathcal{J}$ – zbiór indeksów dostępnych rodzajów komputerów, $\mathcal{J} = \{j \in \mathbb{N} : j = \overline{1,J}\}$,
$\mathcal{J}^*$ – zbiór indeksów komputerów o zagwarantowanej liczbie egzemplarzy w gridzie, $\mathcal{J}^* \subset \mathcal{J}$.

Projektant może rozważać modernizację, dla której gwarantuje się zainstalowanie dokładnie ν_j komputerów j-tego typu. Wówczas obowiązują ograniczenia równościowe, jak niżej:

$$\sum_{i=1}^{I} x_{i,j}^{\beta} = \nu_j,\ j \in \mathcal{J}^*. \tag{4.33}$$

Sformułowane ograniczenia mogą znacząco zredukować zbiór strategii dopuszczalnych, zawężając przestrzeń poszukiwań w zagadnieniach optymalizacji. Zbyt restrykcyjny zestaw ograniczeń może spowodować, że zbiór dopuszczalnych rozwiązań będzie zbiorem pustym. W takiej sytuacji interesariusze powinni rozluźnić wymagania poprzez modyfikację zadanych wartości granicznych.

4.2 Ocena jakości strategii zespołu agentów

W celu sformułowania zagadnień optymalizacji strategii sieci agentów warstwy pośredniczącej gridu konieczny jest wybór kryteriów oceny jakości rozwiązań. Każde ze zdefiniowanych wcześniej ograniczeń może zostać przekształcone w kryterium optymalizacji. Ponadto definiuje się dodatkowe kryteria związane z wykorzystaniem zasobów gridu oraz jego charakterystykami w czasie realizacji zadań według wybranej strategii.

Obciążenie procesorów w i-tym węźle $\hat{Z}_i(x)$ [s] przy strategii agentów x wyraża się, jak niżej:

$$\hat{Z}_i(x) = \sum_{j=1}^{J}\sum_{v=1}^{V} t_{vj} x_{vi}^{\alpha} x_{ij}^{\beta},\ i = \overline{1,I}. \tag{4.34}$$

Komputery w węzłach cechują się różnymi obciążeniami w zależności od tego, które agenty w nich działają. *Obciążenie procesorów newralgicznego węzła* $\hat{Z}_{\max}(x)$ [s] definiuje się, jak niżej:

$$\hat{Z}_{\max}(x) = \max_{i=\overline{1,I}} \hat{Z}_i(x). \tag{4.35}$$

Obciążenie komunikacyjne i-tego węzła $\widetilde{Z}_i(x)$ [s] oraz *obciążenie węzła newralgicznego pod względem komunikacji* $\widetilde{Z}_{\max}(x)$ [s] wyznacza się następująco:

$$\widetilde{Z}_i(x) = \sum_{v=1}^{V} \sum_{\substack{u=1 \\ u \neq v}}^{V} \sum_{\substack{k=1 \\ k \neq i}}^{I} \tau_{vu} x_{vi}^{\alpha} x_{uk}^{\alpha}, \ i = \overline{1,I}, \tag{4.36}$$

$$\widetilde{Z}_{\max}(x) = \max_{i=\overline{1,I}} \widetilde{Z}_i(x). \tag{4.37}$$

W modelu zakłada się, że przepustowość kanałów komunikacyjnych pomiędzy węzłami jest taka sama. W konsekwencji czasy w macierzy τ zależą od komunikującej się pary agentów, ale nie od ich ulokowania w węzłach systemu. Odpowiada to sytuacji w laboratoryjnej instalacji gridu *Comcute*.

W systemach rozproszonych geograficznie czasy komunikacji uzależnione są od liczby serwerów pośrednich i przepustowości łączy pomiędzy nimi. W takim przypadku niewłaściwe migracje agentów do odległych węzłów mogą znacząco pogorszyć wydajność systemu. Istotne jest, aby agenty wymieniające duże zbiory danych przemieszczały się do węzłów połączonych kanałami o większej przepustowości lub nawet do tych samych węzłów w celu minimalizacji opóźnień komunikacyjnych. Warto także zauważyć, że węzeł newralgiczny pod względem obciążenia CPU może się różnić od newralgicznego węzła pod względem obciążenia komunikacyjnego. Dotyczy to także wykorzystania innych zasobów, w tym pamięci: RAM, HDD oraz SSD.

Równoważenie obciążeń w gridzie może odbywać się za pomocą minimalizacji obu funkcji celu $\hat{Z}_{\max}(x)$ i $\widehat{Z}_{\max}(x)$ lub poprzez wprowadzenie ograniczeń zdefiniowanych, jak niżej:

$$\hat{Z}_{\max}(x) \leqslant \hat{Z}_{\text{gr}}, \tag{4.38}$$

$$\widetilde{Z}_{\max}(x) \leqslant \widetilde{Z}_{\text{gr}}, \tag{4.39}$$

gdzie:
$\hat{Z}_{\text{gr}}$ – maksymalne dopuszczalne obciążenie procesorów newralgicznego węzła [s],
$\widetilde{Z}_{\text{gr}}$ – maksymalne dopuszczalne obciążenie komunikacyjne newralgicznego węzła [s].

Obciążenie systemu może być wyrażone za pomocą dwóch kryteriów: $\hat{Z}_{\text{suma}}$ – łączne obciążenie procesorów węzłów gridu [s], a także $\widetilde{Z}_{\text{suma}}$ – łączne obciążenie komunikacyjne [s]. Kryteria te można zdefiniować, jak niżej:

$$\hat{Z}_{\text{suma}}(x) = \sum_{i=1}^{I} \hat{Z}_i(x), \tag{4.40}$$

$$\widetilde{Z}_{\text{suma}}(x) = \sum_{i=1}^{I} \widetilde{Z}_i(x). \tag{4.41}$$

Na łączne obciążenia nałożyć można ograniczenia w następujący sposób:

$$\hat{Z}_{\text{suma}}(x) \leqslant \hat{Z}_{\text{Sgr}}, \tag{4.42}$$

$$\widetilde{Z}_{\text{suma}}(x) \leqslant \widetilde{Z}_{\text{Sgr}}, \tag{4.43}$$

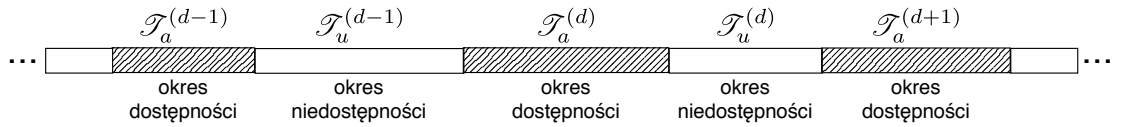
gdzie:
$\hat{Z}_{\text{Sgr}}$ – maksymalne łączne obciążenie procesorów w gridzie [s],
$\widetilde{Z}_{\text{Sgr}}$ – maksymalne łączne obciążenie komunikacyjne w gridzie [s].

Istotnym kryterium oceny jakości strategii agentów zarządzających jest *czas realizacji zadań* $\widehat{T}(x)$ [s]. Zleceniodawcy obliczeń na jego podstawie decydują, czy powierzyć swoje zadania do realizacji w systemie. Dla zastosowań gridu w sytuacjach kryzysowych ważne jest wyznaczenie wyników w oczekiwanym limicie czasu. W przeciwnym razie otrzymane wyniki mogą okazać się bezużyteczne. Prowadzi to do następującego ograniczenia:

$$\widehat{T}(x) \leqslant \widehat{T}_{\text{gr}}, \tag{4.44}$$

gdzie $\widehat{T}_{\text{gr}}$ – maksymalny dopuszczalny czas realizacji zadań użytkowników [s].

Możliwość spełnienia ograniczenia (4.44) zależy w dużym stopniu od mocy obliczeniowej oferowanej przez węzły obliczeniowe. Moc obliczeniowa jest determinowana przez liczbę wolontariuszy udostępniających zasoby komputerowe, wydajności ich komputerów oraz długości przedziałów czasu, w których dostępne są ich zasoby. W modelu zakłada się, że jeśli wolontariusz udostępnia moc obliczeniową komputera, to przetwarzana jest pewna liczba paczek danych w czasie określanym jako *okres dostępności* o długości $\mathscr{T}_a^{(d)}$. Następnie wolontariusz rezygnuje z udostępniania zasobów komputera na czas nazywany *okresem niedostępności* o długości $\mathscr{T}_u^{(d)}$. Po tym okresie wolontariusz ponownie podłącza się do systemu, oferując moc obliczeniową na czas $\mathscr{T}_a^{(d+1)}$ w celu przetworzenia kolejnej partii danych (rys. 4.1). Różnice w wydajności komputerów wolontariuszy mogą zostać uwzględnione za pomocą przeskalowania czasów dostępności według wyników wybranego benchmarku.



Rysunek 4.1: Okresy dostępności i niedostępności komputera wolontariusza [136]

Niech funkcja gęstości prawdopodobieństwa realizacji w czasie $\widehat{t}$ przez grupę $\widehat{I}$ jednostek obliczeniowych zadania wymagającego łącznie ζ jednostek czasu pracy, zdefiniowana jest następująco [136]:

$$f\left(\widehat{t}\right) = \frac{\zeta}{\sqrt{2\pi\sigma_b^2\,\widehat{t}^{\,3}}} \exp\left[-\frac{\left(\zeta - \overline{b}\,\widehat{t}\right)^2}{2\sigma_b^2\widehat{t}}\right], \tag{4.45}$$

przy czym:

$$\overline{b} = \frac{\mathscr{T}_a}{\mathscr{T}_a + \mathscr{T}_u}\ \widehat{I}, \tag{4.46}$$

$$\sigma_b^2 = \frac{\sigma_a^2 \mathscr{T}_u^2 + \sigma_u^2 \mathscr{T}_a^2}{(\mathscr{T}_a + \mathscr{T}_u)^3}\ \widehat{I}, \tag{4.47}$$

gdzie:
$\mathscr{T}_a$ – średni czas dostępności wolontariuszy [s],
σ_a^2 – wariancja czasu dostępności wolontariuszy [s],
$\mathscr{T}_u$ – średni czas niedostępności wolontariuszy [s],
σ_u^2 – wariancja czasu niedostępności wolontariuszy [s].

Wartość oczekiwana czasu realizacji zadania o ζ jednostkach czasu pracy, określona jest zależnością, jak niżej [136]:

$$\overline{f} = \frac{\zeta}{\overline{b}} = \zeta\ \frac{\mathscr{T}_a + \mathscr{T}_u}{\mathscr{T}_a}\ \widehat{I}^{-1}. \tag{4.48}$$

Aby zastosować zależność (4.48), konieczne jest spełnienie założeń przyjętych przez *Kleinrocka* w [136]. Po pierwsze, rozważane zadanie powinno cechować się możliwością podziału wymaganego czasu przetwarzania o długości ζ jednostek na mniejsze fragmenty, które mogą być obsługiwane niezależnie przez różne jednostki obliczeniowe. Jest to prawdziwe dla grida *Comcute*, w którym ζ odpowiada *wymaganemu sumarycznemu czasowi pracy komputerów wolontariuszy*. Jest on sumą czasów przetwarzania wszystkich paczek danych dla zadań obliczeniowych $z_1, \ldots, z_m, \ldots, z_M$, jak niżej:

$$\zeta = \sum_{m=1}^{M} \sum_{l=1}^{L(z_m)} P_T(z_m) = \sum_{m=1}^{M} L(z_m) P_T(z_m). \tag{4.49}$$

Zakłada się, że $P_T(z_m) \ll \zeta$ dla $m = \overline{1,M}$, co oznacza, że każde zadanie z_m dzielone jest na operacje obsługi paczek danych $IN_m^{(1)}, \ldots, IN_m^{(l)}, \ldots, IN_m^{(L(z_m))}$ o krótkim czasie przetwarzania. Ponadto paczki danych mogą być przetwarzane niezależnie. Przy założeniu, że $P_T(z_m) \ll \mathscr{T}_a$ straty mocy obliczeniowej wolontariuszy, wynikające z pobrania paczki danych i odłączenia komputera od gridu przed zwróceniem wyników, są pomijalne.

Zależność (4.48) odzwierciedla działanie systemu rozproszonego dla zadań przetwarzających dane cechujące się dużymi wielkościami, co można wyrazić w postaci ograniczeń: $L(z_m)P_T(z_m) \gg \mathscr{T}_a + \mathscr{T}_u$ oraz $L(z_m) \gg \widehat{I}$, dla $m = \overline{1,M}$. Oba te wymagania są spełnione w gridzie *Comcute*. Ponadto zakłada się, że nie występują czasy oczekiwania w kolejkach systemowych, które świadczą o przeciążeniu węzłów obliczeniowych i zagłodzeniu procesów uruchomionych zadań. Odpowiada to warunkom wykonania zadań w gridzie *Comcute*.

W oparciu o zależności (4.48) i (4.49) można wyznaczyć *wartość oczekiwaną rzeczywistego czasu przetwarzania (ang. wall time) w węzłach obliczeniowych* $\bar{t}$ [s] dla zbioru zadań $z_1, \ldots, z_m, \ldots, z_M$:

$$\bar{t} = \frac{\mathscr{T}_a + \mathscr{T}_u}{\mathscr{T}_a}\, \widehat{I}^{-1} \sum_{m=1}^{M} L(z_m) P_R(z_m) P_T(z_m). \tag{4.50}$$

W modelu wolontariatu obliczeniowego średnie czasy dostępności $\mathscr{T}_a$ i niedostępności $\mathscr{T}_u$ węzłów obliczeniowych mogą zostać wyznaczone w wyniku obserwacji profili zachowań uczestników obliczeń [127]. Z kolei dla obliczeń obligatoryjnych (ang. *obligatory computing*), w których pracownicy administracji zobowiązani są do udostępnienia mocy obliczeniowej w warunkach kryzysowych [29], czasy wynikają z umów z zaangażowanymi jednostkami administracji państwowej.

Na czas realizacji zadań w gridzie *Comcute* składa się rzeczywisty czas przetwarzania w węzłach obliczeniowych oraz czas wynikający z narzutów systemu na dystrybucję i zarządzanie zadaniami. W oparciu o zależność (4.50) wyznaczyć można *oczekiwany czas realizacji zadań* $\overline{T}(x)$ [s]:

$$\overline{T}(x) = \bar{t} + \mathcal{O}(x), \tag{4.51}$$

gdzie $\mathcal{O}(x)$ – narzut (ang. *overhead*) w gridzie *Comcute* na czas wykonania zadań przy strategii zespołu agentów zarządzających x [s].

Narzut oszacować można od góry w oparciu o wyznaczone obciążenia sumaryczne w systemie, jak niżej:

$$\mathcal{O}(x) \leqslant \hat{Z}_{\text{suma}}(x) + \widetilde{Z}_{\text{suma}}(x). \tag{4.52}$$

Na podstawie wyników eksperymentów przeprowadzonych w gridzie *Comcute* stwierdzono, że dla tego systemu prawdziwe jest również dokładniejsze oszacowanie górne, jak niżej:

$$\mathcal{O}(x) \leqslant \hat{Z}_{\max}(x) + \widetilde{Z}_{\max}(x). \tag{4.53}$$

Ograniczenie na czas realizacji zadań sformułować można w kontekście czasu oczekiwanego, co pozwala na uwzględnienie mocy oferowanej przez wolontariuszy, jak następuje:

$$\overline{T}(x) \leqslant \widehat{T}_{\text{gr}}. \tag{4.54}$$

Kolejne kryterium oceny jakości strategii zespołu agentów warstwy pośredniczącej gridu odnosi się do kosztów pracy systemu. Składają się na nie koszty pracy komputerów w węzłach gridu oraz koszty transmisji danych pomiędzy nimi. W przypadku serwerów dzierżawionych w chmurze, koszt wynika z konfiguracji wybranego serwera i z czasu jego działania niezależnie od obciążeń. Opłata wynika zatem z *zarezerwowanej mocy obliczeniowej* – np. z liczby rdzeni i prędkości taktowania wybranego procesora – a nie z faktycznego wykorzystania. Najczęściej stosowane są taryfy, w których opłata pobierana jest za każdą godzinę pracy z góry [12, 67]. Niektórzy dostawcy oferują usługi

z jednominutowym okresem taryfikacji [109, 192]. Koszt komunikacji dla węzłów w chmurze może wynikać z przepustowości łącza wirtualnego pomiędzy nimi lub rozmiaru przesyłanych danych.

W przypadku instancji laboratoryjnej systemu *Comcute* bezpośredni koszt pracy węzłów zależy od zużycia energii elektrycznej. Składa się na nie zużycie energii przez węzły pod obciążeniem w okresach, gdy agenty obsługują zadania oraz zużycie energii w okresach bezczynności, np. gdy agenty typu S w węźle oczekują na propagację danych od agentów typu W działających w innych węzłach. Koszt komunikacji determinowany jest przez zużycie energii przez urządzenia sieciowe. W ogólnym przypadku koszt realizacji zadań opisuje zależność, jak niżej:

$$\begin{aligned} K(x) = &\sum_{i=1}^{I}\sum_{j=1}^{J} x_{ij}^{\beta}\left[\hat{Z}_i(x)\,\hat{c}_{i,j}^{\text{load}} + \left(\overline{T}(x) - \hat{Z}_i(x)\right)\hat{c}_{i,j}^{\text{idle}}\right] + \\ &\sum_{v=1}^{V}\sum_{\substack{u=1\\u\neq v}}^{V}\sum_{i=1}^{I}\sum_{\substack{k=1\\k\neq i}}^{I} x_{vi}^{\alpha}x_{uk}^{\alpha}\left[\tau_{vu}\tilde{c}_{i,k}^{\text{load}} + \left(\overline{T}(x) - \tau_{vu}\right)\tilde{c}_{i,k}^{\text{idle}}\right], \end{aligned} \tag{4.55}$$

gdzie:
$\hat{c}_{i,j}^{\text{load}}$ – koszt wykorzystania komputera j-tego typu w i-tym węźle w warunkach obciążenia [JM/s],
$\hat{c}_{i,j}^{\text{idle}}$ – koszt utrzymania komputera j-tego typu w i-tym węźle w stanie gotowości, gdy nie jest obciążony [JM/s],
$\tilde{c}_{i,k}^{\text{load}}$ – koszt transmisji danych między węzłami ω_i oraz ω_k [JM/s],
$\tilde{c}_{i,k}^{\text{idle}}$ – koszt utrzymania w gotowości kanału komunikacyjnego pomiędzy węzłami ω_i oraz ω_k [JM/s].

W przypadku płatności za czas dzierżawy serwera zachodzi $\hat{c}_{i,j}^{\text{load}} = \hat{c}_{i,j}^{\text{idle}}$. Dla infrastruktury osadzonej w chmurze zazwyczaj obowiązuje $\tilde{c}_{i,k}^{\text{idle}} = 0$, gdyż opłaty pobierane są jedynie za wykorzystanie łączy komunikacyjnych. Z kolei dla laboratoryjnej instancji gridu *Comcute* obowiązuje $\hat{c}_{i,j}^{\text{load}} \neq \hat{c}_{i,j}^{\text{idle}}$ oraz w szczególności: $\hat{c}_{i,j}^{\text{load}} = c_{\varepsilon}\varepsilon_j^{\text{load}}$ i $\hat{c}_{i,j}^{\text{idle}} = c_{\varepsilon}\varepsilon_j^{\text{idle}}$ dla $i = \overline{1,I}$, $j = \overline{1,J}$, przy czym c_{ε} oznacza koszt zużycia jednostki energii elektrycznej [JM/Wh]. Ponadto zakłada się, że $\tilde{c}_{i,j}^{\text{load}} = \tilde{c}_{i,j}^{\text{idle}}$, a koszt komunikacji zależy od zużycia energii elektrycznej przez urządzenia sieciowe $\widetilde{\varepsilon}$ [W]. Infrastruktura sieciowa jest wspólna dla wszystkich węzłów gridu. W rezultacie dla instancji laboratoryjnej wyrażenie na koszt realizacji zadań można przekształcić następująco:

$$K(x) = \frac{c_{\varepsilon}}{3600}\left(\sum_{i=1}^{I}\sum_{j=1}^{J} x_{ij}^{\beta}\left[\hat{Z}_i(x)\,\varepsilon_j^{\text{load}} + \left(\overline{T}(x) - \hat{Z}_i(x)\right)\varepsilon_j^{\text{idle}}\right] + \widetilde{\varepsilon}\,\overline{T}(x)\right). \tag{4.56}$$

Dostawca energii elektrycznej może oferować taryfy, w których koszt zmienia się w zależności od pory dnia, np. niższe stawki obowiązują w porach nocnych. W takiej sytuacji parametr c_{ε} podlega zmianom według taryfikatora dostawcy. Koszt wykorzystania komputera j-tego typu może różnić się w zależności od węzła, w którym zostanie on zainstalowany. Wynika to z różnych kosztów energii elektrycznej w poszczególnych lokalizacjach lub innych taryf dzierżawy u konkurujących ze sobą dostawców.

Koszt realizacji zadań powinien mieścić się w założonym limicie finansowym, co można zapisać, jak niżej:

$$K(x) \leqslant K_{\mathrm{gr}}, \tag{4.57}$$

gdzie K_{gr} – maksymalny dopuszczalny koszt realizacji zadań [JM].

Istotne kryterium odnosi się do dostępności gridu. Zakłada się, że losowe zdarzenia prowadzące do niedostępności komputera j-tego rodzaju występują z intensywnością γ_j^{-1} [s] według rozkładu wykładniczego. W takiej sytuacji dostępność poszczególnych węzłów maleje wykładniczo w czasie, a dostępność Γ całego systemu działającego według strategii x wyznaczana jest, jak niżej [276]:

$$\Gamma(x) = \prod_{i=1}^{I} \prod_{j=1}^{J} \exp\left(-\gamma_j x_{i,j}^{\beta} \overline{T}(x)\right), \tag{4.58}$$

gdzie $\gamma = [\gamma_1, \ldots, \gamma_j, \ldots \gamma_J]$ – wektor współczynników dla rozkładów wykładniczych dostępności komputerów poszczególnych typów.

Przy założeniu, że parametry γ_j nie zmieniają się podczas działania systemu, dostępność gridu uzależniona jest od elementów zawartych w macierzy czasów obsługi zadań T. Niższe wartości współczynników γ i krótsze czasy przetwarzania danych skutkują wyższymi wartościami dostępności systemu [276].

Istotne kryteria oceny jakości strategii agentów odnoszą się do wielkości rezerw pamięci operacyjnej i pamięci trwałej. Obecność rezerw umożliwia migrację agentów w systemie w reakcji na awarię węzłów. Ponadto rezerwy zasobów mogą ograniczyć konieczność rekonfiguracji systemu, np. w celu obsłużenia kolejnych zadań. *Łączną rezerwę pamięci RAM* $M^{\mathrm{RAM}}(x)$ [GB] wyznacza się, jak niżej:

$$M^{\mathrm{RAM}}(x) = \sum_{i=1}^{I} \kappa_i^{\mathrm{RAM}}(x), \text{ dla } x \in \boldsymbol{\mathcal{X}}. \tag{4.59}$$

Łączną rezerwę pamięci HDD $M^{\mathrm{HDD}}(x)$ [TB] opisuje poniższa zależność:

$$M^{\mathrm{HDD}}(x) = \sum_{i=1}^{I} \kappa_i^{\mathrm{HDD}}(x), \text{ dla } x \in \boldsymbol{\mathcal{X}}. \tag{4.60}$$

Z kolei *łączna rezerwa pamięci SSD* $M^{\mathrm{SSD}}(x)$ [TB] definiowana jest następująco:

$$M^{\mathrm{SSD}}(x) = \sum_{i=1}^{I} \kappa_i^{\mathrm{SSD}}(x), \text{ dla } x \in \boldsymbol{\mathcal{X}}. \tag{4.61}$$

Ograniczenia odnoszące się do łącznych rezerw pamięci mają następującą postać:

$$M^{\mathrm{RAM}}(x) \geqslant M_{\mathrm{gr}}^{\mathrm{RAM}}, \tag{4.62}$$

$$M^{\mathrm{HDD}}(x) \geqslant M_{\mathrm{gr}}^{\mathrm{HDD}}, \tag{4.63}$$

$$M^{\mathrm{SSD}}(x) \geqslant M_{\mathrm{gr}}^{\mathrm{SSD}}, \tag{4.64}$$

gdzie:
$M_{\mathrm{gr}}^{\mathrm{RAM}}$ – minimalna wymagana wielkość rezerwy pamięci RAM w gridzie [GB],
$M_{\mathrm{gr}}^{\mathrm{HDD}}$ – minimalna wymagana wielkość rezerwy pamięci HDD w gridzie [TB],
$M_{\mathrm{gr}}^{\mathrm{SSD}}$ – minimalna wymagana wielkość rezerwy pamięci SSD w gridzie [TB].

Kryteria oceny jakości mogą posłużyć do sformułowania zagadnień optymalizacji strategii zespołu agentów warstwy pośredniczącej gridu. Równocześnie, poprzez nałożenie na nie ograniczeń nierównościowych, mogą one modelować wymagania ilościowe. Pozwala to na zdefiniowanie wielu różnych zagadnień optymalizacji w zależności od oczekiwań interesariusza. Poza ograniczeniami wybieranymi przez interesariuszy, każde zagadnienie optymalizacji powinno uwzględniać wymagania formalne odnoszące się do: przydziału zadań użytkowników do agentów (4.4), rozmieszczenia agentów w węzłach (4.6) oraz wyboru typów komputerów do instalacji w węzłach (4.8).

4.3 Agenty-solwery do jednokryterialnej optymalizacji strategii zespołu agentów warstwy pośredniczącej

Niech za wyznaczanie rozwiązań dla zagadnień optymalizacji formułowanych przez interesariuszy odpowiedzialne są *agenty-solwery*. W przypadku optymalizacji jednokryterialnej *agent-solwer* wykorzystuje kryterium oceny jakości strategii zespołu agentów, zestaw ograniczeń oraz wymagane dane wejściowe. Zbiór istotnych danych wejściowych jest uzależniony od zadanego kryterium oraz ograniczeń. Przykładowo, wprowadzenie ograniczenia na koszt działania systemu $K(x)$ wymusza uwzględnienie wektorów zużycia energii elektrycznej przez komputery: $\varepsilon^{\mathrm{idle}} = [\varepsilon_1^{\mathrm{idle}}, \ldots, \varepsilon_j^{\mathrm{idle}}, \ldots, \varepsilon_J^{\mathrm{idle}}]$ oraz $\varepsilon^{\mathrm{load}} = [\varepsilon_1^{\mathrm{load}}, \ldots, \varepsilon_j^{\mathrm{load}}, \ldots, \varepsilon_J^{\mathrm{load}}]$. Z kolei w przypadku ograniczenia na koszt zakupu $\Xi(x)$ zadany powinien być wektor cen komputerów różnych typów: $\xi = [\xi_1, \ldots, \xi_j, \ldots, \xi_J]$.

W gridzie *Comcute* interesariusz specyfikuje swoje wymagania odnośnie pożądanych cech systemu za pośrednictwem internetowego interfejsu *Aplikacji Administratora Gridu* AAG'16. W tabeli nr 4.1 zaprezentowano indeksy selektorów μ_k dla poszczególnych ograniczeń. Interesariusz może wybrać dowolny podzbiór preferowanych opcji. Ponadto interesariusz wprowadza wartości graniczne dla wybranych wymagań.

W przypadku wyboru k-tego ograniczenia interesariusz określa wartość selektora $\mu_k \geqslant 1$. Selektor μ_k jest także wagą funkcji kary, jeśli rozwiązanie nie spełnia k-tego wymagania. Wartości selektorów definiowane są zatem następująco:

$$\mu_k = \begin{cases} 0, & \text{gdy } k\text{-te wymaganie nie zostało wybrane,} \\ \geqslant 1, & \text{gdy } k\text{-te wymaganie zostało wybrane,} \end{cases} \quad k = \overline{1,19}. \tag{4.65}$$

Tabela 4.1: Indeksy selektorów ograniczeń dla strategii zespołu agentów warstwy pośredniczącej gridu

k	Wymaganie projektowe	Ograniczenie
1	Ograniczenie obciążenia procesorów newralgicznego komputera	$\hat{Z}_{\max}(x) \leqslant \hat{Z}_{\mathrm{gr}}$
2	Ograniczenie obciążenia komunikacyjnego newralgicznego komputera	$\widetilde{Z}_{\max}(x) \leqslant \widetilde{Z}_{\mathrm{gr}}$
3	Ograniczenie na łączne obciążenie procesorów w gridzie	$\hat{Z}_{\mathrm{suma}}(x) \leqslant \hat{Z}_{\mathrm{Sgr}}$
4	Ograniczenie na łączne obciążenie komunikacyjne w gridzie	$\widetilde{Z}_{\mathrm{suma}}(x) \leqslant \widetilde{Z}_{\mathrm{Sgr}}$
5	Ograniczenie wydajnościowe	$\Theta(x) \geqslant \vartheta_{\min}$
6	Ograniczenie kosztów zakupu komputerów	$\Xi(x) \leqslant \xi_{\max}$
7	Nieprzekroczenie kosztów przetwarzania danych i komunikacji w gridzie	$K(x) \leqslant K_{\mathrm{gr}}$
8	Zapewnienie minimalnego stopnia rozproszenia agentów	$\eta(x) \geqslant \eta_{\min}$
9	Nieprzekroczenia wielkości dostępnej pamięci RAM w newralgicznym węźle	$\kappa_{\min}^{\mathrm{RAM}}(x) \geqslant 0$
10	Nieprzekroczenia wielkości dostępnej pamięci HDD w newralgicznym węźle	$\kappa_{\min}^{\mathrm{HDD}}(x) \geqslant 0$
11	Nieprzekroczenia wielkości dostępnej pamięci SSD w newralgicznym węźle	$\kappa_{\min}^{\mathrm{SSD}}(x) \geqslant 0$
12	Zapewnienie łącznej rezerwy pamięci RAM	$M^{\mathrm{RAM}}(x) \geqslant M_{\mathrm{gr}}^{\mathrm{RAM}}$
13	Zapewnienie łącznej rezerwy pamięci HDD	$M^{\mathrm{HDD}}(x) \geqslant M_{\mathrm{gr}}^{\mathrm{HDD}}$
14	Zapewnienie łącznej rezerwy pamięci SSD	$M^{\mathrm{SSD}}(x) \geqslant M_{\mathrm{gr}}^{\mathrm{SSD}}$
15	Ograniczenie energetyczne	$E(x) \leqslant \varepsilon_{\max}$
16	Ograniczenie na dostępność gridu	$\Gamma(x) \geqslant \Gamma_{\mathrm{gr}}$
17	Ograniczenie na oczekiwany czas wykonania zadań	$\overline{T}(x) \leqslant \widehat{T}_{\mathrm{gr}}$
18	Ograniczenia na liczbę komputerów zadanych rodzajów	$\sum_{i=1}^{I} x_{i,j}^{\beta} \leqslant \nu_j, j \in \{\mathcal{J} \setminus \mathcal{J}^*\}$
19	Zagwarantowanie liczby komputerów zadanych rodzajów	$\sum_{i=1}^{I} x_{i,j}^{\beta} = \nu_j, j \in \mathcal{J}^*$

Źródło: opracowanie własne

Podobnie jak w przypadku ograniczeń, za pośrednictwem interfejsu aplikacji AAG'16 interesariusze mogą wybrać kryterium optymalizacji gridu. W tabeli nr 4.2 przedstawiono numery kryteriów cząstkowych F_n do oceny jakości strategii agentów.

Interesariusz wybiera jedno kryterium cząstkowe oraz zestaw ograniczeń. Następnie *agent-solwer* pobiera odpowiednie dane i rozwiązuje problem optymalizacji w zadanym przez interesariusza czasie, np. 3 minuty. Limit czasowy na wyznaczenie rozwiązania jest szczególnie istotny w systemach cechujących się dużą dynamiką. Uzyskana strategia może posłużyć do rekonfiguracji zasobów systemu.

Tabela 4.2: Kryteria oceny jakości strategii zespołu agentów warstwy pośredniczącej gridu

n	Kryterium cząstkowe F_n	Oznaczenie	Optymalizacja
1	Obciążenie procesorów newralgicznego komputera	$\hat{Z}_{\max}(x)$	min
2	Obciążenie komunikacyjne newralgicznego komputera	$\widetilde{Z}_{\max}(x)$	min
3	Łączne obciążenie procesorów w gridzie	$\hat{Z}_{\text{suma}}(x)$	min
4	Łączne obciążenie komunikacyjne w gridzie	$\widetilde{Z}_{\text{suma}}(x)$	min
5	Łączna wydajność gridu	$\Theta(x)$	max
6	Koszt zakupu hostów	$\Xi(x)$	min
7	Koszt przetwarzania danych i komunikacji	$K(x)$	min
8	Stopień rozproszenia agentów	$\eta(x)$	max
9	Wielkość dostępnej pamięci RAM w newralgicznym hoście	$\kappa_{\min}^{\text{RAM}}(x)$	max
10	Wielkość dostępnej pamięci HDD w newralgicznym hoście	$\kappa_{\min}^{\text{HDD}}(x)$	max
11	Wielkość dostępnej pamięci SSD w newralgicznym hoście	$\kappa_{\min}^{\text{SSD}}(x)$	max
12	Wielkość łącznej rezerwy pamięci RAM	$M^{\text{RAM}}(x)$	max
13	Wielkość łącznej rezerwy pamięci HDD	$M^{\text{HDD}}(x)$	max
14	Wielkość łącznej rezerwy pamięci SSD	$M^{\text{SSD}}(x)$	max
15	Szczytowe zapotrzebowanie energetyczne	$E(x)$	min
16	Dostępność gridu	$\Gamma(x)$	max
17	Oczekiwany czas realizacji zadań	$\overline{T}(x)$	min

Źródło: opracowanie własne

Zagadnienie optymalizacji strategii zespołu agentów $\boldsymbol{\mathcal{Z}}(F_n, \mu)$ ze względu na kryterium cząstkowe F_n przy uwzględnieniu zestawu opcjonalnych ograniczeń $\mu = [\mu_1, \ldots, \mu_{19}]$ sformułowano następująco:

Dla danych wejściowych: $[\mu_1, \ldots, \mu_{19}]$, $\{z_1, \ldots, z_M\}$, V_W, V, I, J, M, w, s, T, τ, $\widehat{Z}_{\text{gr}}$, $\widetilde{Z}_{\text{gr}}$, $\widehat{Z}_{S_{\text{gr}}}$, $\widetilde{Z}_{S_{\text{gr}}}$, $\vartheta, \vartheta_{\min}$, $\xi, \xi_{\max}$, c_ε, $\hat{c}_{i,j}^{\text{load}}$, $\hat{c}_{i,j}^{\text{idle}}$, $\tilde{c}_{i,k}^{\text{load}}$, $\tilde{c}_{i,k}^{\text{idle}}$, K_{gr}, $\eta_{\min}$, r, ram, h, hdd, $\hat{h}$, ssd, $M_{\text{gr}}^{\text{RAM}}$, $M_{\text{gr}}^{\text{HDD}}$, $M_{\text{gr}}^{\text{SSD}}$, $\widetilde{\varepsilon}$, $\varepsilon^{\text{idle}}$, $\varepsilon^{\text{load}}$, $\varepsilon_{\max}$, Γ_{gr}, ν, $\mathcal{J}^*$, $\mathscr{T}_a$, $\mathscr{T}_u$, $\widehat{I}$, $\widehat{T}_{\text{gr}}$, należy wyznaczyć x^*, takie że:

$$F_n(x^*) = \min_{x \in \widehat{\boldsymbol{\mathcal{X}}}} F_n(x), \tag{4.66}$$

gdzie:

$$\widehat{\boldsymbol{\mathcal{X}}} = \{x \in \mathbb{B}^{MV_W + I(V+J)} : \sum_{v=1}^{V_W} x_{mv}^z = 1 \text{ dla } m = \overline{1, M};\ \sum_{i=1}^{I} x_{vi}^\alpha = 1 \text{ dla } v = \overline{1, V};$$

$$\sum_{j=1}^{J} x_{ij}^\beta = 1 \text{ dla } i = \overline{1, I};\ \hat{Z}_{\max}(x) \leqslant \hat{Z}_{\text{gr}} \text{ dla } \mu_1 \neq 0;\ \widetilde{Z}_{\max}(x) \leqslant \widetilde{Z}_{\text{gr}} \text{ dla } \mu_2 \neq 0;$$

$$\hat{Z}_{\text{suma}}(x) \leqslant \hat{Z}_{\text{Sgr}} \text{ dla } \mu_3 \neq 0;\ \widetilde{Z}_{\text{suma}}(x) \leqslant \widetilde{Z}_{\text{Sgr}} \text{ dla } \mu_4 \neq 0;\ \Theta(x) \geqslant \vartheta_{\min} \text{ dla } \mu_5 \neq 0;$$
$$\Xi(x) \leqslant \xi_{\max} \text{ dla } \mu_6 \neq 0;\ K(x) \leqslant K_{\text{gr}} \text{ dla } \mu_7 \neq 0;\ \eta(x) \geqslant \eta_{\min} \text{ dla } \mu_8 \neq 0;$$
$$\kappa_{\min}^{\text{RAM}}(x) \geqslant 0 \text{ dla } \mu_9 \neq 0;\ \kappa_{\min}^{\text{HDD}}(x) \geqslant 0 \text{ dla } \mu_{10} \neq 0;\ \kappa_{\min}^{\text{SSD}}(x) \geqslant 0 \text{ dla } \mu_{11} \neq 0;$$
$$M^{\text{RAM}}(x) \geqslant M_{\text{gr}}^{\text{RAM}} \text{ dla } \mu_{12} \neq 0;\ M^{\text{HDD}}(x) \geqslant M_{\text{gr}}^{\text{HDD}} \text{ dla } \mu_{13} \neq 0;$$
$$M^{\text{SSD}}(x) \geqslant M_{\text{gr}}^{\text{SSD}} \text{ dla } \mu_{14} \neq 0;\ E(x) \leqslant \varepsilon_{\max} \text{ dla } \mu_{15} \neq 0;\ \Gamma(x) \leqslant \Gamma_{\text{gr}} \text{ dla } \mu_{16} \neq 0;$$
$$\overline{T}(x) \leqslant \widehat{T}_{\text{gr}} \text{ dla } \mu_{17} \neq 0;$$
$$\sum_{i=1}^{I} x_{i,j}^{\beta} \leqslant \nu_j,\ j \in \{\mathcal{J} \setminus \mathcal{J}^*\} \text{ dla } \mu_{18} \neq 0;\ \sum_{i=1}^{I} x_{i,j}^{\beta} = \nu_j,\ j \in \mathcal{J}^* \text{ dla } \mu_{19} \neq 0\},$$

przy czym:

$$\hat{Z}_i(x) = \sum_{j=1}^{J}\sum_{v=1}^{V} t_{vj} x_{vi}^{\alpha} x_{ij}^{\beta},\ i = \overline{1,I};\ \ \hat{Z}_{\max}(x) = \max_{i=\overline{1,I}} \hat{Z}_i(x);\ \ \hat{Z}_{\text{suma}}(x) = \sum_{i=1}^{I} \hat{Z}_i(x);$$

$$\widetilde{Z}_i(x) = \sum_{v=1}^{V}\sum_{\substack{u=1\\u\neq v}}^{V}\sum_{\substack{k=1\\k\neq i}}^{I} \tau_{vu} x_{vi}^{\alpha} x_{uk}^{\alpha},\ i = \overline{1,I};\ \ \widetilde{Z}_{\max}(x) = \max_{i=\overline{1,I}} \widetilde{Z}_i(x);\ \ \widetilde{Z}_{\text{suma}}(x) = \sum_{i=1}^{I} \widetilde{Z}_i(x);$$

$$\Theta(x) = \sum_{i=1}^{I}\sum_{j=1}^{J} \vartheta_j x_{ij}^{\beta};\ \ \Xi(x) = \sum_{i=1}^{I}\sum_{j=1}^{J} \xi_j x_{ij}^{\beta};\ \ \eta(x) = \sum_{v=1}^{V}\sum_{\substack{u=1\\u>v}}^{V}\sum_{i=1}^{I}\sum_{\substack{k=1\\k\neq i}}^{I} x_{vi}^{\alpha} x_{uk}^{\alpha};$$

$$K(x) = \sum_{i=1}^{I}\sum_{j=1}^{J} x_{ij}^{\beta}\left[\hat{Z}_i(x)\,\hat{c}_{i,j}^{\text{load}} + \left(\overline{T}(x) - \hat{Z}_i(x)\right)\hat{c}_{i,j}^{\text{idle}}\right] +$$
$$\sum_{v=1}^{V}\sum_{\substack{u=1\\u\neq v}}^{V}\sum_{i=1}^{I}\sum_{\substack{k=1\\k\neq i}}^{I} x_{vi}^{\alpha} x_{uk}^{\alpha}\left[\tau_{vu}\tilde{c}_{i,k}^{\text{load}} + \left(\overline{T}(x) - \tau_{vu}\right)\tilde{c}_{i,k}^{\text{idle}}\right];$$

$$\kappa_i^{\text{RAM}}(x) = \sum_{j=1}^{J}\left(ram_j - r_j^{\min}\right)x_{ij}^{\beta} - \sum_{v=1}^{V} r_v x_{vi}^{\alpha},\ i = \overline{1,I};\ \ \kappa_{\min}^{\text{RAM}}(x) = \min_{i=\overline{1,I}}\left\{\kappa_i^{\text{RAM}}(x)\right\};$$

$$\kappa_i^{\text{HDD}}(x) = \sum_{j=1}^{J}\left(hdd_j - h_j^{\min}\right)x_{ij}^{\beta} - \sum_{v=1}^{V} h_v x_{vi}^{\alpha},\ i = \overline{1,I};\ \ \kappa_{\min}^{\text{HDD}}(x) = \min_{i=\overline{1,I}}\left\{\kappa_i^{\text{HDD}}(x)\right\};$$

$$\kappa_i^{\text{SSD}}(x) = \sum_{j=1}^{J}\left(ssd_j - \hat{h}_j^{\min}\right)x_{ij}^{\beta} - \sum_{v=1}^{V} \hat{h}_v x_{vi}^{\alpha},\ i = \overline{1,I};\ \ \kappa_{\min}^{\text{SSD}}(x) = \min_{i=\overline{1,I}}\left\{\kappa_i^{\text{SSD}}(x)\right\};$$

$$M^{\text{RAM}}(x) = \sum_{i=1}^{I}\kappa_i^{\text{RAM}}(x);\ \ M^{\text{HDD}}(x) = \sum_{i=1}^{I}\kappa_i^{\text{HDD}}(x);\ \ M^{\text{SSD}}(x) = \sum_{i=1}^{I}\kappa_i^{\text{SSD}}(x);$$

$$E(x) = \sum_{i=1}^{I}\sum_{j=1}^{J}\varepsilon_j^{\text{load}} x_{ij}^{\beta};\ \ \Gamma(x) = \prod_{i=1}^{I}\prod_{j=1}^{J}\exp\left(-\gamma_j x_{ij}^{\beta}\overline{T}(x)\right);$$

$$\overline{T}(x) = \bar{t} + \mathcal{O}(x);\ \ \bar{t} = \frac{\mathscr{T}_a + \mathscr{T}_u}{\mathscr{T}_a}\,\widehat{I}^{-1}\sum_{m=1}^{M} L(z_m)P_T(z_m);\ \ \mathcal{O}(x) \leqslant \hat{Z}_{\max}(x) + \widetilde{Z}_{\max}(x).$$

Warto podkreślić, że zagadnienie optymalizacji strategii zespołu inteligentnych agentów warstwy pośredniczącej gridu $\boldsymbol{\mathcal{Z}}(F_n, \mu)$ może być formułowane przez wielu interesariuszy. W podstawowym modelu bez negocjacji między interesariuszami przyjmuje się, że każdy z nich proponuje

funkcję celu i zestaw ograniczeń, a następnie w zagadnieniu optymalizacji $\boldsymbol{\mathcal{Z}}(F_n, \mu)$ akceptowana jest dokładnie jedna funkcja celu w oparciu o ustalone reguły.

Deterministyczne reguły polegają na wyborze zadanej funkcji w wypadku uzyskania przez nią największej liczby głosów interesariuszy. Jeśli kilka funkcji celu uzyska największą, ale jednakową liczbę głosów od interesariuszy, to następuje druga tura głosowania na funkcje, które uzyskały wynik *ex aequo*. Jeśli nie przyniesie to rozstrzygnięcia, to losowo wybierana jest jedna z funkcji rozpatrywanych w drugiej turze.

Do deterministycznych metod należy także reguła wyboru większością zwykłą (½ głosów + 1) lub też większością kwalifikowaną (np. ⅔ głosów + 1). W wypadku nierozstrzygnięcia głosowania, w kolejnej turze głosuje się na dwie funkcje, a jeśli nadal nie uda się wybrać funkcji celu, to w trzeciej turze stosuje się wybór losowy.

W regułach selekcji *stricte* losowych interesariusze proponują kryteria, a następnie wyznaczane jest prawdopodobieństwo wyboru każdego z nich jako stosunek liczby głosów oddanych na daną funkcję do liczby głosujących. Wybór odbywa się zgodnie z zasadą ruletki. Nie rozpatruje się funkcji, na które nikt nie głosował. Alternatywną regułą jest turniej, w którym losuje się dwie funkcje zgodnie z regułą ruletki, a następnie wybiera się tę o większej liczbie głosów. W wypadku braku rozstrzygnięcia losuje się jedną z nich zgodnie z rozkładem równomiernym.

W przypadku wyboru ograniczeń należy uwzględnić wymagania wszystkich interesariuszy. W wypadku większej liczby interesariuszy można przyjąć, że konieczne są co najmniej dwa głosy na wybrane ograniczenie, aby było ono uwzględnione w zagadnieniu optymalizacji. Rzadko stosowaną regułą jest zasada jednomyślności interesariuszy w odniesieniu do zadanego wymagania. Ograniczenia mogą być także wybierane zgodnie z regułami losowymi. Przykładowo, reguła ruletki może być stosowana do wyboru ograniczeń z uwzględnieniem częstości będącej stosunkiem liczby głosów oddanych na to ograniczenie do liczby wszystkich głosów.

Interesariusze mogą także negocjować między sobą wybór funkcji celu oraz wybór ograniczeń. Metoda delficka polega w tym wypadku na podaniu anonimowo przez każdego interesariusza propozycji funkcji celu i zestawu ograniczeń wraz z ich uzasadnieniem. Propozycje są zbierane przez mediatora, który analizuje je i przedstawia zbiorcze opracowanie interesariuszom. Rolę mediatora pełni coraz częściej program komputerowy. W opracowaniu zbiorczym prezentowany jest rozkład głosów na poszczególne funkcje celu oraz ograniczenia. Ponadto zamieszczone są uzasadnienia, przy czym, jeśli takie same argumenty podaje kilku interesariuszy, to argument ten przytaczany jest jednokrotnie i charakteryzowany jest większą wagą. Interesariusze po analizie zestawienia od mediatora, ponownie składają swoje propozycje odnośnie funkcji celu i ograniczeń. Jeśli trzy iteracje nie doprowadzą do konsensusu, to stosuje się ustaloną wcześniej regułę selekcji funkcji celu oraz ograniczeń. W rezultacie sformułowane jest zagadnienie optymalizacji $\boldsymbol{\mathcal{Z}}(F_n, \mu)$.

Interesujący jest wariant, w którym dla każdego z interesariuszy uruchamiany jest *agent-solwer*, rozwiązujący jego zagadnienie optymalizacji. W zmodyfikowanej metodzie delfickiej interesariusze uzasadniają swoje propozycje za pomocą ocen wyznaczonych rozwiązań. Proponowane strategie mogą zostać ocenione pod kątem kryteriów innych interesariuszy. Wybrane rozwiązanie może być wykorzystane jako strategia zespołu agentów warstwy pośredniczącej.

4.4 Wielokryterialna optymalizacja strategii zespołu agentów

W praktycznych zastosowaniach konieczne jest uwzględnienie więcej niż jednego kryterium optymalizacji w celu uzyskania strategii cechujących się zamierzonymi własnościami. Interesariusz może sformułować kryterium wektorowe oceny jakości strategii zespołu agentów poprzez dobór kryteriów cząstkowych (tabela nr 4.2). Wybór odbywa się z wykorzystaniem interfejsu aplikacji AAG'16, a zdefiniowane zagadnienie optymalizacji rozwiązywane jest przez *agenty-solwery*.

Problem wielokryterialny może być rozwiązywany poprzez przekształcenie go na zagadnienie jednokryterialne i wyznaczenie rozwiązań uzyskanego w ten sposób problemu pomocniczego. Jedną z metod jest sformułowanie funkcji użyteczności, która stanowi sumę ważoną kryteriów cząstkowych. Dylematem jest w tym przypadku odpowiedni dobór wag, które ukierunkują przeszukiwanie rozwiązań na właściwy obszar przestrzeni kryterialnej. Bez znajomości rozkładu rozwiązań dopuszczalnych w tej przestrzeni i charakteru kompromisów, jakie pomiędzy nimi zachodzą, interesariusz nie jest w stanie dobrać optymalnych wartości wag. Ponadto oceny z kryteriów cząstkowych mogą być wyrażone w różnych jednostkach, np. w sekundach dla czasu przetwarzania w newralgicznym węźle i w GB dla rezerwy pamięci RAM. Różne mogą być też rzędy wielkości ocen cząstkowych, co także utrudnia dobór wag [64].

Interesująca metoda transformacji zadania polioptymalizacji do zagadnienia jednokryterialnego polega na wyborze kryterium cząstkowego, które będzie optymalizowane i zamianie pozostałych kryteriów cząstkowych na ograniczenia. Podejście to – podobnie jak w przypadku sumy ważonej kryteriów – wymaga pewnej wiedzy *a priori* na temat preferowanych rozwiązań w celu dobrania wartości ograniczeń. Zbyt restrykcyjne wymagania mogą sprawić, że zbiór rozwiązań dopuszczalnych będzie zbiorem pustym, z kolei zbyt luźne – rozszerzą go poza front *Pareto* w przestrzeni wielokryterialnej.

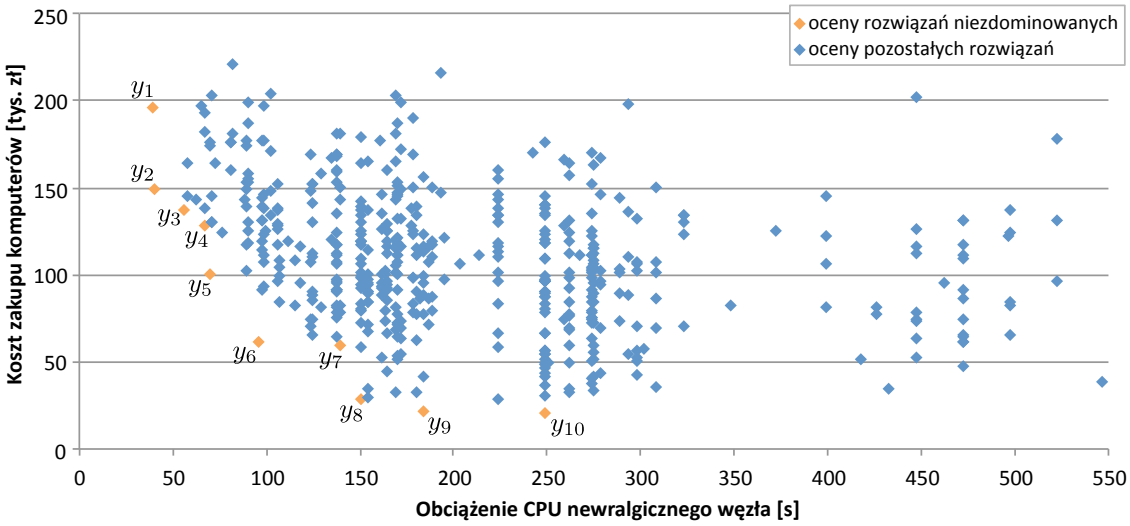
Metoda leksykograficzna opiera się na priorytetach kryteriów cząstkowych. Kryterium o najwyższym priorytecie jest rozważane w pierwszej kolejności. Jeśli istnieje wiele rozwiązań optymalnych dla pierwszego kryterium, rozważane jest kolejne, ale tylko w obrębie rozwiązań uzyskanych w poprzednim etapie. Trudnością w tym podejściu jest dobór priorytetów [102].

Wadą metod sprowadzających problem wielokryterialny do zagadnienia jednokryterialnego jest to, że rodzaj kompromisu pomiędzy kryteriami cząstkowymi powinien zostać zdefiniowany przez interesariusza przed rozpoczęciem optymalizacji, gdy nie jest jeszcze znana charakterystyka możliwych do uzyskania rozwiązań. Z tego powodu stosowane są metody interaktywne, w których interesariusz wielokrotnie wybiera rozwiązania lub w inny sposób ujawnia swoje preferencje podczas prowadzonych obliczeń.

Algorytmy ewolucyjne umożliwiają interaktywne rozwiązywanie problemów wielokryterialnych. Operując na populacji osobników, algorytm ewolucyjny przeszukuje przestrzeń wielokryterialną i zwraca zbiór rozwiązań, które stanowią przybliżenie frontu *Pareto*. Oznacza to, że nie wymaga się od interesariusza określenia *a priori* kompromisu między kryteriami, zamiast tego wybiera

on preferowany kompromis ze zbioru wyznaczonych już rozwiązań niezdominowanych. Wybory interesariusza wskazują pożądane kierunki dalszej eksploracji przestrzeni rozwiązań. Znane z literatury przedmiotu wielokryterialne odmiany algorytmów ewolucyjnych w wersji nieinteraktywnej omówiono w podrozdziale 1.7.

Wykorzystanie algorytmów ewolucyjnych nie wymaga wiedzy na temat możliwych kompromisów przed rozpoczęciem optymalizacji, jednak wybór jednego spośród wyznaczonych rozwiązań niezdominowanych również może być trudny dla interesariusza. Pokazano to na rysunku 4.2, na którym oznaczono oceny dla pięciuset losowo wygenerowanych strategii zespołu agentów warstwy pośredniczącej gridu. Dwukryterialna przestrzeń $(\hat{Z}_{\max},\ \Xi)$ odzwierciedla oceny rozwiązań pod względem obciążenia CPU newralgicznego węzła i kosztu zakupu komputerów. W wyznaczonym zbiorze istnieje 10 rozwiązań niezdominowanych, których oceny oznaczono jako $y_1, \ldots, y_{10}$ (rys. 4.2). Wartości współrzędnych w przestrzeni kryterialnej podano w tabeli nr 4.3.



Rysunek 4.2: Wybrane oceny strategii zespołu 12 agentów w przestrzeni kryterialnej $(\hat{Z}_{\max},\ \Xi)$ dla gridu o sześciu węzłach i realizacji dwóch zadań obliczeniowych
Źródło: opracowanie własne

Porównując rozwiązania o ocenach y_1 oraz y_2, można zauważyć, że znaczny wzrost kosztu zakupu komputerów skutkuje niewielką poprawą pod względem kryterium obciążenia CPU newralgicznego węzła. Spośród tych dwóch rozwiązań interesariusz najpewniej wybierze to, które cechuje się oceną y_2. Z kolei w przypadku ocen y_9 i y_{10} niewielkie oszczędności przy zakupie serwerów przekładają się na znaczące pogorszenie jakości rozwiązania pod względem kryterium $\hat{Z}_{\max}$. Rozważając tylko te dwa rozwiązania, można wskazać strategię o ocenie y_9 jako intuicyjnie efektywniejszy kompromis pomiędzy kryteriami cząstkowymi.

Znacznie trudniejszy jest wybór pomiędzy rozwiązaniami o numerach 2-9. Reprezentują one szeroki zakres możliwych kompromisów – od kosztownych, ale wysoce wydajnych rozwiązań do znacznie tańszych, lecz cechujących się wyższymi obciążeniami. Równocześnie wyznaczone rozwiązania równomiernie pokrywają front ocen niezdominowanych. Aby dokonać wyboru, interesariusz może wykorzystać dodatkowe kryterium, np. pobór mocy elektrycznej przez grid.

Tabela 4.3: Oceny rozwiązań niezdominowanych wyznaczonych za pomocą metody losowej

Ocena	$\hat{Z}_{\max}$ [s]	Ξ [tys. zł]	Ocena	$\hat{Z}_{\max}$ [s]	Ξ [tys. zł]
y_1	38,82	196,43	y_6	96,02	62,09
y_2	39,93	149,35	y_7	139,07	59,59
y_3	55,81	137,31	y_8	150,78	28,98
y_4	67,38	128,13	y_9	184,19	21,59
y_5	69,33	101,08	y_{10}	248,32	21,30

Źródło: opracowanie własne

Metoda losowego generowania strategii nie gwarantuje uzyskania rozwiązań wysokiej jakości. Niemniej jest ona często wykorzystywana do generowania początkowej populacji w algorytmach ewolucyjnych. Rozwiązania wyznaczone w sposób losowy mogą również stanowić punkt odniesienia dla oceny bardziej wyrafinowanych metod optymalizacji.

Dla optymalizacji uwzględniającej trzy kryteria cząstkowe zdefiniować można kryterium wektorowe F, jak niżej:

$$F : \boldsymbol{\mathcal{X}} \to \mathbb{R}^3, \tag{4.67}$$

gdzie:
$\boldsymbol{\mathcal{X}}$ – zbiór rozwiązań dopuszczalnych zdefiniowany jak w (4.10),
$\mathbb{R}^3$ – przestrzeń ocen rozwiązań, $\mathbb{R}$ – zbiór liczb rzeczywistych.

W przypadku optymalizacji trzech kryteriów cząstkowych: $\hat{Z}_{\max}$ (minimalizacja), $\widetilde{Z}_{\max}$ (minimalizacja) oraz $\kappa_{\min}^{\mathrm{RAM}}$ (maksymalizacja), ocena rozwiązania $x \in \widehat{\boldsymbol{\mathcal{X}}}$, gdzie $\widehat{\boldsymbol{\mathcal{X}}}$ jest definiowane za pomocą (4.66), zadana jest, jak niżej:

$$F(x) = [\hat{Z}_{\max}(x),\ \widetilde{Z}_{\max}(x),\ -\kappa_{\min}^{\mathrm{RAM}}(x)]^T = [F_1(x),\ F_2(x),\ F_3(x)]^T \in \mathbb{R}^3. \tag{4.68}$$

Wybrane kryteria cząstkowe są w konflikcie. Minimalizacja obciążeń komunikacyjnych jest możliwa poprzez migrację agentów do jednego węzła, co przekłada się na większe wykorzystanie pamięci operacyjnej, a w efekcie mniejszą jej rezerwę w zadanym węźle. Przemieszczenie wszystkich agentów do tego samego węzła skutkuje również dużym obciążeniem procesora. Największą rezerwę pamięci RAM w węźle newralgicznym pod tym względem można uzyskać poprzez maksymalne rozproszenie agentów, ale prowadzi to do wysokich obciążeń komunikacyjnych.

W celu wyznaczenia rozwiązań w zagadnieniu optymalizacji wielokryterialnej, konieczne jest określenie relacji porządku w *zbiorze ocen rozwiązań dopuszczalnych* $\boldsymbol{\mathcal{Y}}$. Dla optymalizacji trzech kryteriów zbiór $\boldsymbol{\mathcal{Y}} \subset \mathbb{R}^3$ definiuje się, jak niżej [14]:

$$\boldsymbol{\mathcal{Y}} = \{y \in \mathbb{R}^3 : y = F(x), x \in \widehat{\boldsymbol{\mathcal{X}}}\}. \tag{4.69}$$

Zbiór ocen rozwiązań dopuszczalnych można uporządkować za pomocą „mniejszościowej" *relacji dominowania* $\rho^{\leqslant} \subset \mathbb{R}^3 \times \mathbb{R}^3$ zadanej, jak niżej [224]:

$$\rho^{\leqslant} = \{(y,\, y') \in \boldsymbol{\mathcal{Y}} \times \boldsymbol{\mathcal{Y}} : y_n \leqslant y'_n \text{ dla } n = \overline{1,3}\}, \tag{4.70}$$

przy czym:
$y = [y_1,\, y_2,\, y_3]^T \in \boldsymbol{\mathcal{Y}}$,
$y' = [y'_1,\, y'_2,\, y'_3]^T \in \boldsymbol{\mathcal{Y}}$.

W efekcie zagadnienie optymalizacji wielokryterialnej definiuje się jako uporządkowaną trójkę $(\widehat{\boldsymbol{\mathcal{X}}}, F, \rho^{\leqslant})$. Najbardziej pożądanym rezultatem optymalizacji jest *rozwiązanie dominujące* x^{D} spełniające warunek [13]:

$$F_n(x^{\mathrm{D}}) = \min_{x \in \widehat{\boldsymbol{\mathcal{X}}}} F_n(x),\ n = \overline{1,3}. \tag{4.71}$$

Ocena dominująca $y^{\mathrm{D}} = F(x^{\mathrm{D}})$ dominuje pozostałe oceny w zbiorze $\boldsymbol{\mathcal{Y}}$ w sensie relacji $\rho^{\leqslant}$. Można pokazać, że istnieje co najwyżej jedna ocena dominująca [13, 224]. Natomiast ocenie y^{D} może odpowiadać wiele rozwiązań dominujących, które należą do zbioru $\boldsymbol{\mathcal{X}}^{\mathrm{D}}_{\leqslant}$.

Ze względu na konflikt pomiędzy kryteriami cząstkowymi w wielu praktycznych zagadnieniach polioptymalizacji ocena dominująca nie istnieje [22]. Poszukuje się zatem rozwiązań optymalnych w sensie *Pareto*. W przypadku optymalizacji N kryteriów, rozwiązanie x^{ND} nazywamy *Pareto*-optymalnym wtedy i tylko wtedy, gdy spełnia ono następujący warunek [6]:

$$\neg\exists_{x \in \widehat{\boldsymbol{\mathcal{X}}}} \left[\forall_{n=\overline{1,N}}\ F_n(x) \leqslant F_n(x^{\mathrm{ND}})\ \wedge\ \exists_{k \in \{1,\ldots,N\}}\ F_k(x) < F_k(x^{\mathrm{ND}})\right]. \tag{4.72}$$

Ocena $y^{\mathrm{ND}} = F(x^{\mathrm{ND}})$ rozwiązania optymalnego w sensie *Pareto* spełnia warunek:

$$\neg\exists_{y \in \boldsymbol{\mathcal{Y}}}\ (y,\ y^{\mathrm{ND}}) \in \rho^{\leqslant}. \tag{4.73}$$

Oznacza to, że ocena y^{ND} jest elementem niezdominowanym w zbiorze $\boldsymbol{\mathcal{Y}}$ w sensie relacji $\rho^{\leqslant}$. Przykłady ocen w dwukryterialnej przestrzeni dla rozwiązań niezdominowanych w zbiorze losowo wygenerowanych strategii agentów oznaczono na rysunku 4.2 jako $y_1, \ldots, y_{10}$.

W przypadku równoczesnej minimalizacji obciążenia CPU newralgicznego węzła, minimalizacji obciążenia komunikacyjnego newralgicznego węzła oraz maksymalizacji rezerwy pamięci RAM w węźle newralgicznym zagadnienie trójkryterialnej optymalizacji strategii zespołu agentów warstwy pośredniczącej gridu formułuje się, jak niżej:

Dla danych wejściowych jak w zagadnieniu (4.66) należy wyznaczyć zbiór $\boldsymbol{\mathcal{X}}^{\mathrm{ND}}_{\leqslant}$ *Pareto*-optymalnych strategii zespołu agentów dla problemu optymalizacji wielokryterialnej specyfikowanego jako uporządkowana trójka:

$$(\widehat{\boldsymbol{\mathcal{X}}}, F, \rho^{\leqslant}), \tag{4.74}$$

1) $\widehat{\boldsymbol{\mathcal{X}}}$ – zbiór strategii dopuszczalnych zdefiniowany, jak w problemie (4.66);

2) F – kryterium wektorowe:

$$F : \boldsymbol{\mathcal{X}} \to \mathbb{R}^3,$$

gdzie $F(x) = [\hat{Z}_{\max}(x), \widetilde{Z}_{\max}(x), -\kappa_{\min}^{\mathrm{RAM}}(x)]^T = y \in \mathbb{R}^3$ dla $x \in \widehat{\boldsymbol{\mathcal{X}}}$;

3) $\rho^{\leqslant}$ – relacja dominowania w przestrzeni $\mathbb{R}^3$:

$$\rho^{\leqslant} = \{(a,b) \in \boldsymbol{\mathcal{Y}} \times \boldsymbol{\mathcal{Y}} : a_n \leqslant b_n \text{ dla } n = \overline{1,3}\}.$$

Zbiór rozwiązań dopuszczalnych może zawierać maksymalnie $M^{Vw} I^V J^I$ elementów przy uwzględnieniu wyłącznie wymagań formalnych. Dołączenie ograniczeń opisywanych selektorami $[\mu_1, \ldots, \mu_{19}]$ zawęża zbiór rozwiązań dopuszczalnych, który w szczególnych przypadkach może być zbiorem pustym. Można pokazać, że jeśli zbiór rozwiązań dopuszczalnych $\widehat{\boldsymbol{\mathcal{X}}}$ nie jest pusty, to zbiór ocen $\boldsymbol{\mathcal{Y}} = F(\widehat{\boldsymbol{\mathcal{X}}})$ jest skończony i ograniczony. Można również pokazać, że sformułowany problem optymalizacji wielokryterialnej jest NP-trudny.

Sformułowane zagadnienie optymalizacji (4.74) podlega modyfikacji w zależności od preferencji interesariusza. Kryterium wektorowe oceny jakości gridu może zostać rozszerzone o dodatkowe kryteria cząstkowe. Co więcej, dla tego samego kryterium wektorowego sformułować można wiele zagadnień optymalizacji różniących się zestawem ograniczeń. Aplikacja AAG'16 pozwala na wybór kryteriów cząstkowych oraz ograniczeń spośród tych, które zdefiniowano wcześniej w rozprawie (tabele nr 4.1 i 4.2). Jest to prawdopodobnie pierwsza tak rozbudowana aplikacja w zakresie optymalizacji strategii sieci agentów warstwy pośredniczącej w gridach obliczeniowych [32, 209, 276].

4.5 Wnioski i uwagi

Efektywna strategia zespołu agentów warstwy pośredniczącej gridu może istotnie wpłynąć na jakość jego działania. W gridzie *Comcute* rozważa się przydział zadań użytkowników do agentów, przemieszczanie agentów i rekonfigurację komputerów w węzłach. Wymagania formalne odnośnie strategii zespołu agentów definiują rozległą przestrzeń poszukiwań dla problemów optymalizacji. Interesariusz, chcąc wyznaczyć nową strategię, formułuje zagadnienie optymalizacji za pośrednictwem aplikacji AAG'16.

Pożądane cechy strategii definiowane są za pomocą zbioru wymagań i kryteriów oceny jakości. Do istotnych kryteriów zaliczają się: łączna moc obliczeniowa gridu, koszt zakupu komputerów, koszt realizacji zadań, obciążenie newralgicznego węzła, obciążenie komunikacyjne, minimalny stopień rozproszenia agentów, rezerwy zasobów pamięci operacyjnej i pamięci trwałej, ograniczenia energetyczne, dostępność systemu i oczekiwany czas realizacji zadań.

W oparciu o dostępny zbiór kryteriów sformułować można szereg różnorodnych zagadnień optymalizacji. Ponadto dla tych samych kryteriów definiować można wybrane problemy w zależności od zestawu ograniczeń i ich wartości. Dobór ograniczeń istotnie wpływa na zbiór rozwiązań dopuszczalnych. W szczególności przy zbyt restrykcyjnych wymaganiach może on być zbiorem pustym, co wymusza na interesariuszu rozluźnienie oczekiwań.

Warto podkreślić, że rekonfiguracja może zachodzić w systemie również bez interwencji interesariusza. Agenty nadzorujące działanie systemu mogą zainicjować pracę *agentów-solwerów* w reakcji na zdarzenia istotne z punktu widzenia wybranych kryteriów oceny jakości. Zaliczają się do nich przykładowo: zmiana cen energii elektrycznej, dodanie nowego zadania obliczeniowego, awaria węzła, czy zmiana mocy oferowanej przez węzły obliczeniowe. Warto podkreślić, że inteligentne *agenty-solwery* również rezerwują zasoby gridu dla swoich obliczeń. Powinny zatem być włączone do zespołu agentów, rozszerzając funkcjonalności w warstwie pośredniczącej. Z drugiej strony, rozmieszczenie *agentów-solwerów* w węzłach warstwy pośredniczącej zmniejsza wielkości zasobów dostępnych dla podstawowych agentów zarządzających i dystrybucyjnych.

Agenty-solwery wykorzystują programowanie genetyczne do wyznaczania sprawnych strategii zespołu agentów warstwy pośredniczącej. Dla problemów jednokryterialnych zwracane jest najlepsze spośród uzyskanych rozwiązań, przy zachowaniu ograniczeń. W przypadku zagadnień wielokryterialnych wyznaczany jest zbiór rozwiązań *Pareto*-optymalnych, spośród których interesariusz może wybrać strategię do zastosowania w gridzie.

Dzięki wykorzystaniu programowania genetycznego interesariusz nie musi posiadać wiedzy *a priori* na temat kompromisów zachodzących pomiędzy strategiami niezdominowanymi. Zamiast tego wybiera on odpowiadający mu kompromis spośród przedstawionych rozwiązań. W przypadku licznego zbioru rozwiązań *Pareto*-optymalnych wybór strategii do wdrożenia w systemie może być trudny dla interesariusza. W takiej sytuacji konieczne jest uwzględnienie dodatkowego kryterium w celu wytypowania preferowanego rozwiązania. Programowanie genetyczne może zostać zastosowane w odmianie interaktywnej, w której interesariusz wielokrotnie ujawnia swoje preferencje ukierunkowując przeszukiwanie na wybrane obszary przestrzeni kryterialnej.

Zbiór ograniczeń na strategie dopuszczalne składa się z trzech obligatoryjnych ograniczeń formalnych oraz dziewiętnastu opcjonalnych ograniczeń merytorycznych, które narzucają interesariusze. Wartości selektorów ograniczeń są także wagami w funkcji kary dla rozwiązań niedopuszczalnych. Dzięki temu interesariusz może wyrazić preferencje odnośnie niepożądanych obszarów przestrzeni przeszukiwań. W wypadku kilku interesariuszy można zastosować zmodyfikowaną metodę delficką w celu wyboru funkcji celu spośród siedemnastu kryteriów oraz wyznaczenia opcjonalnych ograniczeń. W ten sposób formułowana jest instancja problemu optymalizacji (4.66).

Natomiast w wypadku zagadnienia polioptymalizacji (4.74) uwzględnia się *a priori* trzy kryteria cząstkowe. Minimalizacji podlega obciążenie CPU newralgicznego komputera, a także obciążenie komunikacyjne newralgicznego węzła. Ponadto maksymalizacji poddano rezerwy pamięci RAM w węźle newralgicznym pod tym względem. Zagadnienie trójkryterialnej optymalizacji strategii zespołu agentów warstwy pośredniczącej gridu można rozszerzyć przez dodanie kolejnych kryteriów cząstkowych i zmodyfikować pod względem zestawu ograniczeń oraz przyjętych wartości granicznych dla rozwiązań dopuszczalnych.

Rozdział 5

Metody wyznaczania optymalnych strategii sieci inteligentnych agentów

5.1 Programowanie genetyczne do optymalizacji jednokryterialnej

Do wyznaczania rozwiązań zagadnienia optymalizacji jednokryterialnej (4.66) proponuje się wykorzystanie programowania genetycznego. Ponieważ metoda ta nie posiada mechanizmów dyskryminacji rozwiązań niedopuszczalnych, wprowadzono funkcję kary, która pogarsza ocenę rozwiązań niespełniających nałożone ograniczenia. W ten sposób na etapie selekcji preferowane są strategie dopuszczalne przed niedopuszczalnymi. Funkcja kary odzwierciedla stopień przekroczenia ograniczeń – im bardziej alternatywa wykracza poza ograniczenia, tym większą otrzyma wartość kary.

Ponieważ umownie przyjmuje się w literaturze przedmiotu maksymalizację sprawności w programowaniu genetycznym, to wartość minimalizowanej funkcji celu odejmowana jest od wartości maksymalnej, jaką ta funkcja może osiągnąć. Natomiast, jeśli funkcja celu jest maksymalizowana, to jej wartość jest dodawana w oszacowaniu sprawności rozwiązania. Warto także zauważyć, że wartości wszystkich funkcji kandydujących na funkcję celu (tabela nr 4.2) są nieujemne dla rozwiązań dopuszczalnych.

Funkcje sprawności mogą być zatem dwojakiego rodzaju. W wypadku minimalizacji funkcji celu dopasowanie rozwiązania wyznaczamy za pomocą następującej zależności:

$$\mathit{fitness}(x) = \begin{cases} -F_n(x) + F_n^{\max} + P^{\max}, & \text{dla } x \in \widehat{\boldsymbol{\mathcal{X}}}, \\ -P(x) + P^{\max}, & \text{dla } x \notin \widehat{\boldsymbol{\mathcal{X}}}, \end{cases} \tag{5.1}$$

gdzie:
$F_n^{\max}$ – oszacowanie górne wartości funkcji celu F_n,
$P^{\max}$ – oszacowanie górne wartości funkcji kary P.

W wypadku maksymalizacji funkcji celu sprawność strategii obliczana jest inaczej, a mianowicie:

$$\mathit{fitness}(x) = \begin{cases} F_n(x) + P^{\max}, & \text{dla } x \in \widehat{\boldsymbol{\mathcal{X}}}, \\ -P(x) + P^{\max}, & \text{dla } x \notin \widehat{\boldsymbol{\mathcal{X}}}. \end{cases} \tag{5.2}$$

Warto zauważyć, że dla dodatnich wartości funkcji celu w zależnościach (5.1) i (5.2) warianty dopuszczalne otrzymują zawsze oceny większe niż $P^{\max}$, natomiast warianty niedopuszczalne – oceny mniejsze niż $P^{\max}$. Gwarantuje to wyższe prawdopodobieństwo selekcji dla rozwiązań dopuszczalnych.

Oszacowania górne $F_n^{\max}$ wartości funkcji celu F_n stosuje się do minimalizowanych funkcji: $\hat{Z}_{\max}$, $\widetilde{Z}_{\max}$, $\hat{Z}_{\text{suma}}$, $\widetilde{Z}_{\text{suma}}$, Ξ, K, E oraz $\overline{T}$ (tabela nr 4.2). Oszacowania górnego nie wymagają maksymalizowane funkcje: Θ, η, $\kappa_{\min}^{\text{RAM}}$, $\kappa_{\min}^{\text{HDD}}$, $\kappa_{\min}^{\text{SSD}}$, M^{RAM}, M^{HDD}, M^{SSD} oraz Γ.

W przypadku minimalizowanych funkcji celu, w oszacowaniach ich maksymalnych wartości należy uwzględnić ograniczenia nałożone przez interesariuszy. Wykorzystanie wartości granicznych dla strategii dopuszczalnych pozwala na wyznaczenie dokładniejszych oszacowań górnych niż te, uzyskane dla zbioru strategii bez ograniczeń. Interesariusz przypisuje wartość $\mu_n \geqslant 1$ w przypadku wyboru ograniczenia związanego z kryterium cząstkowym F_n. Oszacowanie od góry maksymalnej wartości obciążenia CPU newralgicznego komputera (kryterium F_1) jest sumą największych wartości w kolumnach macierzy T, co zapisujemy, jak niżej:

$$\hat{Z}_{\max}^{\max} = \begin{cases} \displaystyle\sum_{v=1}^{V} \max_{j=\overline{1,J}} \{t_{vj}\}, & \text{dla } \mu_1 = 0, \\ \hat{Z}_{\text{gr}}, & \text{dla } \mu_1 \geqslant 1. \end{cases} \tag{5.3}$$

Natomiast górne oszacowanie obciążenia komunikacyjnego newralgicznego komputera jest sumą elementów macierzy τ dla wszystkich par agentów (v,u) w systemie, co zapisujemy następująco:

$$\widetilde{Z}_{\max}^{\max} = \begin{cases} \displaystyle\sum_{v=1}^{V} \sum_{\substack{u=1 \\ u>v}}^{V} \tau_{vu}, & \text{dla } \mu_2 = 0, \\ \widetilde{Z}_{\text{gr}}, & \text{dla } \mu_2 \geqslant 1. \end{cases} \tag{5.4}$$

Oszacowanie górne funkcji $\hat{Z}_{\text{suma}}$ może być wyznaczone dla rozwiązania, w którym procesory wszystkich komputerów obciążone są w maksymalnym stopniu zgodnie z zależnością (5.3). Zachodzi wówczas:

$$\hat{Z}_{\text{suma}}^{\max} = \begin{cases} I\hat{Z}_{\max}^{\max}, & \text{dla } \mu_3 = 0, \\ \widetilde{Z}_{\text{Sgr}}, & \text{dla } \mu_3 \geqslant 1. \end{cases} \tag{5.5}$$

Oszacowanie górne sumarycznego obciążenia komunikacyjnego wszystkich komputerów można wyznaczyć, jak niżej:

$$\widetilde{Z}_{\text{suma}}^{\max} = \begin{cases} I\widetilde{Z}_{\max}^{\max}, & \text{dla } \mu_4 = 0, \\ \widetilde{Z}_{\text{Sgr}}, & \text{dla } \mu_4 \geqslant 1. \end{cases} \tag{5.6}$$

Maksymalny koszt zakupu komputerów występuje, gdy we wszystkich węzłach zainstalowano najdroższe z dostępnych komputerów, co można zapisać następująco:

$$\Xi^{\max} = \begin{cases} I \max\limits_{j=\overline{1,J}} \{\xi_j\}, & \text{dla } \mu_6 = 0, \\ \xi_{\max}, & \text{dla } \mu_6 \geqslant 1. \end{cases} \tag{5.7}$$

Maksymalny pobór mocy elektrycznej występuje, gdy we wszystkich węzłach zainstalowano komputery o największym poborze energii, co przedstawia zależność:

$$E^{\max} = \begin{cases} I \max\limits_{j=\overline{1,J}} \{\varepsilon_j^{\text{load}}\}, & \text{dla } \mu_{15} = 0, \\ \varepsilon_{\max}, & \text{dla } \mu_{15} \geqslant 1. \end{cases} \tag{5.8}$$

Maksymalny czas realizacji zadań zachodzi, gdy czasy przetwarzania i czasy komunikacji w gridzie *Comcute* są największe. Wykonanie zadań kończy się, gdy ostatni z agentów zakończy pracę w węźle, do którego został przypisany. Najdłuższym czasem przetwarzania danych cechuje się węzeł newralgiczny pod względem obciążenia CPU, a najdłuższym czasem komunikacji – węzeł newralgiczny pod względem obciążenia komunikacyjnego. Wartości maksymalne obciążeń dla węzłów newralgicznych zdefiniowano zależnościami (5.3) i (5.4). Oszacowanie górne oczekiwanego czasu realizacji zadań można wyznaczyć, jak niżej:

$$\overline{T}^{\max} = \begin{cases} \hat{Z}_{\max}^{\max} + \widetilde{Z}_{\max}^{\max} + \dfrac{\mathscr{T}_a + \mathscr{T}_u}{\mathscr{T}_a}\, \widehat{I}^{-1} \displaystyle\sum_{m=1}^{M} L(z_m) P_R(z_m) P_T(z_m), & \text{dla } \mu_{17} = 0, \\ \widehat{T}_{\text{gr}}, & \text{dla } \mu_{17} \geqslant 1. \end{cases} \tag{5.9}$$

Maksymalny koszt realizacji zadań w gridzie *Comcute* występuje przy maksymalnym czasie wykonania $\overline{T}^{\max}$ i maksymalnym poborze mocy elektrycznej $E^{\max}$, co można zapisać następująco:

$$K^{\max} = \begin{cases} c_\varepsilon \overline{T}^{\max} \left(E^{\max} + \widetilde{\varepsilon}\right), & \text{dla } \mu_7 = 0, \\ K_{\text{gr}}, & \text{dla } \mu_7 \geqslant 1. \end{cases} \tag{5.10}$$

Oprócz określenia oszacowania górnego $F_n^{\max}$ wartości minimalizowanej funkcji celu F_n za pomocą jednej z zależności (5.3)-(5.10), należy również wyznaczyć wartości funkcji kary P dla rozwiązań niedopuszczalnych oraz wartość $P^{\max}$, będącą oszacowaniem górnym tej funkcji.

Funkcja kary dla ograniczenia $\hat{Z}_{\max}(x) \leqslant \hat{Z}_{\mathrm{gr}}$ jest następująca:

$$P_1(x) = \begin{cases} 0, & \text{dla } \hat{Z}_{\max}(x) \leqslant \hat{Z}_{\mathrm{gr}}, \\ \hat{Z}_{\max}(x) - \hat{Z}_{\mathrm{gr}}, & \text{dla } \hat{Z}_{\max}(x) > \hat{Z}_{\mathrm{gr}}. \end{cases} \tag{5.11}$$

Analogicznie skonstruowane są funkcje kary dla pozostałych ograniczeń nierównościowych typu „$\leqslant$”. W szczególności zachodzi, jak niżej:

$$P_2(x) = \begin{cases} 0, & \text{dla } \widetilde{Z}_{\max}(x) \leqslant \widetilde{Z}_{\mathrm{gr}}, \\ \widetilde{Z}_{\max}(x) - \widetilde{Z}_{\mathrm{gr}}, & \text{dla } \widetilde{Z}_{\max}(x) > \widetilde{Z}_{\mathrm{gr}}, \end{cases} \tag{5.12}$$

$$P_3(x) = \begin{cases} 0, & \text{dla } \hat{Z}_{\mathrm{suma}}(x) \leqslant \hat{Z}_{\mathrm{Sgr}}, \\ \hat{Z}_{\mathrm{suma}}(x) - \hat{Z}_{\mathrm{Sgr}}, & \text{dla } \hat{Z}_{\mathrm{suma}}(x) > \hat{Z}_{\mathrm{Sgr}}, \end{cases} \tag{5.13}$$

$$P_4(x) = \begin{cases} 0, & \text{dla } \widetilde{Z}_{\mathrm{suma}}(x) \leqslant \widetilde{Z}_{\mathrm{Sgr}}, \\ \widetilde{Z}_{\mathrm{suma}}(x) - \widetilde{Z}_{\mathrm{Sgr}}, & \text{dla } \widetilde{Z}_{\mathrm{suma}}(x) > \widetilde{Z}_{\mathrm{Sgr}}, \end{cases} \tag{5.14}$$

$$P_6(x) = \begin{cases} 0, & \text{dla } \Xi(x) \leqslant \xi_{\max}, \\ \Xi(x) - \xi_{\max}, & \text{dla } \Xi(x) > \xi_{\max}, \end{cases} \tag{5.15}$$

$$P_7(x) = \begin{cases} 0, & \text{dla } K(x) \leqslant K_{\mathrm{gr}}, \\ K(x) - K_{\mathrm{gr}}, & \text{dla } K(x) > K_{\mathrm{gr}}, \end{cases} \tag{5.16}$$

$$P_{15}(x) = \begin{cases} 0, & \text{dla } E(x) \leqslant \varepsilon_{\max}, \\ E(x) - \varepsilon_{\max}, & \text{dla } E(x) > \varepsilon_{\max}, \end{cases} \tag{5.17}$$

$$P_{17}(x) = \begin{cases} 0, & \text{dla } \overline{T}(x) \leqslant \widehat{T}_{\mathrm{gr}}, \\ \overline{T}(x) - \widehat{T}_{\mathrm{gr}}, & \text{dla } \overline{T}(x) > \widehat{T}_{\mathrm{gr}}. \end{cases} \tag{5.18}$$

Z kolei kary dla ograniczeń nierównościowych typu „$\geqslant$” wyznaczane są następująco:

$$P_5(x) = \begin{cases} 0, & \text{dla } \Theta(x) \geqslant \vartheta_{\min}, \\ \vartheta_{\min} - \Theta(x), & \text{dla } \Theta(x) < \vartheta_{\min}, \end{cases} \tag{5.19}$$

$$P_8(x) = \begin{cases} 0, & \text{dla } \eta(x) \geqslant \eta_{\min} \\ \eta_{\min} - \eta(x), & \text{dla } \eta(x) < \eta_{\min}, \end{cases} \tag{5.20}$$

$$P_9(x) = \begin{cases} 0, & \text{dla } \kappa_{\min}^{\mathrm{RAM}}(x) \geqslant 0, \\ -\kappa_{\min}^{\mathrm{RAM}}(x), & \text{dla } \kappa_{\min}^{\mathrm{RAM}}(x) < 0, \end{cases} \tag{5.21}$$

$$P_{10}(x) = \begin{cases} 0, & \text{dla } \kappa_{\min}^{\mathrm{HDD}}(x) \geqslant 0, \\ -\kappa_{\min}^{\mathrm{HDD}}(x), & \text{dla } \kappa_{\min}^{\mathrm{HDD}}(x) < 0, \end{cases} \tag{5.22}$$

$$P_{11}(x) = \begin{cases} 0, & \text{dla } \kappa_{\min}^{\mathrm{SSD}}(x) \geqslant 0, \\ -\kappa_{\min}^{\mathrm{SSD}}(x), & \text{dla } \kappa_{\min}^{\mathrm{SSD}}(x) < 0, \end{cases} \tag{5.23}$$

$$P_{12}(x) = \begin{cases} 0, & \text{dla } M^{\mathrm{RAM}}(x) \geqslant M_{\mathrm{gr}}^{\mathrm{RAM}}, \\ M_{\mathrm{gr}}^{\mathrm{RAM}} - M^{\mathrm{RAM}}(x), & \text{dla } M^{\mathrm{RAM}}(x) < M_{\mathrm{gr}}^{\mathrm{RAM}}, \end{cases} \tag{5.24}$$

$$P_{13}(x) = \begin{cases} 0, & \text{dla } M^{\mathrm{HDD}}(x) \geqslant M_{\mathrm{gr}}^{\mathrm{HDD}}, \\ M_{\mathrm{gr}}^{\mathrm{HDD}} - M^{\mathrm{HDD}}(x), & \text{dla } M^{\mathrm{HDD}}(x) < M_{\mathrm{gr}}^{\mathrm{HDD}}, \end{cases} \tag{5.25}$$

$$P_{14}(x) = \begin{cases} 0, & \text{dla } M^{\mathrm{SSD}}(x) \geqslant M_{\mathrm{gr}}^{\mathrm{SSD}}, \\ M_{\mathrm{gr}}^{\mathrm{SSD}} - M^{\mathrm{SSD}}(x), & \text{dla } M^{\mathrm{SSD}}(x) < M_{\mathrm{gr}}^{\mathrm{SSD}}, \end{cases} \tag{5.26}$$

$$P_{16}(x) = \begin{cases} 0, & \text{dla } \Gamma(x) \geqslant \Gamma_{\mathrm{gr}}, \\ \Gamma_{\mathrm{gr}} - \Gamma(x), & \text{dla } \Gamma(x) < \Gamma_{\mathrm{gr}}. \end{cases} \tag{5.27}$$

Ograniczenie dotyczące liczby dostępnych komputerów wybranych typów, można zapisać następująco:

$$\varpi_j(x) = \left(\nu_j - \sum_{i=1}^{I} x_{i,j}^{\beta} \right) \geqslant 0, \ j \in \{\mathcal{J} \setminus \mathcal{J}^*\} \text{ dla } \mu_{18} \neq 0. \tag{5.28}$$

Najbardziej deficytowy pod względem liczby dostępnych komputerów jest typ komputera cechujący się najmniejszą wartością $\varpi_j(x)$, a ograniczenie (5.28) zapisujemy, jak niżej:

$$\varpi^{\min}(x) = \min_{j \in \{\mathcal{J} \setminus \mathcal{J}^*\}} \{\varpi_j(x)\} \geqslant 0, \text{ dla } \mu_{18} \neq 0. \tag{5.29}$$

Kara za niespełnienie ograniczenia jest wyznaczana za pomocą następującej zależności:

$$P_{18}(x) = \begin{cases} 0 & \text{dla } \varpi^{\min}(x) \geqslant 0, \\ -\varpi^{\min}(x) & \text{dla } \varpi^{\min}(x) < 0. \end{cases} \tag{5.30}$$

Wartość funkcji kary za brak instalacji dokładnej liczby komputerów wybranych rodzajów, może być przedstawiona w następującej postaci:

$$P_{19}(x) = \sum_{j \in \mathcal{J}^*} \left| \sum_{i=1}^{I} x_{i,j}^{\beta} - \nu_j \right|, \text{ dla } \mu_{19} \neq 0. \tag{5.31}$$

Reasumując, wartość funkcji kary $P(x)$ dla strategii x jest sumą ważoną kar za niespełnienie poszczególnych ograniczeń, dla których wagami są współczynniki $\mu_1, \ldots, \mu_{19}$ wprowadzone przez interesariusza:

$$P(x) = \sum_{k=1}^{19} \mu_k P_k(x), \ x \notin \widehat{\boldsymbol{\mathcal{X}}}. \tag{5.32}$$

Do wyznaczenia sprawności strategii niezbędna jest wartość maksymalna funkcji kary $P^{\max}$, którą wyznacza się następująco:

$$\begin{aligned} P^{\max} = {} & \mu_1(\hat{Z}^{\max}_{\max} - \hat{Z}_{\mathrm{gr}}) + \mu_2(\widetilde{Z}^{\max}_{\max} - \widetilde{Z}_{\mathrm{gr}}) + \mu_3(\hat{Z}^{\max}_{\mathrm{suma}} - \hat{Z}_{\mathrm{Sgr}}) + \mu_4(\widetilde{Z}^{\max}_{\mathrm{suma}} - \widetilde{Z}_{\mathrm{Sgr}}) + \\ & \mu_5\vartheta_{\min} + \mu_6(\Xi^{\max} - \xi_{\max}) + \mu_7(K^{\max} - K_{gr}) + \mu_8\eta_{min} + \mu_9 I\kappa^{\mathrm{RAM}}_{\max} + \\ & \mu_{10} I\kappa^{\mathrm{HDD}}_{\max} + \mu_{11} I\kappa^{\mathrm{SSD}}_{\max} + \mu_{12}(M^{\mathrm{RAM}}_{\mathrm{gr}} + I\kappa^{\mathrm{RAM}}_{\max}) + \mu_{13}(M^{\mathrm{HDD}}_{\mathrm{gr}} + I\kappa^{\mathrm{HDD}}_{\max}) + \\ & \mu_{14}(M^{\mathrm{SSD}}_{\mathrm{gr}} + I\kappa^{\mathrm{SSD}}_{\max}) + \mu_{15}(E^{\max} - \varepsilon_{\max}) + \mu_{16}\Gamma_{\mathrm{gr}} + \mu_{17}(\overline{T}^{\max} - \widehat{T}_{\mathrm{gr}}) + \\ & \mu_{18} I + \mu_{19} JI, \end{aligned} \tag{5.33}$$

gdzie:
$\kappa^{\mathrm{RAM}}_{\max}$ – maksymalna wielkość pamięci RAM wśród dostępnych komputerów [GB],
$\kappa^{\mathrm{HDD}}_{\max}$ – maksymalna wielkość pamięci HDD wśród dostępnych komputerów [TB],
$\kappa^{\mathrm{SSD}}_{\max}$ – maksymalna wielkość pamięci SSD wśród dostępnych komputerów [TB].

Obliczenie wartości funkcji celu i funkcji kary dla osobnika uzyskanego za pomocą programowania genetycznego wymaga określenia reprezentowanej przez niego strategii zespołu agentów. Osobnik w programowaniu genetycznym składa się z drzew programów. Liście drzew zawierają odwołania do elementów ze *zbioru danych wejściowych* $\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}$ zagadnienia optymalizacji (4.66). Liście mogą również reprezentować wartości losowe. Z kolei wierzchołki wewnętrzne odpowiadają operatorom ze zbioru procedur $\boldsymbol{\mathcal{F}}$. Konieczne jest wprowadzenie mapowania genotyp-fenotyp, które pozwoli na interpretowanie drzew programów (genotyp osobnika) jako zakodowanej całkowitoliczbowo strategii X (fenotyp):

$$X = [X^z_1, \ldots, X^z_m, \ldots, X^z_M, X^\alpha_1, \ldots, X^\alpha_v, \ldots, X^\alpha_V, X^\beta_1, \ldots, X^\beta_i, \ldots, X^\beta_I]^T,$$

przy czym: $1 \leqslant X^z_m \leqslant V_W, m = \overline{1,M}, 1 \leqslant X^\alpha_v \leqslant I, v = \overline{1,V}, 1 \leqslant X^\beta_i \leqslant J, i = \overline{1,I}$.

Mapowanie wykorzystuje procedurę *interpretacja* $\boldsymbol{\mathcal{I}}$. Przyjmuje się, że zmiennej decyzyjnej X^z_m odpowiada V_W drzew programów ponumerowanych od 1 do V_W. Każde drzewo dla zbioru parametrów wejściowych $\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}$ zwraca liczbę rzeczywistą. Numer drzewa o największym wyniku jest wartością zmiennej decyzyjnej X^z_m. Analogicznie, zmiennej X^α_v odpowiada I drzew, a zmiennej X^β_i – J drzew. Do wyznaczenia strategii X wykorzystywany jest *las* obejmujący $D_{\mathrm{count}} = MV_W + (V + J)I$ drzew. Osobnik składa się z D_{count} drzew, które konstruują wektor $\mathcal{D}$ zdefiniowany następująco:

$$\begin{aligned} \mathcal{D} = [& \mathcal{D}^{(1)}_{X^z_1}, \ldots, \mathcal{D}^{(V_W)}_{X^z_1}, \ldots, \mathcal{D}^{(1)}_{X^z_m}, \ldots, \mathcal{D}^{(V_W)}_{X^z_m}, \ldots, \mathcal{D}^{(1)}_{X^z_M}, \ldots, \mathcal{D}^{(V_W)}_{X^z_M}, \\ & \mathcal{D}^{(1)}_{X^\alpha_1}, \ldots, \mathcal{D}^{(I)}_{X^\alpha_1}, \ldots, \mathcal{D}^{(1)}_{X^\alpha_v}, \ldots, \mathcal{D}^{(I)}_{X^\alpha_v}, \ldots, \mathcal{D}^{(1)}_{X^\alpha_V}, \ldots, \mathcal{D}^{(I)}_{X^\alpha_V}, \\ & \mathcal{D}^{(1)}_{X^\beta_1}, \ldots, \mathcal{D}^{(J)}_{X^\beta_1}, \ldots, \mathcal{D}^{(1)}_{X^\beta_i}, \ldots, \mathcal{D}^{(J)}_{X^\beta_i}, \ldots, \mathcal{D}^{(1)}_{X^\beta_I}, \ldots, \mathcal{D}^{(J)}_{X^\beta_I}]^T, \end{aligned} \tag{5.34}$$

gdzie:
$\mathcal{D}^{(1)}_{X^z_m}, \ldots, \mathcal{D}^{(V_W)}_{X^z_m}$ – drzewa programów do wyznaczenia wartości zmiennej decyzyjnej X^z_m,
$\mathcal{D}^{(1)}_{X^\alpha_v}, \ldots, \mathcal{D}^{(I)}_{X^\alpha_v}$ – drzewa programów do wyznaczenia wartości zmiennej decyzyjnej X^α_v,
$\mathcal{D}^{(1)}_{X^\beta_i}, \ldots, \mathcal{D}^{(J)}_{X^\beta_i}$ – drzewa programów do wyznaczenia wartości zmiennej decyzyjnej X^β_i.

Procedura agregacji Ψ integruje drzewa z wektora $\mathcal{D}$ związane z poszczególnymi zmiennymi decyzyjnymi, konstruując w ten sposób *program do wyznaczania strategii zespołu agentów warstwy pośredniczącej w gridzie* χ. Program χ składa się z $M + V + I$ procedur programistycznych, jak niżej:

$$\chi = [\chi_1^z, \ldots, \chi_m^z, \ldots, \chi_M^z, \chi_1^\alpha, \ldots, \chi_v^\alpha, \ldots, \chi_V^\alpha, \chi_1^\beta, \ldots, \chi_i^\beta, \ldots, \chi_I^\beta]^T, \tag{5.35}$$

gdzie:
χ_m^z – procedura, w której zaagregowano drzewa $\mathcal{D}_{X_m^z}^{(1)}, \ldots, \mathcal{D}_{X_m^z}^{(V_W)}$,
χ_v^α – procedura, w której zaagregowano drzewa $\mathcal{D}_{X_v^\alpha}^{(1)}, \ldots, \mathcal{D}_{X_v^\alpha}^{(I)}$,
χ_i^β – procedura, w której zaagregowano drzewa $\mathcal{D}_{X_i^\beta}^{(1)}, \ldots, \mathcal{D}_{X_i^\beta}^{(J)}$.

Reprezentacja osobnika w postaci programu χ jest użyteczna do wyznaczenia strategii X zgodnie z procedurą *interpretacja*. Każdej zmiennej decyzyjnej strategii X odpowiada jedna procedura programu χ. Strategię wyznacza się w zależności od wartości danych wejściowych zagadnienia optymalizacji $\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}$, jak niżej:

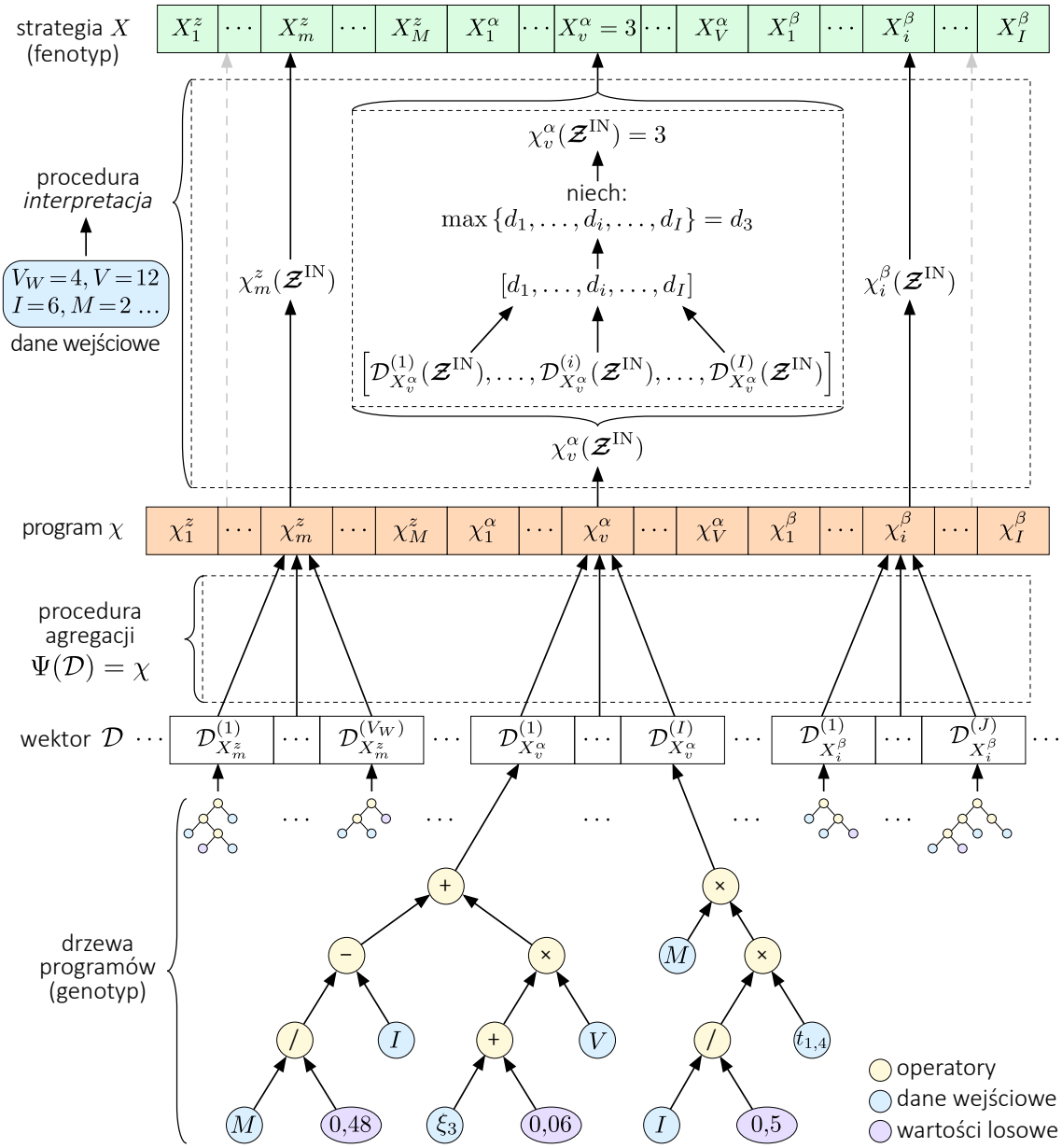
$$\begin{aligned} X &= [X_1^z, \ldots, X_M^z, X_1^\alpha, \ldots, X_V^\alpha, X_1^\beta, \ldots, X_I^\beta]^T \\ &= [\chi_1^z(\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}), \ldots, \chi_M^z(\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}),\ \chi_1^\alpha(\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}), \ldots, \chi_V^\alpha(\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}),\ \chi_1^\beta(\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}), \ldots, \chi_I^\beta(\boldsymbol{\mathcal{Z}}^{\mathrm{IN}})]^T. \end{aligned} \tag{5.36}$$

Sposób wyznaczania strategii X w oparciu o genotyp osobnika w programowaniu genetycznym pokazano na rysunku 5.1. Warto zauważyć, że metoda gwarantuje wyznaczenie rozwiązań spełniających wymagania formalne, a co za tym idzie – należących do zbioru $\boldsymbol{\mathcal{X}}$ zdefiniowanego zależnością (4.10). Wynika to z faktu, że wartość zmiennej decyzyjnej jest wybierana jako numer drzewa, które zwraca największą wartość dla danych wejściowych $\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}$. Natomiast liczba drzew wykorzystywanych dla wybranej zmiennej decyzyjnej odpowiada liczbie jej możliwych wartości.

Można rozważać inne warianty procedury *interpretacja*, np. wartość jednej zmiennej decyzyjnej wektora X wyznaczać na podstawie jednego drzewa programu, które zwraca wartości ze zbioru liczb rzeczywistych. W tym wypadku dylematem jest wyznaczenie ograniczeń dolnych i górnych dla wartości obliczanych za pomocą drzew, a następnie podzielenie tego zakresu na podprzedziały reprezentujące wartości zmiennej decyzyjnej. Inne podejście polega na wykorzystaniu operacji *modulo* na zaokrąglonej wartości zwróconej przez drzewo programu. Wadą takiego podejścia jest niestabilność zwracanych wyników – niewielka zmiana parametrów wejściowych może prowadzić do znacząco różnych rozwiązań.

W celu przeprowadzenia eksperymentów numerycznych jako podstawowy zbiór procedur w programowaniu genetycznym zastosowano zbiór $\boldsymbol{\mathcal{F}} = \{+, -, *, /\}$, który składa się z dwuargumentowych operatorów arytmetycznych. Warto podkreślić, że operator dzielenia jest zdefiniowany tak, że przy dzieleniu przez zero zwraca wartość *inf*, która reprezentuje nieskończoność.

Każde drzewo programistyczne ograniczone jest maksymalną liczbą wierzchołków D_{Vertex} i liczbą poziomów D_{Level}. Niech $D_{\mathrm{Procedure}}$ jest maksymalną liczbą wierzchołków reprezentujących procedury, a D_{Argument} oznacza maksymalną liczbę wierzchołków reprezentujących terminale (dane wejściowe i liczby losowe). Zachodzi: $D_{\mathrm{Vertex}} = D_{\mathrm{Procedure}} + D_{\mathrm{Argument}}$.



Rysunek 5.1: Mapowanie genotyp-fenotyp w programowaniu genetycznym do wyznaczania strategii zespołu agentów
Źródło: opracowanie własne

Generowanie drzew populacji początkowej może odbywać się za pomocą metody konstruowania pełnych drzew, metody przyrostowego konstruowania drzew lub metody hybrydowej, w zależności od wyboru dokonanego przez interesariusza. Procedurę budowania populacji początkowej rozszerzono o możliwość weryfikacji, czy wszystkie dane wejściowe wykorzystano w lesie służącym do wyznaczenia strategii zespołu agentów warstwy pośredniczącej gridu. Dopiero po uwzględnieniu w programie wszystkich danych wejściowych odbywa się weryfikacja, czy liczba wierzchołków przekracza D_{Vertex}.

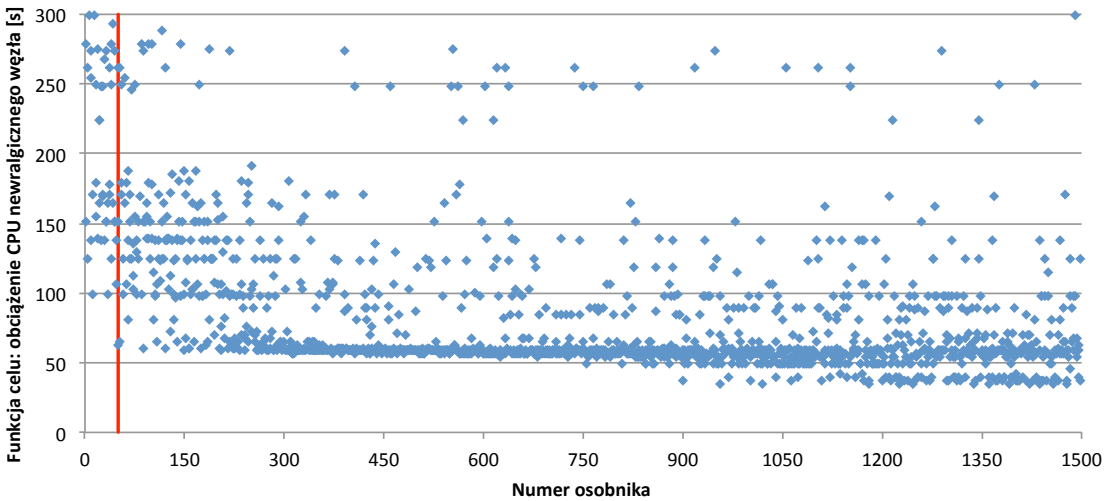
Agenty wyznaczające rozwiązania zagadnienia optymalizacji jednokryterialnej z ograniczeniami $\boldsymbol{\mathcal{Z}}(F_n, \mu)$, w których wykorzystuje się programowanie genetyczne ze zbiorem procedur $\boldsymbol{\mathcal{F}}$, oznaczamy jako $GP(F_n, \mu, \boldsymbol{\mathcal{F}})$. Przeprowadzono eksperymenty z wykorzystaniem agentów $GP(F_1, \mu, \boldsymbol{\mathcal{F}})$, które rozwiązują instancję zagadnienia optymalizacji jednokryterialnej zdefiniowaną, jak niżej:

Dla danych wejściowych: $[\mu_1, \ldots, \mu_{19}]$, $M = 2$, $\{z_1, z_2\}$, $L(z_1) = 128000$, $L(z_2) = 256000$, $P_R(z_1) = P_R(z_2) = 1$, $P_W(z_1) = P_W(z_2) = 4$, $P_T(z_1) = P_T(z_2) = 5$ s, $V_W = 4$, $V = 12$, $I = 6$, $J = 12$ (typy komputerów opisano w tabeli nr 3.1), $T = [t_{vj}]_{V \times J}$ zdefiniowane jak w (4.1) oraz $\tau = [\tau_{vu}]_{V \times V}$ zdefiniowane jak w (4.2), należy wyznaczyć x^*, takie że:

$$\hat{Z}_{\max}(x^*) = \min_{x \in \widehat{\boldsymbol{\mathcal{X}}}} \hat{Z}_{\max}(x), \tag{5.37}$$

gdzie:
$\hat{Z}_{\max}$ – kryterium obciążenia procesorów newralgicznego komputera [s],
$\widehat{\boldsymbol{\mathcal{X}}}$ – zbiór rozwiązań dopuszczalnych zdefiniowany, jak w zagadnieniu (4.66).

Rozważany problem cechuje się $MV_W + I(V + J) = 152$ binarnymi zmiennymi decyzyjnymi oraz $M + V + I = 20$ całkowitoliczbowymi zmiennymi decyzyjnymi. Binarna przestrzeń przeszukiwań zawiera $5{,}7 \cdot 10^{45}$ elementów, a całkowitoliczbowa – ${V_W}^M I^V J^I = 4^2 \cdot 6^{12} \cdot 12^6 \approx 10^{17}$ elementów. W programowaniu genetycznym na każdego osobnika przypadają 152 drzewa. Przyjęto rozmiar populacji na poziomie $G_{\text{size}} = 50$ osobników oraz limit $G_{\max} = 100$ epok. Na rysunku 5.2 zobrazowano oceny osobników należących do 30 początkowych epok dla jednego z zarejestrowanych przebiegów programowania genetycznego.



Rysunek 5.2: Wartości funkcji celu dla rozwiązań wyznaczonych przez agenta $GP(F_1, \mu, \boldsymbol{\mathcal{F}})$
Źródło: opracowanie własne

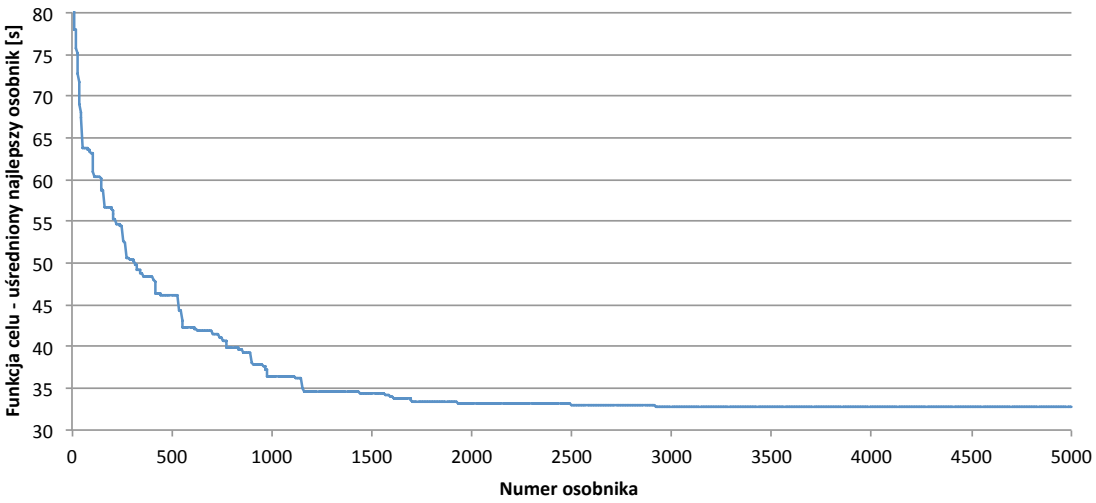
Czerwona linia oddziela wartości funkcji celu dla losowo wygenerowanych osobników populacji początkowej od ocen rozwiązań będących efektem programowania genetycznego. Widoczne jest, że wraz z kolejnymi epokami presja selekcyjna nakierowuje większość osobników w stronę rozwiązania

optymalnego. Po 30 epokach uzyskano wartość funkcji celu 33,69 s, która została poprawiona do 31,17 s po 72 epoce. Dalsza ewolucja do limitu 100 epok nie przyniosła rozwiązań o lepszych ocenach. Najlepszy z osobników reprezentuje strategię zespołu agentów warstwy pośredniczącej zdefiniowaną, jak niżej:

$$X^z = [2, 1],\ X^\alpha = [6, 4, 1, 3, 2, 2, 2, 2, 2, 5, 2, 6],\ X^\beta = [12, 11, 11, 11, 1, 12]. \tag{5.38}$$

Zauważalne jest, że dla węzłów podlegających dużym obciążeniom wybrano komputery cechujące się wysoką wydajnością. Równocześnie agenty klasy W – generujące większe obciążenie CPU niż agenty klasy S – rozdzielono między różne węzły gridu. Dla węzła ω_5 wybrano komputer typu β_1, który posiada mniejszą wydajność. Ponieważ w węźle tym uruchomiony jest tylko jeden agent klasy S, więc większa moc obliczeniowa nie jest konieczna.

Programowanie genetyczne jest metodą stochastyczną, zatem dla oceny jego zbieżności w funkcji liczby mutacji uruchomiono 10 agentów $GP(F_1, \mu, \boldsymbol{\mathcal{F}})$. Następnie uśredniono najlepsze wartości funkcji celu uzyskane przez różne agenty. Na rysunku 5.3 zaprezentowano wyniki eksperymentu. Średni czas obliczeń dla populacji 50 osobników i 100 generacji programowania genetycznego realizowanego przez agenty $GP(F_1, \mu, \boldsymbol{\mathcal{F}})$, wynosi 43,2 s na komputerze klasy PC wyposażonym w procesor Intel Core i5-4258U.

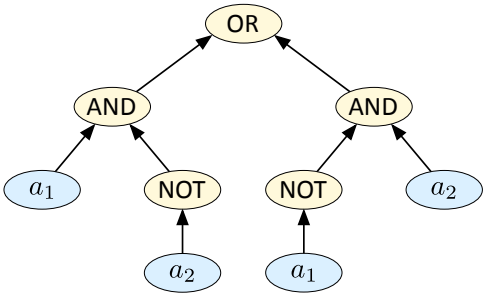


Rysunek 5.3: Uśredniona zbieżność wartości funkcji celu wyznaczonych za pomocą agentów $GP(F_1, \mu, \boldsymbol{\mathcal{F}})$
Źródło: opracowanie własne

Podstawowy zbiór procedur można rozszerzyć o procedurę *IF*, która pozwala na warunkowe wykonanie poddrzew w zależności od wartości pierwszego argumentu. Przyjmuje się, że wartości ujemne (w szczególności -1), reprezentują logiczną wartość *fałsz*, podczas gdy wartości nieujemne (w szczególności 1) – logiczną *prawdę*. Złożone warunki w instrukcji *IF* można konstruować za pomocą operatorów *OR*, *AND* oraz *NOT*, które zwracają wartości liczbowe -1 lub 1 (rys. 5.4).

Warto podkreślić, że dwa operatory *NOT* i *OR* lub *NOT* i *AND* wystarczą do skonstruowania dowolnej funkcji logicznej [161].

Argumentami zmodyfikowanych operatorów *NOT*, *OR* i *AND* są liczby rzeczywiste, co pozwala na zachowanie *zgodności* z operatorami arytmetycznymi w zbiorze $\mathcal{F}$. Wszystkie operatory przyjmują i zwracają wartości rzeczywiste, toteż argumentem każdego z nich może być wynik każdego operatora oraz wszystkie terminale. Zdefiniowane w ten sposób operatory *NOT*, *OR* i *AND* nie są klasycznymi operatorami logicznymi i mogą wystąpić w dowolnym wierzchołku programu, który nie jest liściem.



Zbiór terminali: $\mathcal{T} = \{a_1, a_2\},\ a_1, a_2 \in \{-1, 1\}$

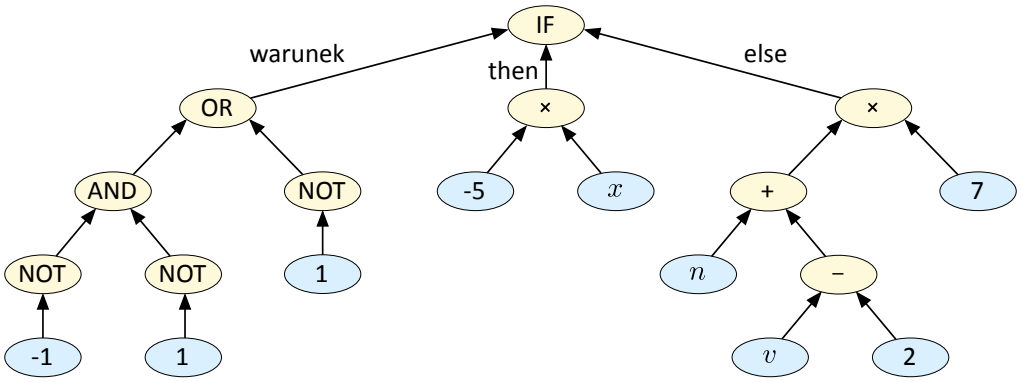
Zbiór procedur: $\mathcal{F} = \{AND, OR, NOT\}$

Wyrażenie w języku Common Lisp:

```
(OR
   (AND
      a1 (NOT a2))
   (AND
      (NOT a1) a2))
```

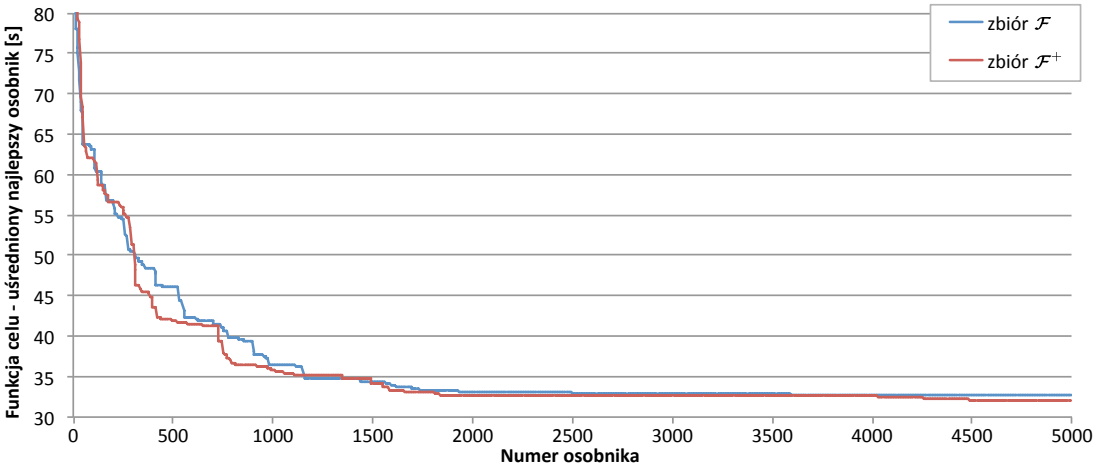
Rysunek 5.4: Zastosowanie operatorów *OR*, *AND* i *NOT* do implementacji operatora *XOR*
Źródło: opracowanie własne

Proponuje się trójargumentową procedurę *IF* (rys. 5.5). Pierwszym argumentem jest wartość wykorzystywana jako warunek. Drugi argument określa wartość zwracaną przez procedurę *IF*, jeśli warunek zostanie zinterpretowany jako *prawda*. Trzeci argument określa wartość zwracaną, gdy warunek jest fałszywy.



Rysunek 5.5: Przykładowa procedura warunkowa *IF* w programowaniu genetycznym
Źródło: opracowanie własne

W celu przeprowadzenia kolejnych eksperymentów numerycznych jako zbiór procedur zastosowano $\mathcal{F}^+ = \{+, -, *, /, IF, OR, AND, NOT\}$. Podobnie jak poprzednio, uruchomiono 10 agentów $GP(F_1, \mu, \mathcal{F}^+)$, a następnie uśredniono otrzymane rezultaty w celu wyznaczenia zbieżności w zależności od liczby mutacji. Na rysunku 5.6 zobrazowano uzyskane wyniki na tle rezultatów testowanych wcześniej agentów $GP(F_1, \mu, \mathcal{F})$.



Rysunek 5.6: Porównanie zbieżności agentów $GP(F_1, \mu, \boldsymbol{\mathcal{F}})$ i $GP(F_1, \mu, \boldsymbol{\mathcal{F}}^+)$
Źródło: opracowanie własne

Zbieżność programowania genetycznego dla obu zbiorów funkcji jest zbliżona. Agenty klasy $GP(F_1, \mu, \boldsymbol{\mathcal{F}}^+)$ są jednak w stanie wyznaczyć rozwiązania o wartości funkcji celu 30,05 s, która nie została uzyskana przez żadnego z agentów $GP(F_1, \mu, \boldsymbol{\mathcal{F}})$ przy limicie 100 epok ewolucji. Można wnioskować, że wprowadzenie dodatkowych operatorów *IF*, *NOT*, *OR* oraz *AND* ułatwia przeszukiwanie przestrzeni rozwiązań dopuszczalnych. Strategia o ocenie 30,05 s reprezentowa przez najlepszego osobnika zadana jest, jak niżej:

$$X^z = [3,3],\ X^\alpha = [5,1,2,6,3,4,5,6,1,4,3,4],\ X^\beta = [12,12,11,10,12,12]. \tag{5.39}$$

Równocześnie wykorzystanie zbioru procedur $\boldsymbol{\mathcal{F}}^+$ wydłuża średni czas obsługi jednej epoki do 576 ms, co stanowi wzrost o 30% w stosunku do zbioru $\boldsymbol{\mathcal{F}}$. Nawet w tej sytuacji pełny przebieg programowania genetycznego dla 100 epok może zostać zrealizowany w łącznym czasie poniżej minuty.

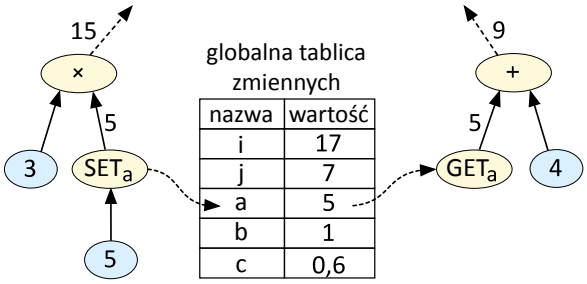
Zbiór procedur można rozszerzyć również o procedurę pętli. Pierwszym argumentem proponowanej procedury *WHILE* jest warunek kontynuacji obliczeń w pętli, który jest interpretowany, jak w przypadku procedury *IF*. Drugi argument reprezentuje ciało pętli – poddrzewo programu, którego wykonanie będzie powtarzane dopóki spełniony jest warunek. Gdy warunek przyjmie wartość fałszu, działanie pętli kończy się, a wartością zwracaną przez procedurę *WHILE* jest wynik ostatniego wykonania ciała pętli.

Aby pętla mogła się zakończyć, konieczna jest zmiana wartości jej warunku z logicznej prawdy na fałsz. Struktura poddrzewa warunku jest jednak niezmienna, a wartości danych wejściowych są stałe podczas wykonania drzewa programu. Umożliwienie zmian wartości warunku wymaga wprowadzenia zmiennych, które mogą być modyfikowane w trakcie działania programu.

W tym celu zastosowano dwie dodatkowe procedury: *GET*, która służy do odczytywania wartości zmiennej, a także *SET* – do przypisywania nowych wartości. Zmienne wykorzystywane

w obrębie drzewa są przechowywane w globalnej tablicy. Przyjęto, że liczba zmiennych D_{var} zdefiniowana jest przed uruchomieniem programowania genetycznego. Zmienne rozróżnia się w oparciu o unikatowe nazwy wyspecyfikowane w wektorze $Vars = [var_1, \ldots, var_d, \ldots, var_{D_{\text{var}}}]$, przy czym var_d jest nazwą d-tej zmiennej. Dla zmiennej var_d włącza się do zbioru procedur operator GET_{var_d} służący do jej odczytu i operator SET_{var_d} pozwalający na zmianę jej wartości. Zbiór procedur zawiera zatem tyle par $(GET_{var_d}, SET_{var_d})$, $d = \overline{1, D_{\text{var}}}$, ile zmiennych może być wykorzystanych w obrębie drzewa.

Operator GET_{var_d} nie ma żadnych argumentów i zwraca wartość d-tej zmiennej jako liczbę rzeczywistą. Z kolei operator SET_{var_d} ma jeden argument, który jest nową wartością d-tej zmiennej do zapisania w globalnej tablicy. Równocześnie operator SET_{var_d} zwraca nową wartość zmiennej. Operatory GET_{var_d} i SET_{var_d} mogą występować w dowolnym miejscu drzewa. W rezultacie wartość wyznaczona w jednej gałęzi drzewa może zostać wykorzystana w innej, bez konieczności odtwarzania poddrzewa, które do tej wartości prowadzi. Działanie operatorów zobrazowano na rysunku 5.7.



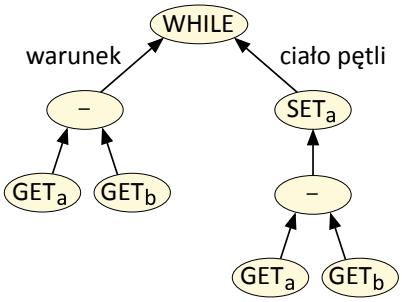
Rysunek 5.7: Operatory obsługi zmiennych w drzewie programu
Źródło: opracowanie własne

Załóżmy, że w pętli *WHILE* poddrzewo warunku zwraca początkowo wartość odpowiadającą logicznej *prawdzie*. Aby pętla została zrealizowana w skończonej liczbie kroków, konieczne jest występowanie operatorów obsługi zmiennych w obrębie drzewa. Poddrzewo warunku powinno zawierać przynajmniej jedno wystąpienie operatora GET_{var_d}, a poddrzewo ciała – operatora SET_{var_d}. Dzięki temu w wyniku wykonania ciała pętli może dojść do zmiany wartości warunku. Obecność operatorów *SET* i *GET* nie gwarantuje jednak zakończenia obliczeń w skończonej liczbie iteracji. Z tego względu wprowadza się limit maksymalnej liczby iteracji D_{MaxIt}. Po jego osiągnięciu pętla zostaje zakończona niezależnie od wartości warunku.

W szczególnych przypadkach program może zawierać zagnieżdżone pętle. Przekłada się to na pogorszenie wydajności programowania genetycznego. Z tego powodu wprowadzono dodatkowy parametr $D_{\text{MaxRuntime}}$ określający limit czasu, po którym wykonanie drzewa zostaje przerwane.

Przykładowe zastosowanie procedury pętli *WHILE* w programowaniu genetycznym przedstawiono na rysunku 5.8. Zaprezentowane drzewo pozwala na wyznaczenie reszty z dzielenia zmiennej a przez zmienną b. Wyrażenie $a - b$ w gałęzi warunku zwraca wartości nieujemne – interpretowane jako logiczna prawda – gdy $a \geqslant b$. Jeśli ten warunek jest spełniony, wykonywane jest ciało pętli, w którym ponownie obliczane jest wyrażenie $a - b$, a jego wynik zostaje zapisany do zmiennej a.

Operator SET_a zwraca nową wartość zmiennej. Z kolei procedura *WHILE* zwraca wartość ciała pętli z ostatniej iteracji, gdy warunek kontynuacji przestanie być prawdziwy.



Pętla while w pseudokodzie:

```
while a >= b
    a := a - b;
return a
```

Rysunek 5.8: Przykładowa procedura *WHILE* w programowaniu genetycznym
Źródło: opracowanie własne

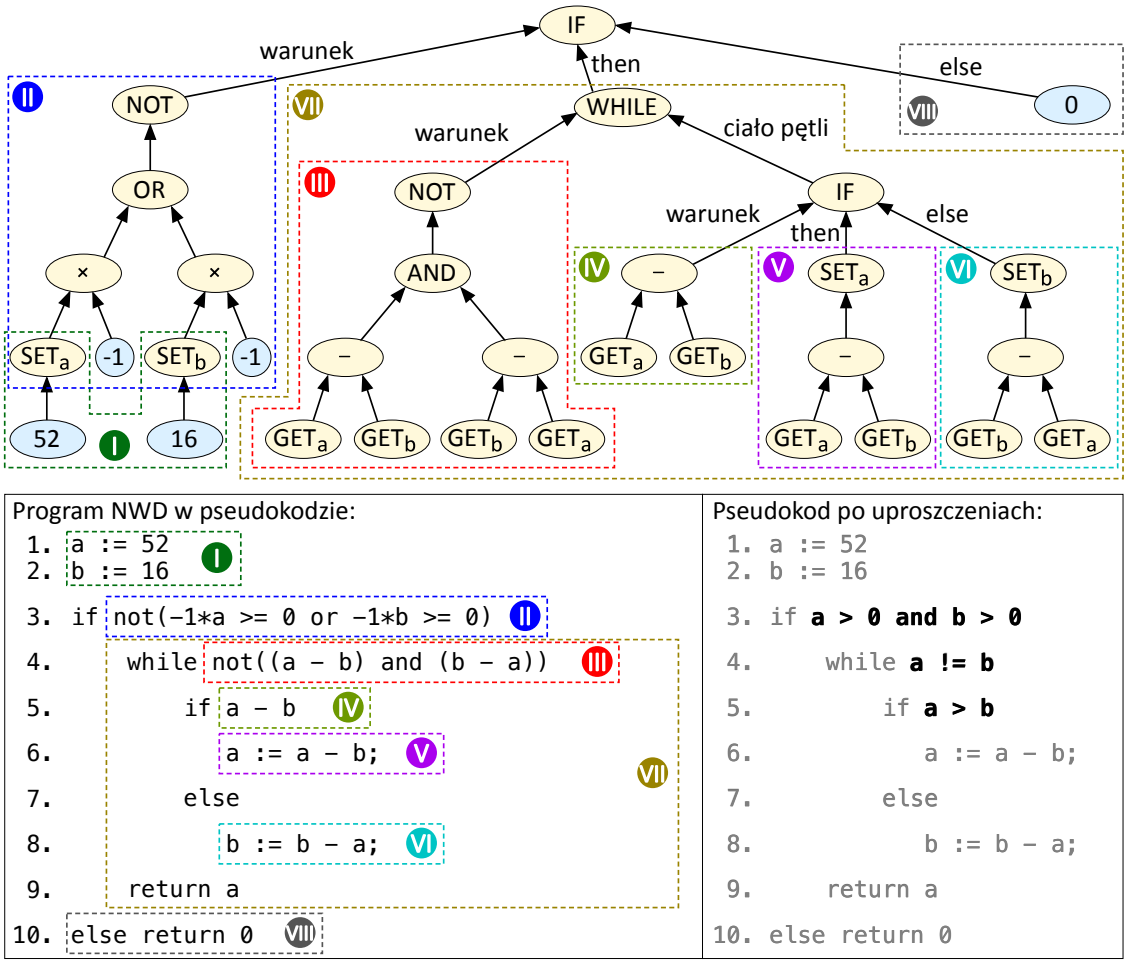
Na rysunku 5.9 zaprezentowano wykorzystanie procedury pętli *WHILE*, procedury warunkowej *IF* oraz operatorów logicznych i operatorów obsługi zmiennych do skonstruowania drzewa programu, które wyznacza największy wspólny dzielnik (NWD) zmiennych a i b. Numerami I-VIII oznaczono gałęzie drzewa, które odpowiadają wyrażeniom wyróżnionym w pseudokodzie programu NWD (rys. 5.9). Jeśli wartości zmiennych a i b są większe od zera (warunek w poddrzewie II na rys. 5.9), wykorzystywany jest algorytm Euklidesa (poddrzewo VII). W przeciwnym razie program zwraca wartość 0 (poddrzewo VIII).

Gałąź warunku pętli (poddrzewo III) jest równoważna warunkowi logicznemu $a \neq b$. Gdy warunek kontynuacji jest prawdziwy, wartość mniejszej zmiennej jest odejmowana od większej, a wynik działania zapisuje się w zmiennej reprezentującej odjemną (gałąź V lub VI w zależności od wartości zmiennych). Pętla kończy działanie, gdy $a = b$, a ich wartość jest największym wspólnym dzielnikiem liczb, które przypisano do zmiennych w gałęziach oznaczonych liczbą I. Dla drzewa na rysunku 5.9 wynikiem wykonania jest NWD(52,16) = 4.

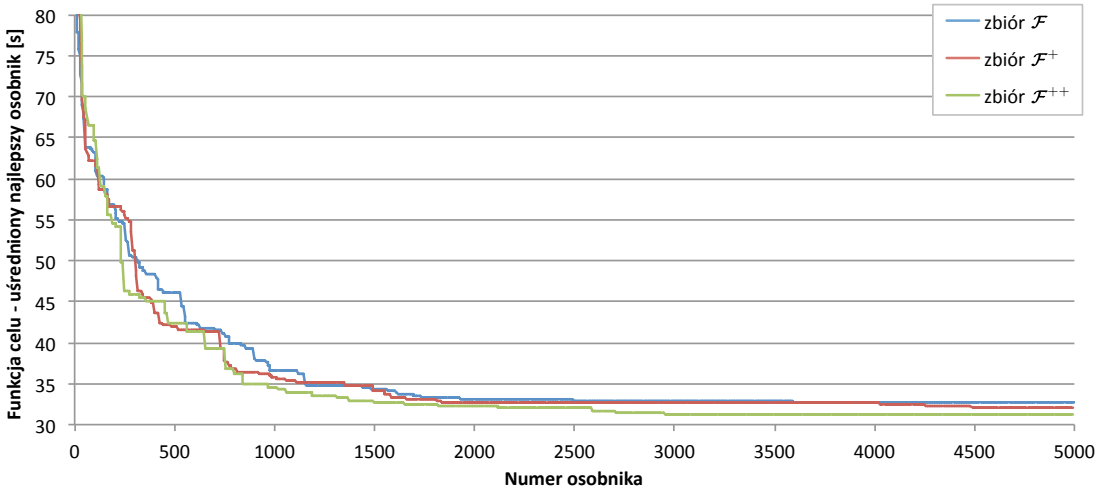
Zbadano trzeci zbiór procedur elementarnych rozszerzony o procedurę pętli i operatory do obsługi zmiennych: $\boldsymbol{\mathcal{F}}^{++} = \{+, -, *, /, IF, WHILE, OR, NOT, SET_{var_1}, \ldots, SET_{var_{D_{\mathrm{var}}}}, GET_{var_1}, \ldots, GET_{var_{D_{\mathrm{var}}}}\}$. Porównanie zbieżności z testowanymi wcześniej agentami przedstawiono na rysunku 5.10.

W eksperymencie przeprowadzonym z użyciem agenta $GP(F_n, \mu, \boldsymbol{\mathcal{F}}^{++})$ około 2% drzew nie mieściło się w limicie czasu wykonania $D_{\mathrm{MaxRuntime}} = 100$ ms. W zagadnieniu (5.37) osobnik reprezentowany jest przez 152 drzew, a populacja 50 osobników – przez 7600 drzew. Choć odsetek drzew, które nie kończą obliczeń w limicie czasu jest niewielki, to przy łącznej liczbie drzew w populacji średni czas przetwarzania pojedynczej epoki wzrasta do 19,7 s, co stanowi ponad 30-krotne pogorszenie w stosunku do agentów $GP(F_n, \mu, \boldsymbol{\mathcal{F}}^{+})$. Widoczne jest, że średnia zbieżność w zależności od liczby mutacji jest nieco lepsza niż dla testowanych wcześniej zbiorów. Niemniej jednak, najlepsza wartość funkcji celu, uzyskana przez grupę dziesięciu agentów $GP(F_n, \mu, \boldsymbol{\mathcal{F}}^{++})$ jest taka sama, jak dla agentów $GP(F_n, \mu, \boldsymbol{\mathcal{F}}^{+})$ i wynosi 30,05 s.

Agenty $GP(F_n, \mu, \boldsymbol{\mathcal{F}}^{++})$ cechują się satysfakcjonującą zbieżnością w zależności od liczby mutacji, jednak wydłużony czas wykonania powoduje, że przetworzenie 100 generacji wymaga ponad 30



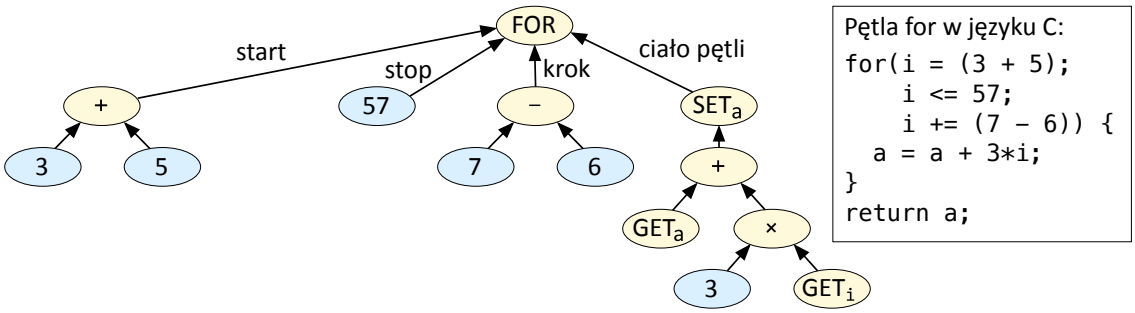
Rysunek 5.9: Drzewo programu wyznaczającego NWD i odpowiadający mu pseudokod
Źródło: opracowanie własne



Rysunek 5.10: Porównanie zbieżności programowania genetycznego dla trzech zbiorów procedur: $\mathcal{F}$, $\mathcal{F}^+$ oraz $\mathcal{F}^{++}$
Źródło: opracowanie własne

minut pracy programowania genetycznego. Agenty $GP(F_n, \mu, \boldsymbol{\mathcal{F}}^{+})$ zwracają rozwiązania zbliżonej jakości w czasie poniżej minuty. Agenty wykorzystujące zbiór procedur $\boldsymbol{\mathcal{F}}^{+}$ mogą prowadzić poszukiwanie rozwiązań z użyciem znacznie większej populacji i większej liczby epok niż agenty $GP(F_n, \mu, \boldsymbol{\mathcal{F}}^{++})$, dysponując takim samym czasem na obliczenia.

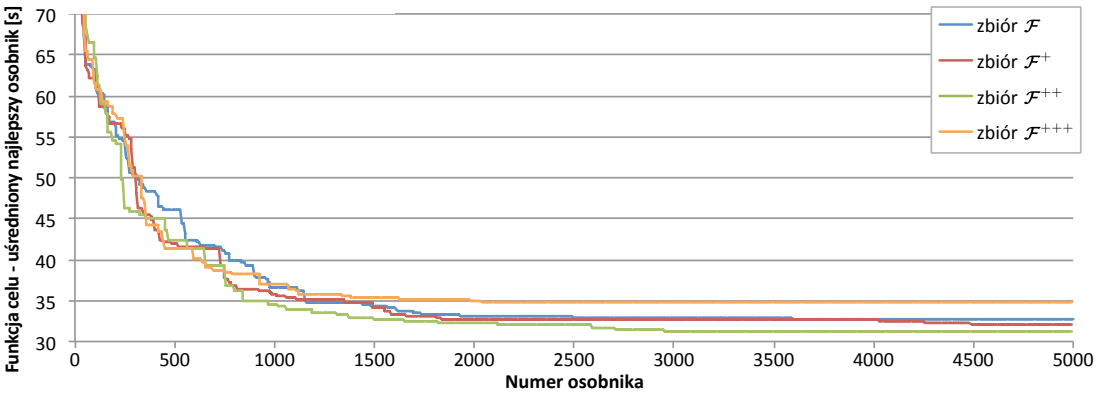
Rozważać można także procedurę pętli *FOR*, która przyjmuje 4 argumenty. Pierwszy argument określa początkową wartość licznika pętli, drugi – wartość końcową. Trzeci argument wyznacza krok, o jaki licznik pętli będzie modyfikowany w każdej iteracji. Ostatni, czwarty argument reprezentuje ciało pętli (rys. 5.11).



Rysunek 5.11: Przykładowa procedura *FOR* w programowaniu genetycznym
Źródło: opracowanie własne

Procedura *FOR* eksponuje w globalnej tablicy zmiennych wartość licznika pętli, która może być odczytywana w ciele pętli poprzez operator GET_i. W ten sposób wartość poddrzewa ciała pętli może zmieniać się z każdą iteracją. Wartością zwracaną przez procedurę *FOR* jest wartość ciała pętli z ostatniej iteracji. Podobnie jak dla pętli *WHILE* stosuje się limit iteracji pętli D_{MaxIt} oraz limit czasu wykonania drzewa $D_{\text{MaxRuntime}}$.

Przeprowadzono eksperyment, w którym procedurę pętli *WHILE* zastąpiono procedurą *FOR*, specyfikując zbiór procedur $\boldsymbol{\mathcal{F}}^{+++} = \{+, -, *, /, IF, FOR, OR, NOT, SET_{var_1}, \ldots, SET_{var_{D_{\text{var}}}}, GET_{var_1}, \ldots, GET_{var_{D_{\text{var}}}}\}$. Porównanie zbieżności z wcześniejszymi podejściami przedstawiono na rysunku 5.12.



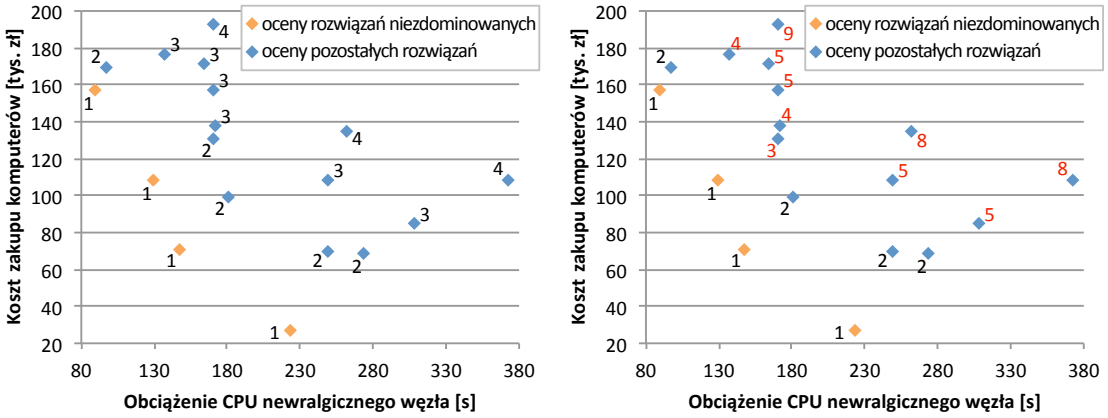
Rysunek 5.12: Porównanie zbieżności programowania genetycznego dla czterech zbiorów procedur: $\boldsymbol{\mathcal{F}}$, $\boldsymbol{\mathcal{F}}^{+}$, $\boldsymbol{\mathcal{F}}^{++}$ oraz $\boldsymbol{\mathcal{F}}^{+++}$
Źródło: opracowanie własne

Widoczne jest, że wprowadzenie pętli *FOR* niekorzystnie wpływa na zbieżność programowania genetycznego. Średni czas przetwarzania jednej epoki wyniósł 1013 ms. Na podstawie zebranych wyników rekomendować można zbiór procedur $\mathcal{F}^+$ jako oferujący najszybszą zbieżność w stosunku do czasu działania programowania genetycznego.

5.2 Wielokryterialne warianty programowania genetycznego

Kluczowy dylemat w wielokryterialnym programowaniu genetycznym polega na transformacji zagadnienia polioptymalizacji w funkcję sprawności. W algorytmach ewolucyjnych rozwiązano go za pomocą procedury nadawania rang w odniesieniu do alternatyw dopuszczalnych. Na rysunku 5.13 przedstawiono rangi dla przykładowych strategii zespołu agentów warstwy pośredniczącej w gridzie. Wykresy obrazują przestrzeń ocen dla dwukryterialnego problemu minimalizacji obciążenia CPU newralgicznego węzła oraz minimalizacji kosztu zakupu komputerów. Wybrane kryteria są w konflikcie: droższe komputery cechują się zazwyczaj wyższą wydajnością, co przekłada się na mniejsze obciążenie wyrażane w sekundach czasu pracy procesora. Poprawa względem jednego kryterium powoduje pogorszenie oceny względem drugiego, co oznacza, że dla rozpatrywanego problemu istnieje zbiór rozwiązań *Pareto*-optymalnych.

Rangi służące do obliczenia wartości *fitness* wyznaczane są zazwyczaj za pomocą procedury sortowania ocen wg relacji dominowania, którą opracował *Goldberg* [106]. Rozwiązania niezdominowane w populacji otrzymują rangę o wartości 1, po czym są tymczasowo usuwane z rozpatrywanego zbioru. Rozwiązania niezdominowane w okrojonej populacji otrzymują rangi większe o jeden od przydzielonych w poprzednim kroku, po czym również są usuwane. Operacja jest powtarzana do momentu, gdy rangi zostaną przypisane wszystkim osobnikom [106]. Procedura *Goldberga* wyznacza zatem rangi w oparciu o liczbę poziomów niezdominowania (rys. 5.13a).



(a) W oparciu o liczbę poziomów niezdominowania (procedura *Goldberga*)

(b) W oparciu o liczbę ocen dominujących (procedura *Fonseci-Flemminga*)

Rysunek 5.13: Rangi wyznaczone za pomocą wybranych procedur
Źródło: opracowanie własne

W alternatywnej procedurze *Fonseci-Fleminga* ranga osobnika jest równa liczbie rozwiązań, które go dominują (rys. 5.13b) [82]. W tym podejściu zakres rang przypisywanych osobnikom jest zazwyczaj większy niż przy wykorzystaniu procedury *Goldberga*, co przekłada się na większą presję selekcyjną. Na rys. 5.13b kolorem czerwonym oznaczono rangi, które różnią się wartościami dla tych samych strategii ze względu na zastosowaną procedurę. Niezależnie od wybranej procedury, wartości *fitness* wyznaczane na podstawie rang będą takie same dla strategii o równych rangach. Warto podkreślić, że rangi wyznaczane są wyłącznie dla rozwiązań dopuszczalnych. Dla pozostałych osobników obliczane są wartości funkcji kary, które stanowią podstawę do wyznaczenia sprawności rozwiązań.

W oparciu o rangi r wyznaczone dla rozwiązań dopuszczalnych i wartości kar dla pozostałych osobników, wyliczane są sprawności strategii sieci agentów zgodnie z następującą formułą:

$$\mathit{fitness}(x) = \begin{cases} -r(x) + r^{\max} + P^{\max} + 1, & \text{dla } x \in \widehat{\boldsymbol{\mathcal{X}}}, \\ -P(x) + P^{\max}, & \text{dla } x \notin \widehat{\boldsymbol{\mathcal{X}}}. \end{cases} \tag{5.40}$$

gdzie $r^{\max}$ – maksymalna wartość rangi w populacji rozwiązań.

Dla tak sformułowanej funkcji *fitness* przeprowadzono eksperyment, w którym badano przebieg eksploracji przestrzeni rozwiązań dopuszczalnych w instancji dwukryterialnego zagadnienia optymalizacji, które sformułowano, jak niżej:

Dla danych wejściowych: $[\mu_1, \ldots, \mu_{19}]$, $\{z_1, z_2\}$, $M = 2$, $L(z_1) = 128000$, $L(z_2) = 256000$, $P_R(z_1) = P_R(z_2) = 1$, $P_W(z_1) = P_W(z_2) = 4$, $P_T(z_1) = P_T(z_2) = 5$ s, $V_W = 4$, $I = 6$, $V = 12$, $J = 12$, ξ (koszty i pozostałe parametry komputerów opisano w tabeli nr 3.1), $T = [t_{vj}]_{V \times J}$ zdefiniowane jak w (4.1) oraz $\tau = [\tau_{vu}]_{V \times V}$ zdefiniowane jak w (4.2), należy wyznaczyć zbiór $\boldsymbol{\mathcal{X}}^{\mathrm{ND}}_{\leqslant}$ *Pareto*-optymalnych strategii sieci agentów warstwy pośredniczącej dla problemu optymalizacji wielokryterialnej zadanego uporządkowaną trójką:

$$(\widehat{\boldsymbol{\mathcal{X}}}, F, \rho^{\leqslant}), \tag{5.41}$$

1) $\widehat{\boldsymbol{\mathcal{X}}}$ – zbiór rozwiązań dopuszczalnych zdefiniowany, jak w zadaniu (4.66) przy ograniczeniu na liczbę dostępnych komputerów poszczególnych typów ($\mu_{18} = 1$);

2) F – kryterium wektorowe:

$$F : \boldsymbol{\mathcal{X}} \to \mathbb{R}^2,$$

przy czym $F(x) = [\hat{Z}_{\max}(x), \Xi(x)]^T = y \in \mathbb{R}^2$, dla $x \in \widehat{\boldsymbol{\mathcal{X}}}$,
gdzie:
$\hat{Z}_{\max}(x)$ – obciążenie CPU newralgicznego węzła [s],
$\Xi(x)$ – koszt zakupu komputerów [zł];

3) $\rho^{\leqslant}$ – relacja dominowania w przestrzeni $\mathbb{R}^2$:

$$\rho^{\leqslant} = \left\{(a, b) \in \boldsymbol{\mathcal{Y}} \times \boldsymbol{\mathcal{Y}} : a_n \leqslant b_n \text{ dla } n = \overline{1,2}\right\}.$$

Problem (5.41) cechuje się 152 binarnymi oraz 20 całkowitoliczbowymi zmiennymi decyzyjnymi. Binarna przestrzeń przeszukiwań zawiera $5{,}7 \cdot 10^{45}$ elementów, a całkowitoliczbowa – 10^{17}. Oceny populacji z różnych etapów programowania genetycznego przedstawiono na rys. 5.14. Populacja początkowa składająca się z losowo wygenerowanych osobników przeszukuje rozległy obszar przestrzeni kryterialnej (rys. 5.14a). W kolejnych epokach presja selekcyjna nakierowuje populację na eksplorację frontu *Pareto*. Osobniki gromadzą się w pobliżu rozwiązań niezdominowanych, a odległe obszary przestrzeni kryterialnej nie są badane (rys. 5.14b-5.14d). Zaobserwowany przebieg programowania genetycznego potwierdza, że selekcja oparta o funkcję *fitness* (5.40) wytwarza wystarczającą presję, aby właściwie ukierunkować poszukiwanie.

Większa presja selekcyjna przekłada się na szybsze przesunięcie ocen populacji w sąsiedztwo frontu *Pareto*. Z drugiej strony może doprowadzić do przedwczesnej stagnacji wyszukiwania w wyniku zmniejszenia różnorodności rozwiązań w populacji. Mniejsza presja ułatwia utrzymanie zróżnicowanych osobników w populacji, ale zwiększa też prawdopodobieństwo utraty rozwiązań niezdominowanych podczas wyboru osobników do kolejnej epoki. Poziom presji wynika z przyjętej procedury nadawania rang oraz wybranego operatora selekcji i jego parametrów, np. rozmiaru turnieju w selekcji turniejowej.

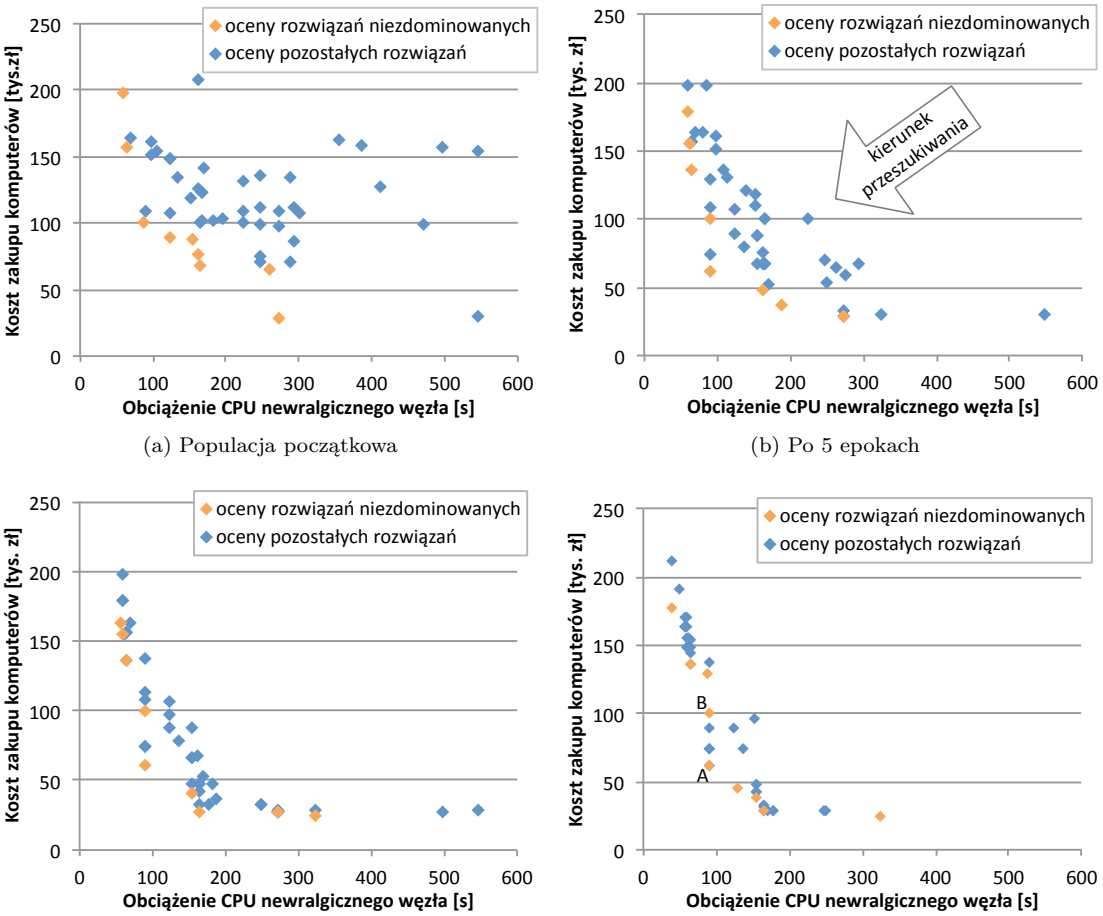
Na rysunku 5.14d zobrazowano możliwe kompromisy wśród rozwiązań wyznaczonych po 15 epokach programowania genetycznego. Warto zauważyć, że zwiększenie budżetu na zakup komputerów nie zawsze przekłada się na istotne obniżenie obciążenia. W szczególności oceny rozwiązań oznaczone jako A i B cechują się współrzędnymi: $F(x_A) = [90{,}17\text{ s};\ 61{,}69\text{ zł}]$ i $F(x_B) = [89{,}17\text{ s};\ 100{,}65\text{ zł}]$, co oznacza, że zwiększenie kosztów o blisko 40 tys. zł przekłada się na jednosekundową poprawę w odniesieniu do kryterium obciążenia newralgicznego węzła.

Zbieżność metody przedstawiono na rysunku 5.15, na którym pokazano oceny rozwiązań niezdominowanych na różnych etapach programowania genetycznego. Kolorem niebieski oznaczono oceny uzyskane dla populacji początkowej. Pomarańczowy kolor reprezentuje wyniki po 10 epokach ewolucji. Widoczne jest znaczące przesunięcie ocen rozwiązań niezdominowanych w stosunku do ocen osobników początkowych. W kolejnych epokach następuje dostrajanie odszukanych strategii – front rozwiązań przesuwa się w wolniejszym tempie. Kolor czerwony reprezentuje oceny dla końcowych strategii po zrealizowaniu 100 epok ewolucji.

Po zakończeniu programowania genetycznego skrajna najdroższa strategia uzyskała ocenę $F(x) = [33{,}28\text{ s};\ 173453\text{ zł}]$, przy czym:

$$X^z = [1, 3],\ X^\alpha = [6, 5, 1, 4, 3, 2, 2, 2, 4, 4, 3, 2],\ X^\beta = [10, 5, 2, 12, 11, 10]. \tag{5.42}$$

Zauważa się, że dla węzłów, w których pracują najbardziej wymagające agenty pod kątem obciążenia CPU, wybrano najbardziej wydajne typy komputerów ($j = \overline{10,12}$). Równocześnie mniej obciążone węzły otrzymują komputery tańsze, ale na tyle wydajne, aby nie przekroczyć czasu przetwarzania w newralgicznym węźle.



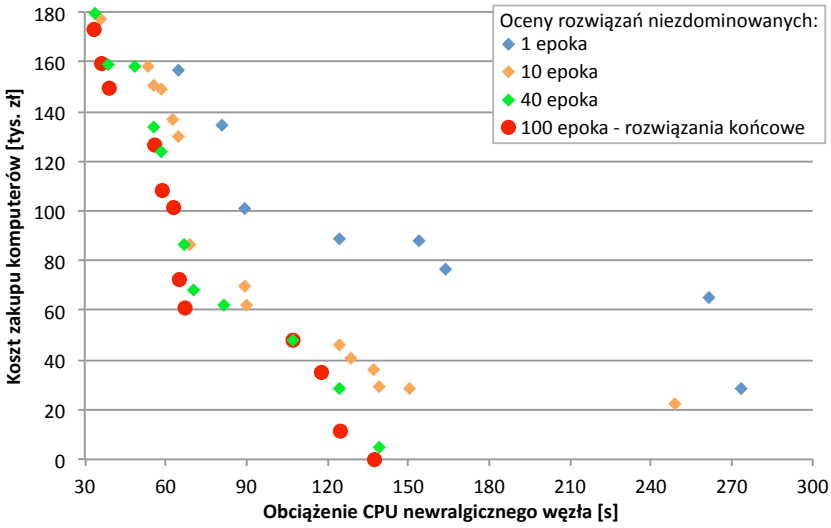
(a) Populacja początkowa

(b) Po 5 epokach

(c) Po 10 epokach

(d) Po 15 epokach

Rysunek 5.14: Oceny rozwiązań na różnych etapach programowania genetycznego
Źródło: opracowanie własne



Rysunek 5.15: Zbieżność programowania genetycznego podczas 100 epok
Źródło: opracowanie własne

Najtańsza strategia otrzymała ocenę $F(x) = [137{,}47\text{ s};\ 0\text{ zł}]$, przy czym:

$$X^z = [2,4],\ X^\alpha = [3,4,1,5,1,6,2,2,5,4,2,2],\ X^\beta = [2,2,1,1,2,2]. \tag{5.43}$$

W tym przypadku we wszystkich węzłach umieszczono komputery typu β_1 lub β_2, które są na wyposażeniu gridu, a zatem nie wprowadzają dodatkowych kosztów zakupu. Równocześnie komputery te cechują się najniższą wydajnością spośród wszystkich dostępnych (tabela nr 3.1).

W rozwiązaniach pośrednich kompromis polega na wyborze droższych, lecz bardziej wydajnych komputerów do węzłów, w których pracują agenty wymagające pod względem wydajności procesorów. Pozostałe węzły otrzymują mniej wydajne, ale za to tańsze typy komputerów. Jedna ze strategii pośrednich uzyskała ocenę $F(x) = [64{,}50\text{ s};\ 72591\text{ zł}]$, przy czym:

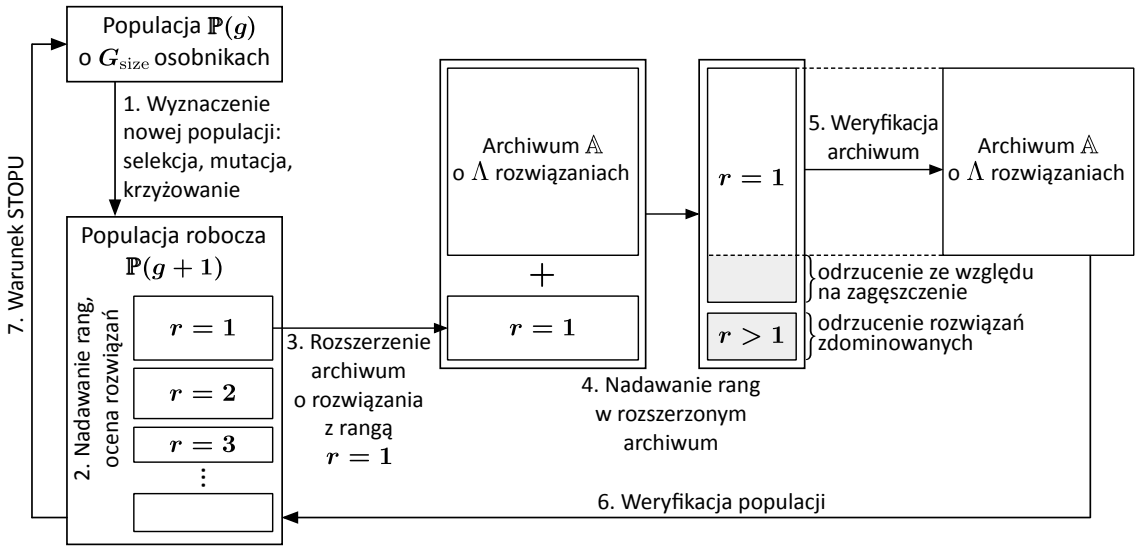
$$X^z = [3,3],\ X^\alpha = [3,5,1,4,3,4,4,4,5,6,1,6],\ X^\beta = [5,2,5,10,5,2]. \tag{5.44}$$

Agenty typu *W*, które powodują największe obciążenie CPU, pracują w węzłach o numerach: 1, 3, 4 i 5. W węźle ω_4 działają dodatkowo trzy agenty typu *S*. Dla tego węzła wybrano najbardziej wydajny komputer typu β_{10} ($\vartheta_{10} = 42246$ pkt *CPU Mark*, $\xi_{10} = 37338$ zł). Inne węzły, w których pracują agenty klasy *W*, otrzymały mniej wydajne, ale znacznie tańsze komputery typu β_5 o parametrach $\vartheta_5 = 23922$ pkt *CPU Mark* i $\xi_5 = 11751$ zł. Do pozostałych węzłów przydzielono komputery typu β_2 o najmniejszej wydajności $\vartheta_2 = 11324$ pkt *CPU Mark*, które jednak znajdują się już na wyposażeniu systemu ($\xi_2 = 0$ zł).

Wprawdzie wielokryterialne programowanie genetyczne opiera się na procedurze nadawania rang, to może się zdarzyć sytuacja utraty rozwiązań niezdominowanych z bieżącej populacji w wyniku utworzenia populacji potomnej. Aby uniknąć utraty najlepszych strategii, można zastosować archiwum zewnętrzne, w którym po przeprowadzeniu weryfikacji pozostawiane są rozwiązania wyznaczające alternatywy lokalnie niezdominowane (rys. 5.16).

Proponowane w rozprawie wielokryterialne programowanie genetyczne z archiwum MOGPA (ang. *Multiobjective Genetic Programming with Archive*) opiera się na wykorzystaniu archiwum o co najwyżej Λ rozwiązaniach. Po wyznaczeniu nowej populacji i nadaniu rang osobnikom (kroki 1 i 2 na rys. 5.16), rozwiązania z rangą $r = 1$ zostają wykorzystane do rozszerzenia archiwum (krok 3). Następnie procedura nadawania rang jest uruchamiana w rozszerzonym archiwum (krok 4), po czym usuwane są osobniki zdominowane. Jeśli liczba niezdominowanych rozwiązań jest większa od rozmiaru archiwum Λ, konieczne jest zredukowanie nadmiarowych osobników. Rozwiązania mogą być eliminowane w sposób losowy lub w oparciu o dodatkowe kryterium redukcji alternatyw, np. miarę zagęszczenia (krok 5). Rozwiązania zgromadzone w archiwum mogą posłużyć do zweryfikowania, czy w populacji roboczej na skutek selekcji nie utracono osobników niezdominowanych z wcześniejszych epok (krok 6). Procedura jest powtarzana do momentu spełnienia warunku stopu (krok 7).

Interesujący sposób redukcji nadmiarowych rozwiązań w archiwum opiera się na analizie ich zagęszczenia w przestrzeni kryterialnej. Im mniejsze zagęszczenie ocen innych strategii wokół oceny



Rysunek 5.16: Diagram wielokryterialnego programowania genetycznego z archiwum MOGPA
Źródło: opracowanie własne

rozważanego rozwiązania, tym większą ma ono szansę na zakwalifikowanie do archiwum. To podejście pozwala utrzymać w archiwum rozwiązania, które równomiernie pokrywają front *Pareto*, a dzięki temu prezentują szeroką gamę możliwych kompromisów pomiędzy kryteriami cząstkowymi.

W literaturze przedmiotu rozpatrywane są różne miary zagęszczenia. Algorytm NSGA-II wykorzystuje miarę CD (ang. *Crowding Distance*) opierającą się na odległości od sąsiednich rozwiązań w przestrzeni kryterialnej. Początkowa wartość zagęszczenia wynosi 0 dla wszystkich osobników. Następnie rozwiązania zostają uporządkowane według ocen dla pierwszego kryterium optymalizacji. Osobniki skrajne na posortowanej liście otrzymują wartość miary symbolizującą nieskończoność, dzięki czemu zakres frontu niezdominowanego nie zostanie zmniejszony w wyniku redukcji nadmiarowych rozwiązań. Dla każdego z pozostałych osobników miara zagęszczenia jest powiększana o różnicę ocen względem rozpatrywanego kryterium pomiędzy jego sąsiadami na posortowanej liście rozwiązań. Operacja jest powtarzana dla wszystkich kryteriów optymalizacji. Odrzucane są rozwiązania o najmniejszych wartościach miary CD.

Luo et al. [181] proponują *dynamiczną miarę zagęszczenia* DCD, która eliminuje dwa mankamenty opisanego wcześniej podejścia. Po pierwsze, uwzględniane są statystyczne wariancje oddalenia od sąsiednich osobników pod względem poszczególnych kryteriów, co redukuje ryzyko utraty rozwiązań różniących się istotnie od otoczenia względem tylko jednego kryterium. Po drugie, osobniki redukowane są pojedynczo, a po wyeliminowaniu każdego z nich miara DCD jest przeliczana na nowo. Zabezpiecza to przed usunięciem wszystkich ocen osobników z gęsto pokrytego fragmentu przestrzeni kryterialnej bez pozostawienia choćby jednej oceny, która będzie reprezentowała ten obszar [181].

Wadą miar CD i DCD jest to, że zagęszczenie jest wyznaczane wyłącznie na podstawie sąsiednich ocen rozwiązań bez uwzględnienia pozostałych ocen. Ponadto rozwiązaniom o takich samych ocenach mogą zostać przypisane różne wartości miary zagęszczenia. Dodatkowo miara DCD

może zwracać wyniki ujemne, co utrudnia interpretowanie uzyskiwanych wartości w sposób intuicyjny [209].

Ze względu na powyższe wady wprowadzona została *geometryczna miara zagęszczenia* GCD (ang. *Geometric Crowding Distance*) wyznaczana w znormalizowanej przestrzeni kryterialnej [209]. Oceny strategii ze *zbioru wzajemnie niezdominowanych rozwiązań konkurujących w celu kwalifikacji do archiwum* $\boldsymbol{\mathcal{X}}^{\mathrm{A}}_{\leqslant}$ normalizuje się, jak niżej:

$$\overline{F}_n(x) = \frac{F_n(x) - F_n^{\min}}{F_n^{\max} - F_n^{\min}},\ x \in \boldsymbol{\mathcal{X}}^{\mathrm{A}}_{\leqslant},\ n = \overline{1,N}, \tag{5.45}$$

gdzie:
$\overline{F}_n(x)$ – znormalizowana wartość oceny strategii x pod względem n-tego kryterium cząstkowego,
$F_n^{\max}$ – maksymalna wartość n-tego kryterium wśród kandydujących strategii,
$F_n^{\min}$ – minimalna wartość n-tego kryterium wśród kandydujących strategii.

Miara GCD dla strategii x wyznaczana jest jako suma odległości euklidesowych w znormalizowanej przestrzeni kryterialnej pomiędzy oceną rozpatrywanej strategii a ocenami pozostałych rozwiązań ze zbioru $\boldsymbol{\mathcal{X}}^{\mathrm{A}}_{\leqslant}$, jak niżej:

$$\mathrm{GCD}(x) = \sum_{\substack{x' \in \boldsymbol{\mathcal{X}}^{\mathrm{A}}_{\leqslant} \\ x' \neq x}} \sqrt{\sum_{n=1}^{N} \left(\overline{F}_n(x) - \overline{F}_n(x')\right)^2}. \tag{5.46}$$

Preferowane są rozwiązania o wyższych wartościach miary GCD. Geometryczna miara zagęszczenia obejmuje położenie względem wszystkich konkurujących rozwiązań. Ponadto odległość euklidesowa pozwala w równym stopniu respektować odchylenia względem jednego kryterium, jak i odchylenia pod względem kilku kryteriów. Dodatkowo nie jest w tym przypadku konieczne sortowanie ocen według poszczególnych kryteriów cząstkowych.

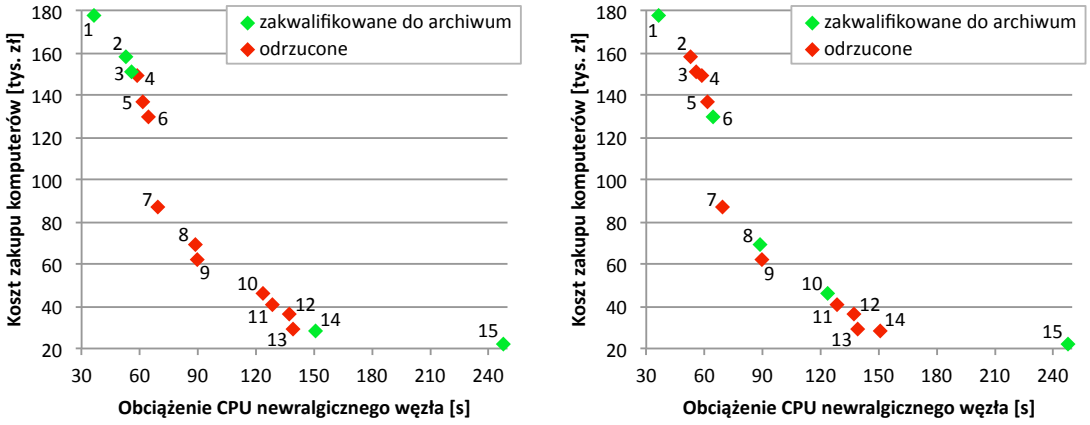
Wadą miary GCD jest jednak to, że rekomenduje ona odrzucenie rozwiązań, których oceny usytuowane są w rejonie średniej jakości ocen dla rozpatrywanego zbioru, nawet w sytuacji, gdy obszar ten nie jest najbardziej zagęszczony. W tabeli nr 5.1 rozpatruje się wybór $\Lambda = 5$ osobników spośród 15 kandydujących do uwzględnienia w archiwum. Osobniki uporządkowano według ocen dla kryterium obciążenia CPU newralgicznego węzła. Rozwiązania przechowywane w archiwum oraz osobniki odrzucone pokazano na rysunku 5.17a.

W oparciu o metrykę GCD odrzucono rozwiązania o numerach 4-13, które usytuowane są w środkowej części frontu ocen niezdominowanych – preferowane są rozwiązania na jego skrajach. Prowadzi to do nierównomiernej reprezentacji rozwiązań niezdominowanych w archiwum. Zaskakujące jest odrzucenie rozwiązania nr 7 w sytuacji, gdy zachowano rozwiązania nr 2 i 3, których oceny są bardzo blisko siebie w przestrzeni kryterialnej.

Tabela 5.1: Wartości miar zagęszczenia GCD i DGCD dla przykładowego zbioru strategii kandydujących do archiwum

Lp.	$\hat{Z}_{\max}$ [s]	Ξ [tys. zł]	GCD	DGCD										
				1	2	3	4	5	6	7	8	9	10	11
1	36,22	177,03	9,85 ✓	63,32	66,16	57,85	64,87	54,65	62,07	49,42	56,62	42,33	30,77	22,69
2	53,54	157,81	7,95 ✓	54,82	59,13	51,38	×	×	×	×	×	×	×	×
3	55,81	150,77	7,46 ✓	54,16	×	×	×	×	×	×	×	×	×	×
4	58,77	148,83	7,31 ×	55,30	60,07	52,11	56,62	47,25	×	×	×	×	×	×
5	62,33	136,73	6,77 ×	64,73	66,63	57,77	57,96	48,20	52,67	41,42	×	×	×	×
6	64,86	129,61	6,55 ×	75,73	75,90	65,86	63,73	52,96	54,69	42,92	54,17	42,31	30,01	26,84
7	69,33	86,43	6,04 ×	173,45	149,90	127,28	106,43	87,49	71,17	56,50	45,71	43,57	29,83	×
8	89,17	69,31	5,82 ×	134,03	116,37	100,89	86,85	74,23	60,07	51,05	38,61	45,02	32,47	37,93
9	90,17	61,69	5,96 ×	118,67	103,85	91,81	80,02	70,10	57,28	50,47	38,11	×	×	×
10	124,16	45,65	6,30 ×	71,42	63,42	62,57	55,89	57,55	48,80	56,21	43,82	41,55	45,80	36,78
11	129,00	40,73	6,50 ×	64,51	57,25	58,02	51,81	55,55	47,33	×	×	×	×	×
12	137,47	36,10	6,83 ×	60,93	54,12	57,03	51,09	×	×	×	×	×	×	×
13	139,07	29,27	7,23 ×	60,34	53,71	×	×	×	×	×	×	×	×	×
14	150,78	28,13	7,63 ✓	63,32	56,58	60,89	55,10	60,61	52,71	58,24	46,79	39,01	×	×
15	248,60	22,14	12,61 ✓	126,14	115,33	113,93	105,55	103,98	91,98	86,99	70,96	53,79	64,89	42,08

Źródło: opracowanie własne



(a) GCD (b) DGCD

Rysunek 5.17: Oceny rozwiązań zakwalifikowane do archiwum przy różnych miarach zagęszczenia
Źródło: opracowanie własne

Ze względu na powyższy mankament proponuje się *dynamiczną geometryczną miarę zagęszczenia* DGCD (ang. *dynamic geometric crowding distance*), której wartości wyznaczane są, jak niżej:

$$\mathrm{DGCD}(x) = \frac{\mathrm{GCD}(x)}{\mathrm{Var}(x)}, \quad x \in \boldsymbol{\mathcal{X}}^{\mathrm{A}}_{\leqslant}, \tag{5.47}$$

przy czym $\mathrm{Var}(x)$ oznacza wariancję odległości pomiędzy rozwiązaniem x a pozostałymi rozwiązaniami ze zbioru kandydatów $\boldsymbol{\mathcal{X}}^{\mathrm{A}}_{\leqslant}$, którą to wariancję wyznacza się według następującej zależności:

$$\mathrm{Var}(x) = \frac{\sum\limits_{\substack{x' \in \boldsymbol{\mathcal{X}}^{\mathrm{A}}_{\leqslant} \\ x' \neq x}} \left(\frac{\mathrm{GCD}(x)}{\left| \boldsymbol{\mathcal{X}}^{\mathrm{A}}_{\leqslant} \setminus \{x\} \right|} - \sqrt{\sum_{n=1}^{N} \left(\overline{F}_n(x) - \overline{F}_n(x') \right)^2} \right)^2}{\left| \boldsymbol{\mathcal{X}}^{\mathrm{A}}_{\leqslant} \setminus \{x\} \right|}, \; x \in \boldsymbol{\mathcal{X}}^{\mathrm{A}}_{\leqslant}, \tag{5.48}$$

gdzie $\boldsymbol{\mathcal{X}}^{\mathrm{A}}_{\leqslant} \setminus \{x\}$ – zbiór rozwiązań kandydujących pomniejszony o rozwiązanie x.

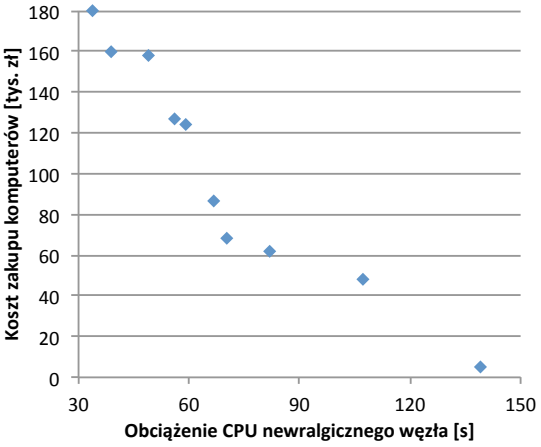
Rozwiązania o równomiernym oddaleniu od pozostałych kandydatów cechują się niższymi wartościami wariancji $\mathrm{Var}(x)$, co przekłada się na wyższe wartości $\mathrm{DGCD}(x)$. Miara DGCD reprezentuje zatem kompromis pomiędzy odległością od innych rozwiązań oraz równomiernym rozmieszczeniem kandydatów w przestrzeni kryterialnej. Dodatkowo osobniki eliminowane są pojedynczo ze zbioru $\boldsymbol{\mathcal{X}}^{\mathrm{A}}_{\leqslant}$, a po usunięciu każdego z nich wartości miary są przeliczane na nowo.

Wybór osobników do archiwum na podstawie miary DGCD zobrazowano w tabeli nr 5.1. Kolumny oznaczone liczbami 1-11 zawierają wartości miary wyznaczone po usunięciu kolejnych osobników ze zbioru kandydatów. Rozwiązania były odrzucane w następującej kolejności: 3, 13, 2, 12, 4, 11, 5, 9, 14, 7. Natomiast do archiwum zakwalifikowano strategie o numerach: 1, 6, 8, 10 i 15. Efekt końcowy eliminacji nadmiarowych rozwiązań zaprezentowano na rysunku 5.17b. Widoczne jest, że wybrane oceny strategii równomiernie pokrywają front rozwiązań niezdominowanych.

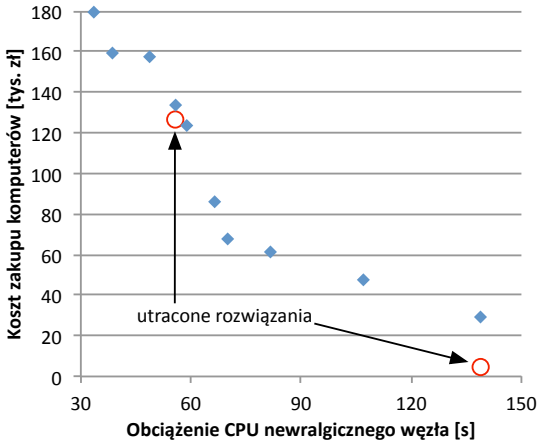
Sposób wykorzystania archiwum może być bierny, gdy archiwum jest jedynie pamięcią lokalnie niezdominowanych rozwiązań. Archiwum można wykorzystać także w sposób aktywny, weryfikując bieżącą populację. Spośród Λ strategii z archiwum, można wylosować λ_{size} osobników, które zostaną dołączone do bieżącej populacji. Pozwala to na przywrócenie rozwiązań niezdominowanych, które mogły zostać utracone na etapie selekcji. Na rysunku 5.18a przedstawiono oceny strategii niezdominowanych w g-tej generacji osobników. W kolejnej epoce utracono dwa rozwiązania niezdominowane ze względu na niską presję selekcyjną (rys. 5.18b). Przywrócenie utraconych osobników następuje na etapie weryfikacji populacji w epoce $g + \lambda_{\mathrm{epoch}}$ (rys. 5.18c).

Osobniki wybrane z archiwum mogą również wyeliminować z bieżącej populacji rozwiązania o najniższej sprawności. Intensywność weryfikacji określana jest przez parametr λ_{size} wskazujący liczbę osobników, które są przenoszone z archiwum do bieżącej populacji. Dla $\lambda_{\mathrm{size}} = 0$ weryfikacja nie jest wykonywana, natomiast dla $\lambda_{\mathrm{size}} = \Lambda$ wszystkie rozwiązania z archiwum są dołączane do populacji. Weryfikacja przeprowadzana jest co λ_{epoch} epok. Parametry Λ, λ_{size} i λ_{epoch} ustawiane są przez interesariusza w interfejsie aplikacji AAG'16.

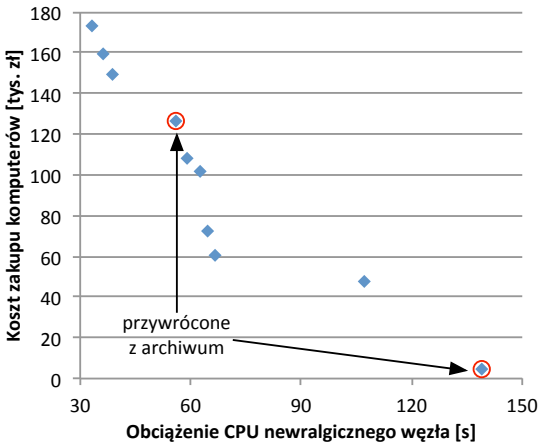
Archiwum może zostać również wykorzystane do utrzymywania różnorodności w populacji. W tym przypadku na etapie weryfikacji konstruowany jest zbiór obejmujący rozwiązania niezdominowane bieżącej populacji i osobniki z archiwum. Następnie stosowana jest miara DGCD dla wyznaczenia zagęszczenia osobników w obrębie uzyskanego zbioru. Do kolejnej epoki wybierany jest podzbiór osobników o najwyższych wartościach miary, które równomiernie pokrywają front niezdominowanych ocen. Pozwala to ukierunkować wyszukiwanie na cały front, a nie tylko na minima lokalne bieżącej populacji, co pokazano na rysunku 5.19.



(a) Oceny niezdominowane w g-tej generacji



(b) Generacja $g+1$: utrata rozwiązań na etapie selekcji

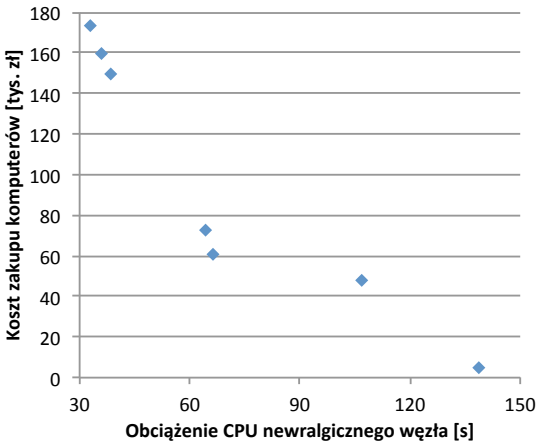


(c) Weryfikacja populacji w epoce $g + \lambda_{\text{epoch}}$: przywrócenie rozwiązań z archiwum

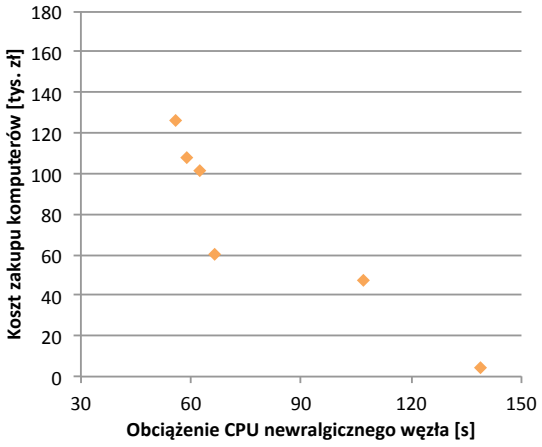
Rysunek 5.18: Utrata rozwiązań niezdominowanych i ich przywrócenie z archiwum
Źródło: opracowanie własne

Metaheurystyka MOGPA (rys. 5.20) może występować w wielu wariantach w zależności od metody nadawania rang (procedura *Goldberga* lub *Fonseci-Flemminga*), sposobu redukcji archiwum rozszerzonego (selekcja losowa lub w oparciu o miarę zagęszczenia), a także sposobu wykorzystania archiwum i jego parametrów. Algorytm MOGPA/G/R wykorzystuje procedurę *Goldberga*, a redukcja nadmiarowych rozwiązań w archiwum odbywa się losowo. Z kolei w algorytmie MOGPA/G/DGCD wykorzystuje się miarę zagęszczenia DGCD. Algorytm MOGPA/FF/R cechuje się zastosowaniem procedury *Fonseci-Flemminga*. Natomiast w algorytmie MOGPA/FF/DGCD stosuje się dodatkowo miarę DGCD. Każdy z wymienionych algorytmów może być realizowany przy różnych wartościach parametrów Λ, λ_{size} i λ_{epoch}.

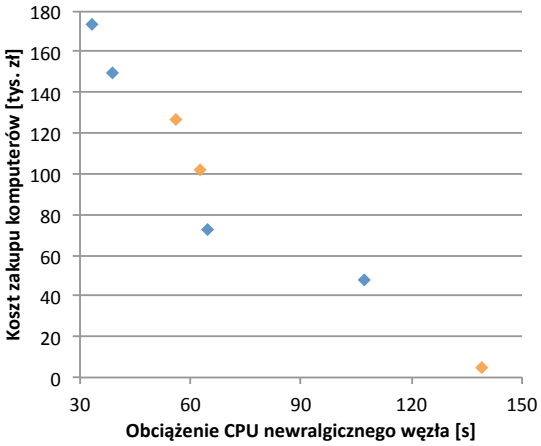
Złożoność obliczeniowa metaheurystyki klasy MOGPA dla problemu wyznaczania strategii sieci agentów warstwy pośredniczącej gridu zależy od doboru kryteriów optymalizacji. Spośród



(a) Rozwiązania niezdominowane bieżącej populacji



(b) Rozwiązania w archiwum



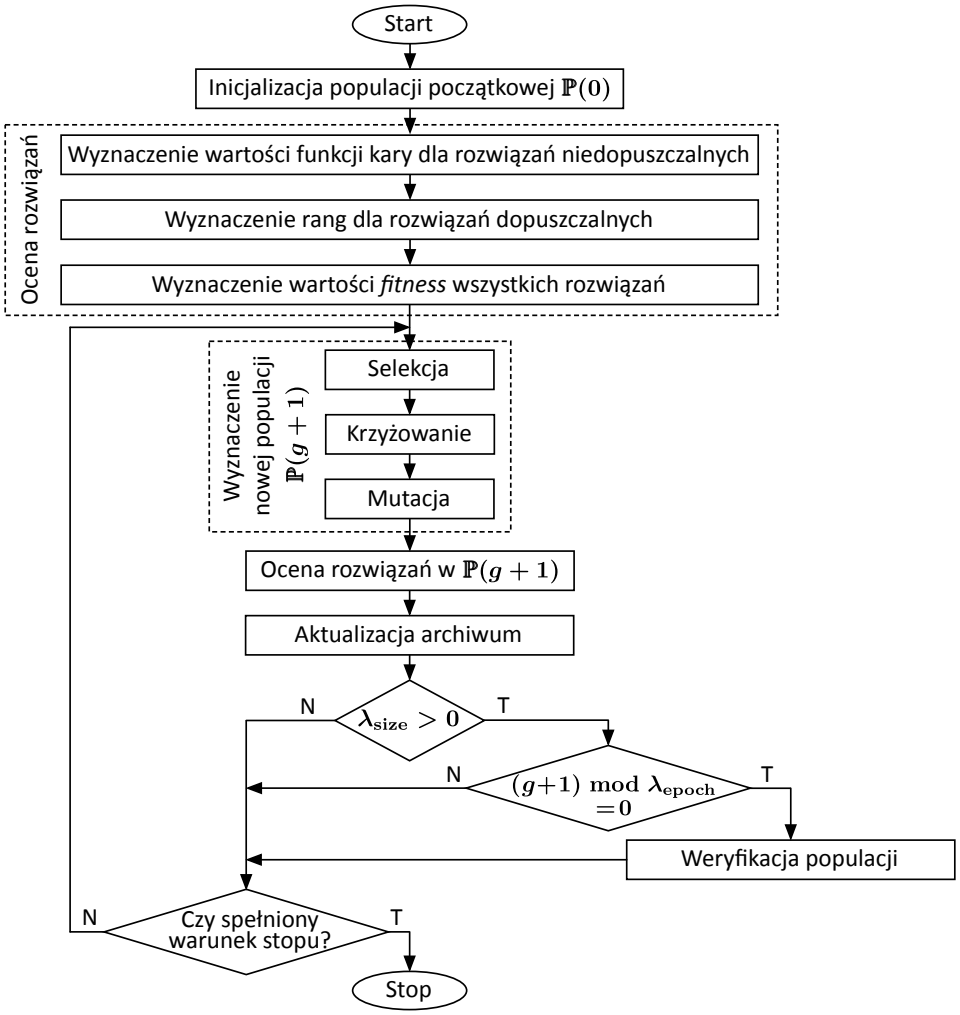
(c) Rozwiązania wybrane do kolejnej epoki

Rysunek 5.19: Wykorzystanie archiwum do utrzymania różnorodności w populacji
Źródło: opracowanie własne

kryteriów cząstkowych najbardziej kosztowne obliczeniowo jest wyznaczenie stopnia rozproszenia agentów $\eta(x)$ oraz łącznego obciążenia komunikacyjnego $\widetilde{Z}_{\text{suma}}(x)$ – dla obydwu kryteriów złożoność wynosi $O(n^4)$, przy czym $n = \max\{I, V, J\}$. Z kolei wyznaczenie łącznego obciążenia procesorów węzłów gridu cechuje się złożonością $O(n^3)$, $n = \max\{I, V, J\}$. W przypadku $G_{\max}$ epok i populacji rozmiaru G_{size} osobników, wartości ocen dla wybranych kryteriów optymalizacji wyznaczane są $G_{\max}G_{\text{size}}$ razy. Zatem złożoność metaheurystyki MOGPA dla rozważanego problemu wynosi $O(n^6)$, gdzie $n = \max\{I, V, J, G_{\max}, G_{\text{size}}\}$.

W oparciu o zaproponowane metody polioptymalizacji strategii agentów warstwy pośredniczącej gridu, opracowano *agenty-solwery* wyznaczające reprezentacje *Pareto*-optymalnych rozwiązań w zagadnieniu $(\widehat{\boldsymbol{\mathcal{X}}}, F, \rho^{\leqslant})$. Cechą charakterystyczną *agentów-solwerów* jest wykorzystanie wielokryterialnego programowania genetycznego MOGPA/xx/yy, przy czym xx oznacza procedurę nadawania rang, a yy – sposób redukcji przepełnionego archiwum. Agenty te będziemy oznaczali jako $MGP(F, \mu, \rho^{\leqslant}, \boldsymbol{\mathcal{F}}, \mathbb{A}, xx, yy)$, gdzie $\mathbb{A}$ jest symbolem archiwum. Różne instancje agentów klasy MGP

mogą różnić się parametrami archiwum (Λ, λ_{size} i λ_{epoch}), rozmiarem populacji G_{size} i liczbą epok programowania genetycznego $G_{\max}$.



Rysunek 5.20: Schemat wielokryterialnej metaheurystyki programowania genetycznego MOGPA
Źródło: opracowanie własne

5.3 Kompromisowe wersje programowania genetycznego

W rozważanych zagadnieniach optymalizacji wielokryterialnej zbiór rozwiązań *Pareto*-optymalnych obejmuje szeroką gamę strategii, które cechują się różnymi kompromisami pomiędzy kryteriami cząstkowymi. Konieczne jest jednak wybranie jednego rozwiązania, które zostanie wdrożone w systemie. W przypadku optymalizacji kompromisowej wybór odbywa się w oparciu o odległości *ocen kandydujących rozwiązań* $y(x)$ (dla $x \in \widehat{\boldsymbol{\mathcal{X}}}$) od *punktu idealnego* y^{inf} w przestrzeni kryterialnej [197]. Pożądane są oceny najbliższe y^{inf} pod względem przyjętej metryki.

Dla zagadnień minimalizacji kryteriów cząstkowych współrzędne punktu idealnego odpowiadają kresom dolnym (łac. *infimum*) wartości poszczególnych kryteriów w zbiorze rozwiązań dopuszczalnych. Jeśli interesariusz wybierze kryterium cząstkowe podlegające maksymalizacji, to maksymalizacja może zostać zastąpiona minimalizacją funkcji kryterialnej ze znakiem przeciwnym. Bez utraty ogólności punkt idealny $y^{\inf}$ w zagadnieniu N-kryterialnym można zatem zdefiniować, jak niżej [13]:

$$y^{\inf} = \left[y_1^{\inf}, \ldots, y_n^{\inf}, \ldots, y_N^{\inf}\right]^T, \tag{5.49}$$

przy czym:

$$y_n^{\inf} = \inf_{x \in \widehat{\boldsymbol{x}}} y_n(x), n = \overline{1, N}. \tag{5.50}$$

Jeśli punkt idealny $y^{\inf}$ należy do zbioru ocen strategii dopuszczalnych $\boldsymbol{\mathcal{Y}}$, to jest oceną dominującą w sensie relacji dominowania $\rho^{\leqslant}$ [14]. Oznacza to, że wszystkie inne oceny w zbiorze $\boldsymbol{\mathcal{Y}}$ są zdominowane przez ocenę $y^{\inf}$, a więc nie występuje zbiór rozwiązań *Pareto*-optymalnych. Istnieje natomiast przynajmniej jedno rozwiązanie o ocenie $y^{\inf}$, a w szczególnych przypadkach może występować zbiór rozwiązań cechujących się oceną dominującą [13].

Obecność oceny $y^{\inf}$ w zbiorze $\boldsymbol{\mathcal{Y}}$ jest pożądana, gdyż eliminuje konieczność wyboru kompromisu spośród ocen *Pareto*-optymalnych. Jednak w wielu zagadnieniach osiągnięcie oceny $y^{\inf}$ jest niemożliwe przy zachowaniu wymagań formalnych i wymagań nałożonych przez interesariuszy [22]. W szczególności w sytuacjach konfliktowych zbiór rozwiązań dopuszczalnych zazwyczaj nie zawiera strategii dominującej. W takim wypadku wybierana jest ocena najbliższa punktowi idealnemu w przestrzeni kryterialnej.

Współrzędne oceny $y^{\inf}$ można oszacować teoretycznie lub za pomocą wybranej metaheurystyki. W przypadku kryterium $\hat{Z}_{\max}$ czas pracy newralgicznego węzła nie może być krótszy niż czas wykonania najbardziej wymagającego obliczeniowo agenta w węźle, do którego przypisano najbardziej wydajny komputer [209]:

$$y_1^{\inf} = \min_{j=\overline{1,J}} \max_{v=\overline{1,V}} \{t_{vj}\}. \tag{5.51}$$

Kryterium czasu komunikacji newralgicznego węzła $\widetilde{Z}_{\max}$ osiąga wartość zero, gdy wszystkie agenty pracują w obrębie jednego węzła, a narzuty komunikacyjne nie występują:

$$y_2^{\inf} = 0. \tag{5.52}$$

Łączne obciążenie procesorów gridu $\hat{Z}_{\text{suma}}$ nie może być mniejsze niż czas działania wszystkich agentów na komputerach o największej wydajności:

$$y_3^{\inf} = \sum_{v=1}^{V} \min_{j=\overline{1,J}} \{t_{vj}\}. \tag{5.53}$$

Skoro kres dolny dla kryterium $\widetilde{Z}_{\max}$ wynosi $y_2^{\inf} = 0$, to kryterium łącznego obciążenia komunikacyjnego $\widetilde{Z}_{\text{suma}}$ również jest od dołu ograniczone wartością zero:

$$y_4^{\inf} = 0. \tag{5.54}$$

W przypadku optymalizacji łącznej wydajności gridu Θ następuje zamiana na kryterium ze znakiem przeciwnym $-\Theta$, którego kres dolny osiągany jest za pomocą strategii wykorzystującej najbardziej wydajne komputery we wszystkich węzłach:

$$y_5^{\inf} = -I \max_{j=\overline{1,J}}\{\vartheta_j\}. \tag{5.55}$$

Z kolei najniższy koszt zakupu komputerów Ξ występuje, gdy dla wszystkich węzłów wybrane zostaną najtańsze komputery:

$$y_6^{\inf} = I \min_{j=\overline{1,J}}\{\xi_j\}. \tag{5.56}$$

Oczekiwany czas realizacji zadań $\overline{T}$ nie może być krótszy niż oczekiwany czas przetwarzania paczek danych w węzłach obliczeniowych, przy założeniu że wszystkie zadania zostaną zrealizowane:

$$y_{17}^{\inf} = \bar{t} = \frac{\mathscr{T}_a + \mathscr{T}_u}{\mathscr{T}_a}\, \widehat{I}^{-1} \sum_{m=1}^{M} L(z_m) P_T(z_m). \tag{5.57}$$

Najniższy koszt K występuje przy minimalnym czasie realizacji zadań, wykorzystaniu najmniej kosztownych w eksploatacji komputerów, minimalnym czasie przetwarzania w węzłach warstwy pośredniczącej i wyeliminowaniu kosztu komunikacji, co skutkuje oszacowaniem, jak niżej:

$$y_7^{\inf} = y_3^{inf} \min_{i=\overline{1,I}} \min_{j=\overline{1,J}}\{\hat{c}_{i,j}^{\text{load}}\} + (y_{17}^{\inf} - y_3^{inf}) \min_{i=\overline{1,I}} \min_{j=\overline{1,J}}\{\hat{c}_{i,j}^{\text{idle}}\}. \tag{5.58}$$

Kres dolny zbioru wartości przyjmowanych przez kryterium $-\eta$ wyznaczany jest dla sytuacji maksymalnego rozproszenia agentów, co prowadzi do następującego oszacowania:

$$y_8^{\inf} = -\frac{V(V-1)}{2}. \tag{5.59}$$

Maksymalizowane kryteria odnoszące się do rezerw pamięci RAM, HDD i SSD w węzłach newralgicznych, zamieniane są na minimalizowane kryteria ze znakiem przeciwnym: $-\kappa_{\min}^{\text{RAM}}(x)$, $-\kappa_{\min}^{\text{HDD}}(x)$ oraz $-\kappa_{\min}^{\text{SSD}}(x)$. Maksymalne wielkości rezerw można uzyskać, gdy agenty o największych wymaganiach zostaną uruchomione na komputerach najlepiej wyposażonych pod względem poszczególnych typów pamięci. Zachodzi zatem:

$$y_9^{\inf} = \max_{v=\overline{1,V}}\{r_v\} - \max_{j=\overline{1,J}}\{ram_j - r_j^{\min}\}, \tag{5.60}$$

$$y_{10}^{\inf} = \max_{v=\overline{1,V}}\{h_v\} - \max_{j=\overline{1,J}}\{hdd_j - h_j^{\min}\}, \tag{5.61}$$

$$y_{11}^{\inf} = \max_{v=\overline{1,V}}\{\hat{h}_v\} - \max_{j=\overline{1,J}}\{ssd_j - \hat{h}_j^{\min}\}. \tag{5.62}$$

Kresy dolne zbiorów wartości dla kryteriów $-M^{\mathrm{RAM}}$, $-M^{\mathrm{HDD}}$ i $-M^{\mathrm{SSD}}$ odnoszących się do łącznych rezerw zasobów można oszacować dla strategii, w której do wszystkich węzłów przypisano komputery najlepiej wyposażone pod względem poszczególnych typów pamięci. W efekcie uzyskuje się następujące zależności:

$$y_{12}^{\inf} = -I \max_{j=\overline{1,J}}\{ram_j - r_j^{\min}\} + \sum_{v=1}^{V} r_v, \tag{5.63}$$

$$y_{13}^{\inf} = -I \max_{j=\overline{1,J}}\{hdd_j - h_j^{\min}\} + \sum_{v=1}^{V} h_v, \tag{5.64}$$

$$y_{14}^{\inf} = -I \max_{j=\overline{1,J}}\{ssd_j - \hat{h}_j^{\min}\} + \sum_{v=1}^{V} \hat{h}_v. \tag{5.65}$$

Minimalizacja łącznego poboru mocy elektrycznej E jest możliwa za pomocą wyboru najbardziej energooszczędnych komputerów, co umożliwia oszacowanie kresu dolnego:

$$y_{15}^{\inf} = I \min_{j=\overline{1,J}}\{\varepsilon_j^{\mathrm{load}}\}. \tag{5.66}$$

Oszacowanie dolne zbioru wartości dla kryterium $-\Gamma$ wyznaczane jest w oparciu o najbardziej niezawodne typy komputerów, przy minimalnym czasie realizacji scenariusza, jak niżej:

$$y_{16}^{\inf} = \prod_{i=1}^{I} \exp\left(-y_{17}^{\inf} \min_{j=\overline{1,J}}\{\gamma_j\}\right). \tag{5.67}$$

Oszacowania teoretyczne pozwalają na wyznaczenie punktów odniesienia w celu wyboru rozwiązania kompromisowego. W oszacowaniach nie uwzględnia się ograniczeń, jakie mogą zostać wprowadzone przez interesariusza przy formułowaniu zagadnienia optymalizacji. Przykładowo, wyznaczając oszacowanie dolne zbioru wartości przyjmowanych przez kryterium $\hat{Z}_{\mathrm{suma}}$ zakłada się, że agenty są uruchomione na komputerach o największej wydajności. Jeśli wprowadzone zostanie ograniczenie na koszt systemu, to zakup najbardziej wydajnych – a co za tym idzie najdroższych – komputerów w odpowiedniej liczbie może okazać się niemożliwy. Oznacza to, że rozwiązanie cechujące się oceną będącą kresem dolnym $y_3^{\inf}$ nie należy do zbioru dopuszczalnego $\widehat{\boldsymbol{\mathcal{X}}}$ w tak sformułowanym zagadnieniu z ograniczeniami.

Interesujący sposób oszacowania współrzędnych punktów charakterystycznych polega na wykorzystaniu agentów jednokryterialnych $GP(F_n, \mu, \boldsymbol{\mathcal{F}})$ w celu wyznaczenia rozwiązań suboptymalnych dla każdego z kryteriów cząstkowych, przy zachowaniu ograniczeń nałożonych przez interesariusza dla problemu wielokryterialnego. Agenty klasy $GP(F_n, \mu, \boldsymbol{\mathcal{F}})$ nie gwarantują wyznaczenia rozwiązań optymalnych. W efekcie oceny wyznaczonych rozwiązań są przybliżonymi współrzędnymi punktu $y^{\inf}$.

Drugim punktem charakterystycznym wykorzystywanym podczas wyboru rozwiązania kompromisowego jest *ocena antyidealna* $y^{\sup}$, która wskazuje, jakich wartości kryteriów należy unikać przy przeszukiwaniu przestrzeni rozwiązań. Współrzędne punktu $y^{\sup}$ stanowią kres górny (łac. *supremum*) zbioru ocen minimalizowanych kryteriów cząstkowych, jak niżej:

$$y_n^{\sup} = \sup_{x \in \widehat{\boldsymbol{x}}} F_n(x),\ n = \overline{1, N}. \tag{5.68}$$

Dla kryteriów cząstkowych: $\hat{Z}_{\max}$, $\widetilde{Z}_{\max}$, $\hat{Z}_{\text{suma}}$, $\widetilde{Z}_{\text{suma}}$, Ξ, K, E oraz $\overline{T}$ oszacowania górne wyprowadzono w zależnościach (5.3)-(5.10) w podrozdziale 5.1. Współrzędne punktu antyidealnego dla powyższych kryteriów definiuje się zatem następująco:

$$y_1^{\sup} = \hat{Z}_{\max}^{\max}, \tag{5.69}$$
$$y_2^{\sup} = \widetilde{Z}_{\max}^{\max}, \tag{5.70}$$
$$y_3^{\sup} = \hat{Z}_{\text{suma}}^{\max}, \tag{5.71}$$
$$y_4^{\sup} = \widetilde{Z}_{\text{suma}}^{\max}, \tag{5.72}$$
$$y_6^{\sup} = \Xi^{\max}, \tag{5.73}$$
$$y_7^{\sup} = K^{\max}, \tag{5.74}$$
$$y_{15}^{\sup} = E^{\max}, \tag{5.75}$$
$$y_{17}^{\sup} = \overline{T}^{\max}. \tag{5.76}$$

Dla kryterium $-\Theta$ oszacowanie górne wyznaczane jest dla strategii uwzględniającej najmniej wydajne typy komputerów, jak niżej:

$$y_5^{\sup} = -I \min_{j=\overline{1,J}} \{\vartheta_j\}. \tag{5.77}$$

Z kolei dla $-\Gamma$ kres górny zadany jest następującą zależnością:

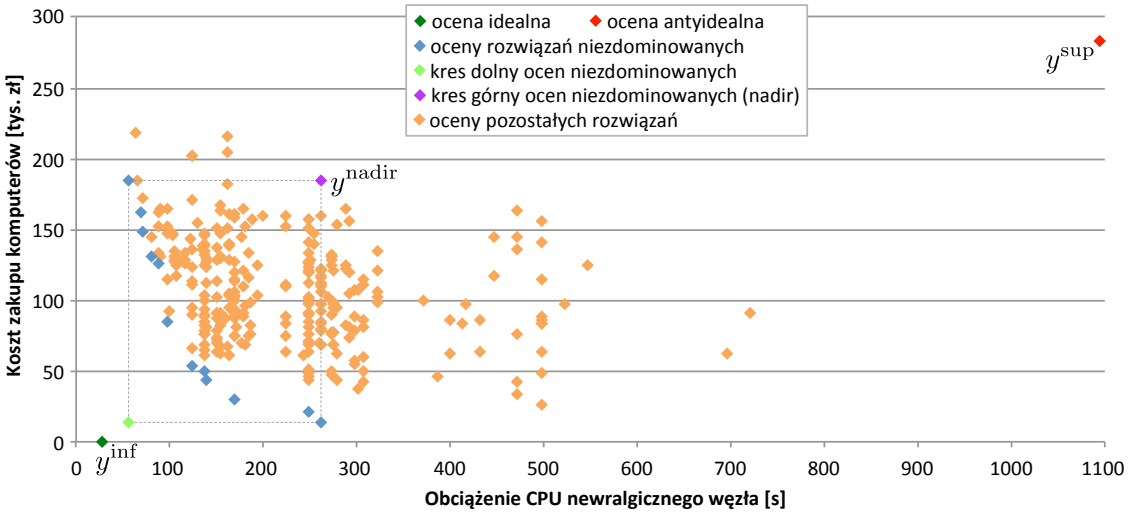
$$y_{16}^{\sup} = \prod_{i=1}^{I} \exp\left(-\overline{T}^{\max} \max_{j=\overline{1,J}} \{\gamma_j\}\right). \tag{5.78}$$

Dla kryteriów ze znakiem przeciwnym: $-\eta$, $-\kappa_{\min}^{\text{RAM}}(x)$, $-\kappa_{\min}^{\text{HDD}}(x)$, $-\kappa_{\min}^{\text{SSD}}(x)$, $-M^{\text{RAM}}$, $-M^{\text{HDD}}$ i $-M^{\text{SSD}}$, kres górny przyjmuje wartość zero:

$$y_8^{\sup} = y_9^{\sup} = y_{10}^{\sup} = y_{11}^{\sup} = y_{12}^{\sup} = y_{13}^{\sup} = y_{14}^{\sup} = 0. \tag{5.79}$$

Jeśli kryterium cząstkowe występuje również w roli ograniczenia w rozpatrywanym zagadnieniu optymalizacji, to kres górny przyjmuje wartość równą zadanemu ograniczeniu dla minimalizowanych kryteriów. Dla kryteriów ze znakiem przeciwnym odpowiadające im współrzędne punktu antyidealnego otrzymują wartości przeciwne do zdefiniowanych ograniczeń.

Na rysunku 5.21 zaprezentowano punkty charakterystyczne y^{inf} i y^{sup} na tle ocen populacji 300 osobników wygenerowanych losowo dla zagadnienia (5.41). Punkt y^{inf} nie należy do zbioru ocen rozwiązań dopuszczalnych $\boldsymbol{\mathcal{Y}}$, podobnie jak punkt y^{sup}. Najgorsza ocena w odniesieniu do kryterium $\hat{Z}_{\max}$ występuje, gdy wszystkie agenty działają na najmniej wydajnym komputerze. Z kolei osiągnięcie najgorszej oceny dla kryterium kosztu Ξ wymaga zakupu najdroższych komputerów, które zarazem oferują największą wydajność. Nie jest możliwe wyznaczenie strategii, która łączy obie te cechy. Ponadto nie istnieje strategia minimalizująca oceny dla obu kryteriów.



Rysunek 5.21: Punkty charakterystyczne w przestrzeni kryterialnej dla zagadnienia optymalizacji strategii sieci agentów w gridzie
Źródło: opracowanie własne

Jeśli nie są znane oszacowania teoretyczne, przybliżenia punktów y^{inf} i y^{sup} można wyznaczyć na podstawie ocen bieżącej populacji. Ocena idealna może być wyznaczona jako kres dolny zbioru rozwiązań niezdominowanych $\boldsymbol{\mathcal{X}}_{\leqslant}^{\text{ND}}$, a ocena antyidealna – jako kres górny, co zapisuje się następująco:

$$y_n^{\text{inf}} = \min_{x \in \boldsymbol{\mathcal{X}}_{\leqslant}^{\text{ND}}} F_n(x), \ n = \overline{1,N}, \tag{5.80}$$

$$y_n^{\text{sup}} = \max_{x \in \boldsymbol{\mathcal{X}}_{\leqslant}^{\text{ND}}} F_n(x), \ n = \overline{1,N}. \tag{5.81}$$

W rozpatrywanym przypadku ocena y^{sup} odpowiada punktowi *nadir* [14], który oznaczono na rys. 5.21 jako y^{nadir}. Za wadę omawianego podejścia należy uznać fakt, że współrzędne punktów referencyjnych zależą od przebiegu przeszukiwania przestrzeni rozwiązań. Jeśli pewna grupa rozwiązań sprawnych nie posiada reprezentacji w zbiorze $\boldsymbol{\mathcal{X}}_{\leqslant}^{\text{ND}}$ w wybranej epoce, to nie znajdzie też odzwierciedlenia we współrzędnych punktów y^{inf} i y^{sup}. Może to doprowadzić do wyłączenia z eksploracji pewnych obszarów przestrzeni rozwiązań.

Ważne podejście polega na wykorzystaniu w roli punktów $y^{\inf}$ i $y^{\sup}$ ocen zdefiniowanych przez interesariusza. Może on w ten sposób wyrazić preferencje odnośnie rozwiązań i ukierunkować poszukiwanie. Ponadto wybór punktów przez interesariusza może odbywać się w trakcie działania programowania genetycznego w oparciu o uzyskane rozwiązania niezdominowane. Prowadzi to do interaktywnych metod, w których metaheurystyka wyznacza rozwiązania na podstawie preferencji interesariusza, który dokonuje wyborów w oparciu o zbiór możliwych kompromisów.

Różne sposoby wyznaczania ocen $y^{\inf}$ i $y^{\sup}$ mogą prowadzić do innych współrzędnych punktów referencyjnych, co przekłada się na późniejszy wybór rozwiązania kompromisowego. W celu wyznaczenia odległości od punktu idealnego, oceny kandydujących rozwiązań podlegają normalizacji. Niech $\boldsymbol{\mathcal{N}}$ będzie zbiorem indeksów kryteriów optymalizacji wybranych przez interesariusza. Przykładowo dla zbioru $\boldsymbol{\mathcal{N}} = \{1, 6, 9\}$ rozpatrywane są kryteria: $F_1 = \hat{Z}_{\max}$, $F_6 = \Xi$ oraz kryterium ze zmienionym znakiem $F_9 = -\kappa_{\min}^{\mathrm{RAM}}$ (tabela nr 4.2), a zbiór $\boldsymbol{\mathcal{Y}} \subset \mathbb{R}^3$. Znormalizowane współrzędne oceny rozwiązania x w zagadnieniu optymalizacji kryteriów określonych zbiorem $\boldsymbol{\mathcal{N}}$, wyznacza się następująco:

$$\overline{y}_n(x) = \frac{y_n(x) - y_n^{\inf}}{y_n^{\sup} - y_n^{\inf}},\ y_n^{\sup} > y_n^{\inf},\ n \in \boldsymbol{\mathcal{N}}. \tag{5.82}$$

W znormalizowanej przestrzeni współrzędne puntu idealnego przyjmują wartość zero, a współrzędne punktu antyidealnego – wartość jeden, jak niżej:

$$\overline{y}_n^{\inf} = 0, n \in \boldsymbol{\mathcal{N}}, \tag{5.83}$$

$$\overline{y}_n^{\sup} = 1, n \in \boldsymbol{\mathcal{N}}. \tag{5.84}$$

Rozwiązaniem kompromisowym x^p jest rozwiązanie dopuszczalne, które cechuje się najmniejszą wartością *p-normy* ϕ_p wyznaczanej w zależności od parametru $p = 1, 2, \ldots$, jak niżej [14]:

$$\phi_p(x) = \left\| \overline{y}(x) - \overline{y}^{\inf} \right\|_p = \sqrt[p]{\sum_{n \in \boldsymbol{\mathcal{N}}} \left(\overline{y}_n(x) - \overline{y}_n^{\inf} \right)^p}. \tag{5.85}$$

Podstawiając (5.82) i (5.83) do zależności (5.85), otrzymujemy dla ocen w przestrzeni znormalizowanej p-normę w postaci:

$$\phi_p(x) = \sqrt[p]{\sum_{n \in \boldsymbol{\mathcal{N}}} \left(\frac{y_n(x) - y_n^{\inf}}{y_n^{\sup} - y_n^{\inf}} \right)^p}. \tag{5.86}$$

Warto zauważyć, że norma ϕ_2 odpowiada odległości euklidesowej od oceny $y^{\inf}$ w znormalizowanej przestrzeni kryterialnej. Rozwiązanie $x^{p=2}$ nazywane jest *rozwiązaniem Salukwadze*.

Zagadnienie wyznaczania strategii Salukwadze sformułować można jako następujący problem optymalizacji jednokryterialnej:

Dla danych wejściowych jak w zagadnieniu (4.66), zadanych punktów $y^{\sup}$, $y^{\inf}$ oraz kryteriów optymalizacji specyfikowanych zbiorem indeksów $\boldsymbol{\mathcal{N}}$ należy wyznaczyć $x^{p=2}$, takie że:

$$\phi_2(x^{p=2}) = \min_{x \in \widehat{\boldsymbol{\mathcal{X}}}} \sqrt{\sum_{n \in \boldsymbol{\mathcal{N}}} \left(\frac{y_n(x) - y_n^{\inf}}{y_n^{\sup} - y_n^{\inf}} \right)^2}, \tag{5.87}$$

gdzie $\widehat{\boldsymbol{\mathcal{X}}}$ – zbiór rozwiązań dopuszczalnych zdefiniowany w zagadnieniu optymalizacji (4.66).

W wypadku trzech kryteriów $\boldsymbol{\mathcal{N}} = \{1, 6, 9\}$ funkcja celu przyjmuje następującą postać:

$$\phi_2(x) = \sqrt{\overline{y}_1(x)^2 + \overline{y}_6(x)^2 + \overline{y}_9(x)^2}, \tag{5.88}$$

przy czym:

$$\overline{y}_1(x) = \frac{\hat{Z}_{\max}(x) - \min\limits_{j=\overline{1,J}} \max\limits_{v=\overline{1,V}} \{t_{vj}\}}{\sum\limits_{v=1}^{V} \max\limits_{j=\overline{1,J}} \{t_{vj}\} - \min\limits_{j=\overline{1,J}} \max\limits_{v=\overline{1,V}} \{t_{vj}\}}, \tag{5.89}$$

$$\overline{y}_6(x) = \frac{\Xi(x) - I \min_{j=\overline{1,J}} \{\xi_j\}}{I \left(\max\limits_{j=\overline{1,J}} \{\xi_j\} - \min\limits_{j=\overline{1,J}} \{\xi_j\} \right)}, \tag{5.90}$$

$$\overline{y}_9(x) = 1 - \frac{\kappa_{\min}^{\mathrm{RAM}}(x)}{\max_{j=\overline{1,J}} \{ram_j - r_j^{\min}\} - \max_{v=\overline{1,V}} \{r_v\}}. \tag{5.91}$$

Przy modyfikacji parametru p minimalizowanej p-normy w zagadnieniu (5.87) zmianie ulega metryka wykorzystana do oszacowania odległości od punktu idealnego. Parametr $p = 1$ powoduje, że otrzymuje się metrykę manhattańską [221]:

$$\phi_1(x) = \overline{y}_1(x) + \overline{y}_6(x) + \overline{y}_9(x). \tag{5.92}$$

Z kolei dla $p \to \infty$ uzyskuje się metrykę Czebyszewa [13], jak niżej:

$$\phi_\infty(x) = \max \{\overline{y}_1(x);\ \overline{y}_6(x);\ \overline{y}_9(x)\}. \tag{5.93}$$

W zależności od zastosowanej p-normy zmianie może podlegać wybierane rozwiązanie kompromisowe. Warto zauważyć, że dla $p \to \infty$ wytypowane rozwiązanie może nie być *Pareto*-optymalne, gdyż norma ϕ_∞ opiera się na wartości największej współrzędnej oceny w przestrzeni znormalizowanej, pomijając wartości pozostałych współrzędnych [14].

Rozwiązania kompromisowe w zagadnieniu (5.87) mogą być wyznaczane na dwa sposoby. Zaproponowana w rozprawie metoda CMGP (ang. *Compromise Multi-objective Genetic Programming*)

wykorzystuje algorytm MOGPA/xx/yy w celu wyznaczenia zbioru rozwiązań niezdominowanych $\boldsymbol{\mathcal{X}}_{\leqslant}^{\mathrm{ND}}$ dla problemu wielokryterialnego. W przypadku kryteriów cząstkowych opisanych zbiorem $\boldsymbol{\mathcal{N}} = \{1, 6, 9\}$, kryterium wektorowe optymalizacji wielokryterialnej przyjmuje postać $F = [F_1, F_6, F_9] = \left[\hat{Z}_{\max}, \Xi, \kappa_{\min}^{\mathrm{RAM}}\right]$. W konsekwencji można wykorzystać agenty klasy $MGP(F, \mu, \rho^{\leqslant}, \boldsymbol{\mathcal{F}}, \mathbb{A}, xx, yy)$ realizujące metaheurystykę MOGPA. Ze zbioru wyznaczonych rozwiązań niezdominowanych $\boldsymbol{\mathcal{X}}_{\leqslant}^{\mathrm{ND}}$ wybierane jest rozwiązanie kompromisowe x^p, takie że:

$$\phi_p(x^p) = \min_{x \in \boldsymbol{\mathcal{X}}_{\leqslant}^{\mathrm{ND}}} \phi_p(x). \tag{5.94}$$

W zaproponowanej alternatywnie metodzie CGP (ang. *Compromise Genetic Programming*) wykorzystuje się jednokryterialne programowanie genetyczne, w którym rolę funkcji celu pełni p-norma ϕ_p. Zastosować można agenty klasy $GP(\phi_p, \mu, \boldsymbol{\mathcal{F}})$ wyznaczające pojedyncze rozwiązania zadania optymalizacji jednokryterialnej $\boldsymbol{\mathcal{Z}}(\phi_p, \mu)$. Jest to przykład wyszukiwania rozwiązań poprzez skalaryzację kryterium wektorowego. Funkcja sprawności przybiera w tym przypadku postać, jak następuje:

$$\mathit{fitness}(x) = \begin{cases} -\phi_p(x) + \phi_p^{\max} + P^{\max}, & \text{dla } x \in \widehat{\boldsymbol{\mathcal{X}}}, \\ -P(x) + P^{\max}, & \text{dla } x \notin \widehat{\boldsymbol{\mathcal{X}}}, \end{cases} \tag{5.95}$$

gdzie $\phi_p^{\max}$ – oszacowanie górne zbioru wartości p-normy ϕ_p.

Największa odległość w znormalizowanej przestrzeni kryterialnej występuje pomiędzy ocenami $y^{\sup}$ i $y^{\inf}$. Dla $p = 2$ górne oszacowanie p-normy wyznacza się, jak niżej;

$$\phi_2^{\max} = \sqrt{\sum_{n \in \boldsymbol{\mathcal{N}}} \left(\frac{y_n^{\sup} - y_n^{\inf}}{y_n^{\sup} - y_n^{\inf}} \right)^2} = \sqrt{\sum_{n \in \boldsymbol{\mathcal{N}}} \left(\frac{1-0}{1-0} \right)^2} = \sqrt{\sum_{n \in \boldsymbol{\mathcal{N}}} 1} = \sqrt{|\boldsymbol{\mathcal{N}}|} \tag{5.96}$$

Złożoność obliczeniowa metody CMGP jest nie mniejsza niż złożoności algorytmu MOGPA, którą oszacowano na $O(n^6)$, gdzie $n = \max\{I, V, J, G_{\max}, G_{\mathrm{size}}\}$. Po zakończeniu algorytmu MOGPA odbywa się wybór rozwiązania kompromisowego z archiwum najlepszych osobników. Obliczenie wartości p-normy dla pojedynczego rozwiązania wymaga stałej liczby kroków, ponieważ współrzędne ocen wyznaczono za pomocą algorytmu MOGPA i zapisano w archiwum, a liczba wybranych kryteriów jest ograniczona wartością stałą. Wytypowanie rozwiązania o najniższej wartości p-normy wymaga liniowej liczby kroków w stosunku do rozmiaru archiwum. Wybór rozwiązania kompromisowego cechuje się zatem złożonością $O(n)$, przy czym $n = \Lambda$. Zatem metoda CMGP cechuje się złożonością $O(n^6)$, gdzie $n = \max\{I, V, J, G_{\max}, G_{\mathrm{size}}, \Lambda\}$.

W przypadku metody CGP złożoność obliczeniowa procedury wyznaczania p-normy wynika ze złożoności cechujących funkcje kryterialne. Najbardziej wymagające spośród nich cechują się złożonością $O(n^4)$, gdzie $n = \max\{I, V, J\}$. Procedura obliczania kary dla rozwiązania również jest ograniczona przez $O(n^4)$, $n = \max\{I, V, J\}$, ponieważ stosowane są te same funkcje, które mogą występować w roli kryteriów optymalizacji. Przy znanych wartościach p-normy i funkcji kary wyznaczenie sprawności osobnika odbywa się za pomocą stałej liczby kroków. Operacje powtarzane

są dla wszystkich osobników z populacji we wszystkich epokach. Otrzymujemy zatem złożoność $O(n^6)$, $n = \max\{I, V, J, G_{\max}, G_{\text{size}}\}$.

5.4 Optymalizacja w dynamicznym środowisku

Wykorzystanie opracowanej metaheurystyki MOGPA oraz metod CMGP i CGP pozwala na wyznaczanie sprawnych rozwiązań w zagadnieniach optymalizacji strategii sieci agentów dla statycznego modelu gridu. W szczególności mogą to być strategie kompromisowe. Należy wziąć pod uwagę fakt, że systemy rozproszone cechują się dynamiką, gdyż warunki realizacji zadań mogą ulegać zmianom w trakcie ich wykonania. W konsekwencji strategia sieci agentów wyznaczona dla jednego stanu może nie być optymalna, gdy system przejdzie do innego stanu.

Do zdarzeń istotnych z punktu widzenia strategii zespołu agentów warstwy pośredniczącej gridu zaliczyć można: dodanie nowego zadania obliczeniowego, zakończenie realizacji wybranego zadania, utratę węzła (np. w wyniku awarii), pozyskanie nowego węzła (np. przyłączenie po modernizacji systemu, przywrócenie po awarii), zmiany kosztu energii elektrycznej lub kosztu zakupu bądź dzierżawy hostów, a także zmiany mocy oferowanej przez węzły obliczeniowe. Wymienione zdarzenia znajdują odzwierciedlenie w danych wejściowych $\boldsymbol{\mathcal{Z}}^{\text{IN}}$ dla zagadnień optymalizacji, a co za tym idzie – mogą mieć wpływ na wyznaczane strategie.

W przypadku wykorzystania algorytmów ewolucyjnych, a także innych metaheurystyk, jak PSO (ang. *Particle Swarm Optimization*), wynikiem optymalizacji dla sformułowanych w rozprawie zagadnień jest wektor X. Dla takich metod optymalizacji zmiana danych wejściowych wymaga ponownego uruchomienia wybranej metaheurystyki w celu wyznaczenia nowego rozwiązania. W zagadnieniach obejmujących kilkadziesiąt zadań, agentów, węzłów i typów komputerów czas poszukiwania rozwiązań może być bardzo długi. Natomiast dopóki nowa strategia nie zostanie wyznaczona, system działa według ostatnio wyznaczonej strategii, zazwyczaj w sposób nieoptymalny.

W tym kontekście do zalet programowania genetycznego należy zaliczyć fakt, że strategia sieci agentów X jest wyznaczana przez program χ obejmujący zbiór procedur programistycznych, których wyniki zależą od stanu systemu $\boldsymbol{\mathcal{Z}}^{\text{IN}}$. Pożądane jest, aby skonstruowany program χ generował optymalne strategie nie tylko dla jednego zestawu danych wejściowych, ale dla całego szeregu stanów odzwierciedlających zdarzenia, jakie mogą zachodzić w systemie. W ten sposób reakcja na zmiany w gridzie będzie wymagała jedynie ponownego wyznaczenia strategii za pomocą wybranego programu χ. Eliminuje to konieczność oczekiwania na obliczenie kolejnego rozwiązania za pomocą metaheurystyki, skracając czas reakcji na występujące zdarzenia.

Niech *scenariusz wykonania zadań* w dynamicznym środowisku definiuje się jako sekwencję uporządkowanych chronologicznie stanów wejściowych dla zagadnienia optymalizacji w modelu statycznym, jak niżej:

$$(\boldsymbol{\mathcal{Z}}_1^{\text{IN}}, \ldots, \boldsymbol{\mathcal{Z}}_q^{\text{IN}}, \ldots, \boldsymbol{\mathcal{Z}}_Q^{\text{IN}}). \tag{5.97}$$

W tabeli nr 5.2 zaprezentowano przykładowy scenariusz składający się z $Q = 6$ etapów. Przedstawiono, w jaki sposób zmieniają się dane wejściowe zagadnienia optymalizacji ze względu na

wybrane zdarzenia w systemie. Każdy etap może wymagać innego przypisania zadań użytkowników, innego przemieszczenia agentów i innego rozdziału zasobów w gridzie dla uzyskania jak najwyższych ocen pod względem wybranych przez interesariusza kryteriów. Warto zauważyć, że niektóre elementy strategii X będą niezmienne we wszystkich etapach scenariusza. Przykładowo, gdy nie jest planowany zakup komputerów pomiędzy etapami, raz wybrany zestaw sprzętowy nie będzie podlegał zmianom. W związku z tym przypisanie typów komputerów do węzłów – wektor X^{β} – jest stałe w całym scenariuszu, a modyfikacjom podlega przypisanie zadań obliczeniowych do agentów oraz rozmieszczenie agentów w węzłach.

Jeśli planowane jest wykorzystanie wyłącznie posiadanych komputerów, można uwzględnić wymaganie zagwarantowania liczby komputerów zadanych typów ($\mu_{19} \geqslant 1$) i zawęzić zbiór dostępnych typów już w pierwszym etapie scenariusza. W efekcie zagwarantowane zostanie wykorzystanie komputerów, które są na wyposażeniu gridu. Możliwe są też rozwiązania pośrednie, w których system będzie rozbudowany o nowe węzły. Zbiór danych wejściowych pozwala na specyfikację każdego z tych stanów.

Przy analizie scenariusza konieczne jest uwzględnienie czasu, jaki upływa pomiędzy kolejnymi zdarzeniami. Przez $t^{\mathrm{int}}_{q_1,q_2}$ rozumiana jest długość interwału pomiędzy stanami $\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}_{q_1}$ i $\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}_{q_2}$. W tym czasie paczki danych dla zadań, które już uruchomiono w systemie, są przetwarzane w węzłach obliczeniowych. W gridzie *Comcute* moc oferowana przez wolontariuszy jest równomiernie rozdzielana pomiędzy zadania. Niech $\widehat{L}_{q_1,q_2}(z_m)$ oznacza liczbę przetworzonych paczek danych zadania z_m w czasie $t^{\mathrm{int}}_{q_1,q_2}$. Wartości $\widehat{L}_{q_1,q_2}(z_m)$ można oszacować, przekształcając zależność (4.50), jak niżej:

$$\widehat{L}_{q_1,q_2}(z_m) = \frac{\widehat{I}}{M} \cdot \frac{\mathscr{T}_a}{\mathscr{T}_a + \mathscr{T}_u} \cdot \frac{t^{\mathrm{int}}_{q_1,q_2}}{P_T(z_m)P_R(z_m)}. \tag{5.98}$$

Tabela 5.2: Przykładowy scenariusz wykonania zadań w gridzie

Stan	Zdarzenie
$\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}_1$	*Stan początkowy* Dane wejściowe: $[\mu_1, \ldots, \mu_{19}]$, $\{z_1, z_2\}$, $M = 2$, przy czym $L(z_1) = 1024000$, $L(z_2) = 512000$, $P_R(z_1) = P_R(z_2) = 1$, $P_W(z_1) = P_W(z_2) = 4$, $P_T(z_1) = P_T(z_2) = 5$ s, $V_W = 4$, $I = 6$, $V = 12$, $J = 12$, $c_\varepsilon = 0{,}3588$ zł/kWh, ξ, ϑ, $\varepsilon^{\mathrm{load}}$, $\varepsilon^{\mathrm{idle}}$ (parametry komputerów opisano w tabeli nr 3.1), $T = [t_{vj}]_{V\times J}$ zdefiniowane jak w (4.1) oraz $\tau = [\tau_{vu}]_{V\times V}$ zdefiniowane jak w (4.2).
$\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}_2$	*Dodanie nowego zadania obliczeniowego* ($t^{\mathrm{int}}_{1,2} = 600$ s) Zbiór zadań rozszerzono o zadanie z_3, przy czym: $L(z_3) = 512000$, $P_R(z_3) = 1$, $P_W(z_3) = 4$, $P_T(z_3) = 10$ s. Nowa liczba zadań wynosi $M = 3$. W czasie $t^{\mathrm{int}}_{1,2}$ zadania z_1 i z_2 zostały częściowo zrealizowane, liczbę pozostałych dla nich paczek danych pomniejszono zgodnie z zależnością (5.98). Przyjęto, że zakup i instalacja komputerów w węzłach odbywa się przed rozpoczęciem realizacji zadań. Wybrane typy komputerów nie zmieniają się pomiędzy etapami scenariusza, co wyraża aktywne wymaganie $\mu_{19} = 1$. Wybrane komputery zostają dodane do zbioru wymaganych typów $\mathcal{J}^*$, a liczba dostępnych sztuk ν_j jest równa liczbie węzłów, w których wykorzystano komputer j-tego typu dla wszystkich $j \in \mathcal{J}^*$. Pozostałe dane są takie, jak w $\boldsymbol{\mathcal{Z}}^{\mathrm{IN}}_1$.

$\boldsymbol{\mathcal{Z}}_3^{\mathrm{IN}}$	*Awaria węzła* ($t_{2,3}^{\mathrm{int}} = 3600$ s) Jeden z komputerów uległ awarii, numer jego typu oznacza się jako j^{error}. Liczba dostępnych sztuk $\nu_{j^{\mathrm{error}}}$ zostaje ustawiona na wartość o jeden mniejszą od liczby węzłów, w których komputer tego typu został wykorzystany. Do zbioru typów komputerów zostaje wprowadzony nowy typ o numerze j^*, cechujący się wydajnością $\vartheta_{j^*} = 0$ i liczbą dostępnych sztuk $\nu_{j^*} = 1$. Jego użycie jest obowiązkowe, co wyraża przynależność j^* do zbioru $\mathcal{J}^*$. Powyższa zmiana danych wejściowych wymusza wyłączenie z użycia jednego z węzłów do momentu jego naprawy. Liczba pozostałych do przetworzenia paczek danych dla wszystkich zadań jest pomniejszona zgodnie z zależnością (5.98). Pozostałe dane są takie, jak w stanie $\boldsymbol{\mathcal{Z}}_2^{\mathrm{IN}}$.
$\boldsymbol{\mathcal{Z}}_4^{\mathrm{IN}}$	*Przywrócenie węzła po awarii* ($t_{3,4}^{\mathrm{int}} = 7200$ s) Ze zbioru dostępnych typów komputerów usunięto typ β_{j^*}, parametr $\nu_{j^{\mathrm{error}}}$ otrzymuje wartość jak w stanie $\boldsymbol{\mathcal{Z}}_2^{\mathrm{IN}}$. Liczba pozostałych do przetworzenia paczek danych dla wszystkich zadań jest pomniejszona zgodnie z zależnością (5.98). Pozostałe dane są takie, jak w poprzednim stanie.
$\boldsymbol{\mathcal{Z}}_5^{\mathrm{IN}}$	*Zmiana kosztu energii elektrycznej* ($t_{4,5}^{\mathrm{int}} = 1800$ s) W przypadku taryfy energetycznej o zmiennej stawce w ciągu doby, koszt energii zmienia się o określonych porach, np. niższa opłata obowiązuje w godzinach pozaszczytowych 22:00-6:00. Niech nowa obniżona stawka wynosi $c_\varepsilon = 0{,}2376$ zł/kWh. Liczba pozostałych do przetworzenia paczek danych dla wszystkich zadań jest pomniejszona zgodnie z zależnością (5.98). Pozostałe dane są takie, jak w poprzednim stanie.
$\boldsymbol{\mathcal{Z}}_6^{\mathrm{IN}}$	*Zakończenie wykonywania zadania* Jeśli nie występują inne zdarzenia, kolejną istotną zmianę danych wejściowych dla zagadnienia optymalizacji niesie zakończenie obsługi zadania. Ze względu na sposób równoważenia mocy oferowanej przez wolontariuszy pomiędzy zadania, w pierwszej kolejności będzie zrealizowane zadanie z_m^* takie, że: $$L(z^*)P_R(z^*)P_T(z^*) = \min_{z_m \in \{z_1,\ldots,z_M\}} L(z_m)P_R(z_m)P_T(z_m) \tag{5.99}$$ Czas potrzebny na zakończenie zadania z_m^* można oszacować od dołu w oparciu o zależność (4.50), jak niżej: $$t_{5,6}^{\mathrm{int}} = L(z^*)P_R(z^*)P_T(z^*)\frac{\mathscr{T}_a + \mathscr{T}_u}{\mathscr{T}_a}\, M\widehat{I}^{-1} \tag{5.100}$$ Dla zadań innych niż z^* liczba pozostałych do przetworzenia paczek danych jest pomniejszona zgodnie z zależnością (5.98). Zadanie z^* zostaje usunięte ze zbioru zadań, w związku z czym $M = 2$. Inne parametry zachowują wartości z poprzedniego etapu.

Źródło: opracowanie własne

Szersze możliwości rekonfiguracji grida pomiędzy etapami scenariusza umożliwia dzierżawa węzłów w chmurze obliczeniowej. W tym przypadku węzły mają postać maszyn wirtualnych uruchomionych w obrębie infrastruktury usługodawcy, dzięki czemu możliwa jest zmiana ich konfiguracji w trakcie realizacji scenariusza. Zmiana profilu sprzętowego maszyny wirtualnej rozpoczyna się od wstrzymania jej działania. Następnie na poziomie serwera wirtualizacji alokowane są wymagane

zasoby wynikające z wybranego profilu. Ostatnim krokiem jest wznowienie działania maszyny wirtualnej z nowymi ustawieniami.

Nie występują w tym przypadku opóźnienia związane z koniecznością zakupu, instalacji w węźle i konfiguracji nowego komputera. Nie są też ponoszone koszty zakupu, $\xi_j = 0$ dla $j = \overline{1,J}$. Wybranie maszyny wirtualnej o większej wydajności wiąże się natomiast z wyższymi kosztami dzierżawy. W tabeli nr 5.3 zaprezentowano wybrane profile maszyn wirtualnych dostępne w chmurze *Amazon Elastic Compute Cloud* (*EC2*) [12].

Tabela 5.3: Wybrane profile sprzętowe dla maszyn wirtualnych w chmurze *Amazon EC2*

j	Nazwa profilu	ram_j [GB]	hdd_j [TB]	$\hat{c}_j^{\mathrm{EU}}$ [zł/h]	$\hat{c}_j^{\mathrm{JP}}$ [zł/h]	ϑ_j [ECU]	ν_j
1	m4.large	8	1	0,50	0,68	6,5	$\geqslant I$
2	m4.xlarge	16	1	1,01	1,36	13	$\geqslant I$
3	m4.2xlarge	32	1	2,01	2,71	26	$\geqslant I$
4	m4.4xlarge	64	1	4,02	5,43	53,5	$\geqslant I$
5	m4.10xlarge	160	1	10,06	13,56	124,5	$\geqslant I$
6	*none*	0	0	0	0	0	$\geqslant I$

Źródło: opracowanie własne na podstawie [12]

W chmurze obliczeniowej koszt dzierżawy zazwyczaj zależy od czasu działania maszyny wirtualnej, a nie od jej obciążenia. Dostawcy, którzy oferują dzierżawę w różnych lokalizacjach geograficznych, mogą uzależniać koszty pracy maszyn od rejonów lub państw ich ulokowania. W tabeli nr 5.3 wyróżniono koszty dzierżawy w centrum obliczeniowym zlokalizowanym w Irlandii ($\hat{c}_j^{\mathrm{EU}}$) oraz w Japonii ($\hat{c}_j^{\mathrm{JP}}$). Jeśli interesariusz zażąda uruchomienia węzłów w określonych lokalizacjach, koszt pracy maszyn wirtualnych o takiej samej konfiguracji zależy od węzła.

Ponieważ w chmurze obliczeniowej węzły są maszynami wirtualnymi, nie występują limity dostępnych komputerów, a zatem $\nu_j \geqslant I$, $j = \overline{1,J}$. Zapotrzebowanie energetyczne ε_j nie jest istotne z punktu widzenia użytkownika usług w chmurze, gdyż jest wliczone przez dostawcę w koszt dzierżawy komputerów. Wydajność typów komputerów w tabeli nr 5.3 podano w jednostkach ECU (*EC2 Compute Unit*), dla których przyjmuje się, że 1 ECU $\approx$ 400 punktów benchmarku *CPU Mark* firmy *PassMark* [212].

W tabeli nr 5.3 uwzględniono dodatkowy typ komputera β_6 (profil *none*) cechujący się zerową wydajnością $\vartheta_6 = 0$. Jego wybór interpretowany jest jako rezygnacja z wykorzystania węzła. Taka możliwość zwiększa swobodę w konfigurowaniu systemu, pozwalając wybrać pomiędzy mniejszą liczbą bardziej wydajnych węzłów lub większą liczbą mniej wydajnych, przy niezmiennym limicie kosztu pracy systemu. Ponadto w dynamicznym scenariuszu typ β_6 pozwala wyłączyć niektóre węzły, kiedy spada obciążenie systemu. Jeśli dostępny jest komputer o profilu *none*, parametr I w zagadnieniu optymalizacji interpretowany jest jako *maksymalna liczba aktywnych węzłów*.

Dla zadanego scenariusza za pomocą programowania genetycznego poszukiwane są programy χ do wyznaczania strategii sieci agentów przeznaczonych do wykorzystania na różnych etapach

scenariusza. Zbiór rozwiązań dopuszczalnych $\widehat{\boldsymbol{\mathcal{X}}}_{\boldsymbol{\mathcal{P}}}$ obejmuje programy, które wyznaczają strategie dopuszczalne dla wszystkich stanów systemu w rozpatrywanej sekwencji, co definiuje się, jak niżej:

$$\widehat{\boldsymbol{\mathcal{X}}}_{\boldsymbol{\mathcal{P}}} = \left\{ \chi : X_q^{\chi} \in \widehat{\boldsymbol{\mathcal{X}}},\ q = \overline{1,Q} \right\}, \tag{5.101}$$

gdzie $X_q^{\chi} = \boldsymbol{\mathcal{I}}(\chi, \boldsymbol{\mathcal{Z}}_q^{\mathrm{IN}})$ – strategia wyznaczana przez program χ dla stanu $\boldsymbol{\mathcal{Z}}_q^{\mathrm{IN}}$ w wyniku zastosowania procedury *interpretacja*.

Najbardziej pożądane są programy χ, które wyznaczają wysokiej jakości strategie dla wszystkich zestawów danych $\boldsymbol{\mathcal{Z}}_q^{\mathrm{IN}}$ objętych scenariuszem. Niech funkcja $\phi_p^{\mathrm{suma}}(\chi)$ odpowiada sumie odległości ocen strategii wyznaczanych przez program χ od zadanego punktu idealnego na wszystkich etapach scenariusza:

$$\phi_p^{\mathrm{suma}}(\chi) = \sum_{q=1}^{Q} \phi_p(X_q^{\chi}) = \sum_{q=1}^{Q} \sqrt[p]{\sum_{n \in \boldsymbol{\mathcal{N}}} \left(\frac{y_n(X_q^{\chi}) - y_n^{\mathrm{inf}}}{y_n^{\mathrm{sup}} - y_n^{\mathrm{inf}}} \right)^p}. \tag{5.102}$$

Zatem sformułować można następujące zagadnienie optymalizacji programu do wyznaczania strategii zespołu agentów warstwy pośredniczącej gridu w dynamicznym środowisku.

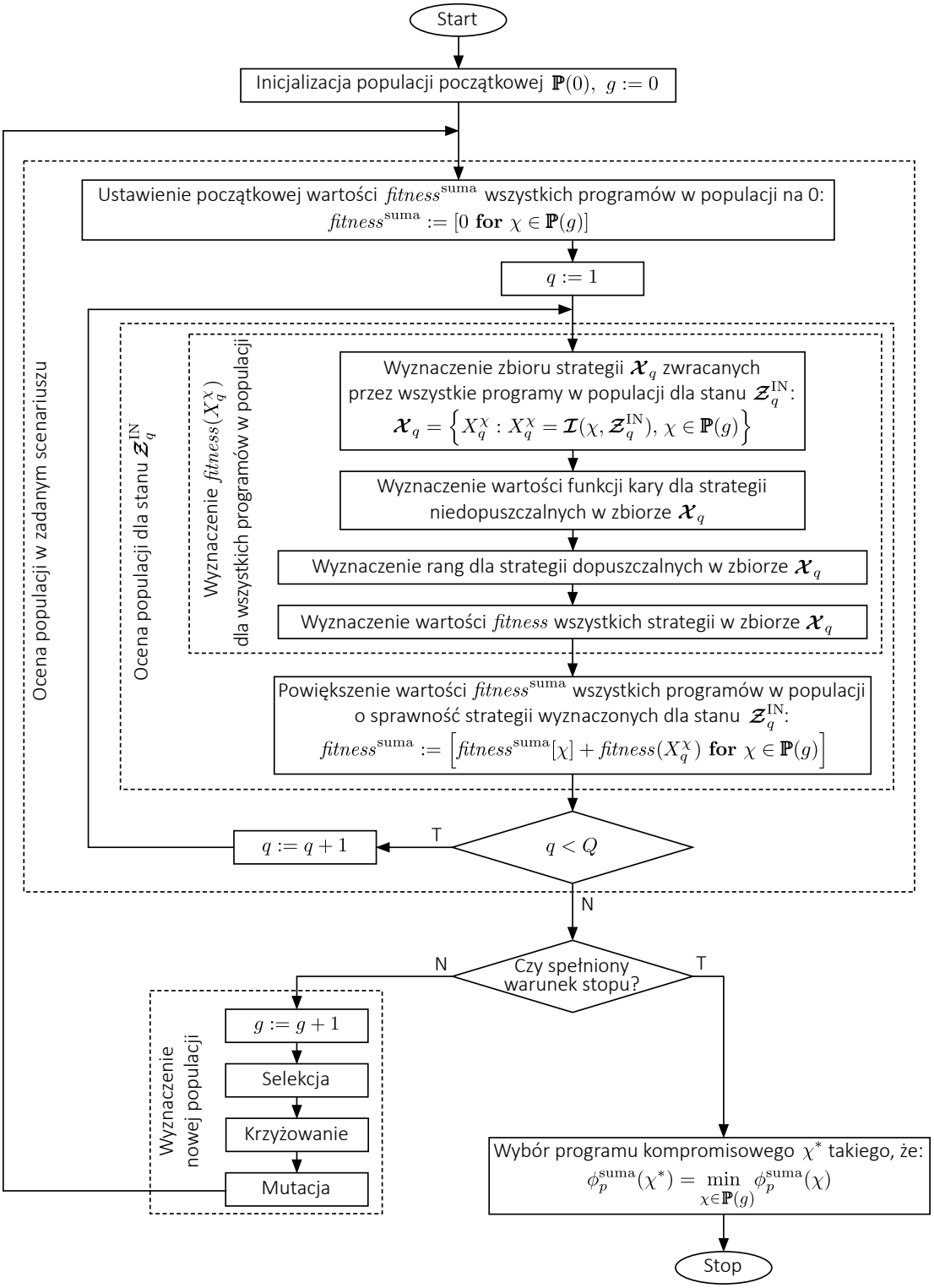
Dla scenariusza opisanego sekwencją stanów $(\boldsymbol{\mathcal{Z}}_1^{\mathrm{IN}}, \ldots, \boldsymbol{\mathcal{Z}}_q^{\mathrm{IN}}, \ldots, \boldsymbol{\mathcal{Z}}_Q^{\mathrm{IN}})$, punktów referencyjnych y^{inf} i y^{sup}, parametru p dla normy ϕ_p oraz kryterium wektorowego F, należy wyznaczyć program χ^*, taki że:

$$\phi_p^{\mathrm{suma}}(\chi^*) = \min_{\chi \in \widehat{\boldsymbol{\mathcal{X}}}_{\boldsymbol{\mathcal{P}}}} \phi_p^{\mathrm{suma}}(\chi), \tag{5.103}$$

gdzie:
$\phi_p^{\mathrm{suma}}(\chi)$ – suma odległości ocen strategii wyznaczanych przez program χ od zadanego punktu idealnego na wszystkich etapach scenariusza, zadana zależnością (5.102),
$\widehat{\boldsymbol{\mathcal{X}}}_{\boldsymbol{\mathcal{P}}}$ – zbiór rozwiązań dopuszczalnych zawierający programy, które wyznaczają strategie dopuszczalne dla wszystkich stanów systemu w sekwencji scenariusza wg zależności (5.101).

Do wyznaczania rozwiązań w zagadnieniu (5.103) proponuje się heurystykę DMGP (ang. *Dynamic Multi-objective Genetic Programming*), która opiera się na zaproponowanej wcześniej metodzie CMGP. Kluczowa modyfikacja odnosi się do procedury oceny osobników, za pomocą której należy uwzględnić stany $\boldsymbol{\mathcal{Z}}_1^{\mathrm{IN}}, \ldots, \boldsymbol{\mathcal{Z}}_q^{\mathrm{IN}}, \ldots, \boldsymbol{\mathcal{Z}}_Q^{\mathrm{IN}}$ w przyjętym scenariuszu. Zmianie ulega również fenotyp osobników. W przeciwieństwie do metod dla modeli statycznych, w których fenotypem była strategia X, w heurystyce DMGP za reprezentację fenotypową uznawany jest program do wyznaczania strategii χ. Diagram metody DMGP przedstawiono na rysunku 5.22.

Niech dla każdego stanu scenariusza wyznaczane są strategie zespołu agentów z wykorzystaniem programów reprezentowanych przez osobniki populacji. Uzyskane strategie podlegają ocenie, jak w algorytmie MOGPA. Dla strategii niedopuszczalnych wyznaczane są wartości funkcji kary, a dla dopuszczalnych – wartości rang. Następnie obliczana jest sprawność strategii zgodnie z zależnością (5.40). Sprawność programu χ jest powiększana o sprawność wyznaczonej przez niego strategii X_q^{χ} dla danych wejściowych $\boldsymbol{\mathcal{Z}}_q^{\mathrm{IN}}$. Operacja jest powtarzana dla wszystkich etapów scenariusza.



Rysunek 5.22: Schemat blokowy metody DMGP
Źródło: opracowanie własne

Łączną sprawność programu χ $fitness^{\text{suma}}(\chi)$ definiuje się następująco:

$$fitness^{\text{suma}}(\chi) = \sum_{q=1}^{Q} fitness(X_q^{\chi}), \tag{5.104}$$

gdzie $fitness(X_q^{\chi})$ – sprawność strategii, którą wyznaczył program χ dla q-tego kroku scenariusza. Sprawność powyższej strategii szacuje się za pomocą formuły (5.40).

Najbardziej pożądane są programy χ, które wyznaczają strategie niezdominowane na wszystkich etapach scenariusza. Po wyznaczeniu wartości $fitness^{\text{suma}}$ dla programów w populacji, realizowane są operacje selekcji, krzyżowania i mutacji, prowadzące do kolejnej epoki programowania genetycznego.

Jeśli warunek stopu zostanie spełniony po etapie oceny rozwiązań, kompromisowy program do wyznaczania strategii jest wybierany spośród osobników aktualnej populacji w oparciu o funkcję $\phi_p^{\text{suma}}(\chi)$ zdefiniowaną za pomocą zależności (5.102). Wybór rozwiązania kompromisowego kończy przebieg algorytmu. Metoda DMGP jest realizowana przez agenty klasy $DGP(F, y^{\text{inf}}, y^{\text{sup}}, p, [\boldsymbol{\mathcal{Z}}_1^{\text{IN}}, \ldots, \boldsymbol{\mathcal{Z}}_Q^{\text{IN}}])$.

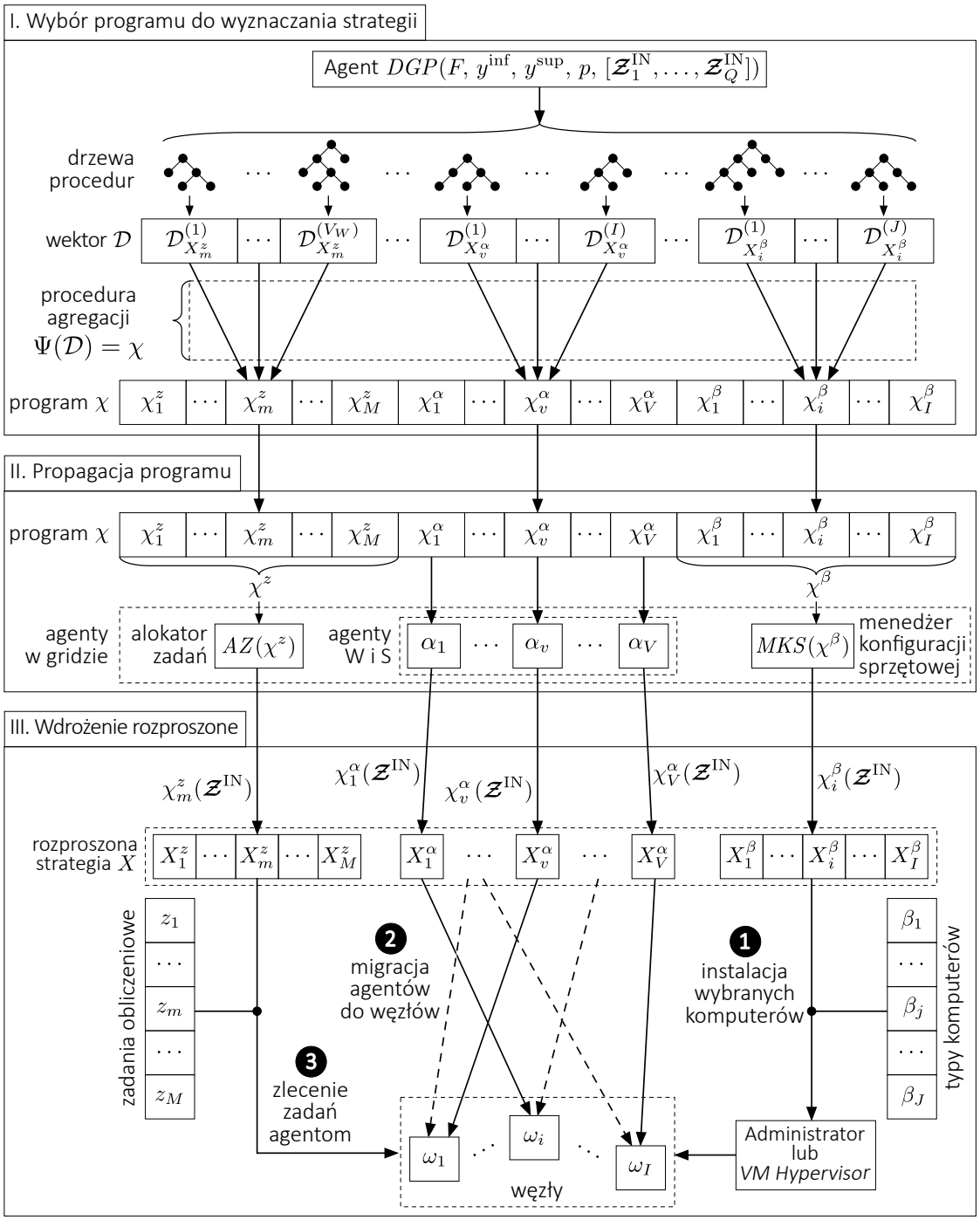
Złożoność obliczeniowa metody CMGP wynosi $O(n^6)$, $n = \max\{I, V, J, G_{\max}, G_{\text{size}}\}$. W metodzie DMGP oceny osobników we wszystkich epokach wyznaczane są Q razy – raz dla każdego kroku scenariusza. Złożoność heurystyki DMGP wynosi zatem $O(n^7)$, gdzie $n = \max\{I, V, J, G_{\max}, G_{\text{size}}, Q\}$.

Zastosowanie wybranego programu do wyznaczania strategii w systemie *Comcute* wymaga realizacji *wdrożenia rozproszonego*, któremu towarzyszy realokacja zasobów gridu i migracja agentów programistycznych, a także przypisanie zadań do agentów zarządzających. Na rysunku 5.23 przedstawiono koncepcję wykorzystania metody DMGP w systemie *Comcute*. Pierwszy etap polega na wyborze programu do wyznaczania strategii χ, przy czym uwzględnia się kryteria optymalizacji wybrane przez interesariusza.

Drugim krokiem jest propagacja programu χ do agentów warstwy pośredniczącej gridu (etap II na rys. 5.23). Każdy agent α_v, $v = \overline{1,V}$ typu W lub S otrzymuje procedurę χ_v^{α} określającą jego pożądane ulokowanie w gridzie w zależności od stanu systemu. Niech za przydział zadań obliczeniowych do agentów typu W odpowiada pomocniczy agent *alokator zadań* $AZ(\chi^z)$, który wykorzystuje w tym celu wektor procedur programistycznych $\chi^z = [\chi_1^z, \ldots, \chi_m^z, \ldots, \chi_M^z]^T$. Natomiast za uruchomienie wybranych typów komputerów w węzłach systemu niech odpowiada kolejny pomocniczy agent *menedżer konfiguracji sprzętowej* $MKS(\chi^{\beta})$, który otrzymuje wektor procedur $\chi^{\beta} = [\chi_1^{\beta}, \ldots, \chi_i^{\beta}, \ldots, \chi_I^{\beta}]^T$. Po propagacji procedur sieć inteligentnych agentów warstwy pośredniczącej gridu działa według programu rozproszonego χ, który zaimplementowano automatycznie przy pomocy programowania genetycznego.

Agenty typu W i S oraz agenty pomocnicze $AZ(\chi^z)$ i $MKS(\chi^{\beta})$ wykorzystują otrzymane procedury programu χ w celu wyznaczenia składowych wektora strategii X dla aktualnego stanu gridu $\boldsymbol{\mathcal{Z}}^{\text{IN}}$. Agent *alokator zadań* wykorzystuje otrzymane procedury χ^z do wyznaczenia wektora

całkowitoliczbowego X^z. Z kolei agent *menedżer konfiguracji sprzętowej* wykorzystuje wektor procedur χ^β do wyznaczenia konfiguracji sprzętowej X^β. Natomiast każdy z agentów typu W i S autonomicznie wyznacza swoje pożądane ulokowanie w gridzie za pomocą otrzymanej procedury χ_v^α.



Rysunek 5.23: Rozproszone wdrożenie strategii przez zespół agentów warstwy pośredniczącej w gridzie *Comcute*
Źródło: opracowanie własne

W efekcie tych operacji następuje wyznaczenie strategii X. Uzyskana strategia ma postać rozproszoną – poszczególne składowe są wyznaczane przez różne agenty, a każdy z nich utrzymuje jedynie ten jej fragment, który jest istotny dla jego działania. Wdrożenie pozyskanej strategii rozproszonej przebiega w trzech krokach (etap III na rys. 5.23).

W pierwszym kroku *menedżer konfiguracji sprzętowej* przekazuje informacje o wybranych typach komputerów do administratora, którego zadaniem jest instalacja i uruchomienie komputerów w węzłach. Jeśli węzły są dzierżawione w chmurze, agent $MKS(\chi^{\beta})$ zleca uruchomienie pożądanych maszyn wirtualnych *nadzorcy wirtualizacji* (ang. *VM Hypervisor*). Dostawcy usług w chmurze zazwyczaj oferują programistyczne API pozwalające na zarządzanie maszynami. W tym przypadku przygotowanie węzłów nie wymaga interwencji administratora i może odbyć się automatycznie. Z kolei w instancji laboratoryjnej grida *Comcute* wykorzystać można narzędzia do zdalnego administrowania, takie jak *iDRAC* (ang. *integrated Dell Remote Access Controller*) w celu włączania i wyłączania skonfigurowanych uprzednio serwerów fizycznych.

Jeśli wskazane typy komputerów zostaną uruchomione w węzłach systemu, nastąpi migracja agentów klasy W i S do wybranych przez nich lokalizacji (etap III, krok 2 na rys. 5.23). W ostatnim kroku agent $AZ(\chi^{z})$ przypisuje zadania obliczeniowe do agentów typu W, które rozpoczynają proces realizacji zadań. Warto zauważyć, że agenty warstwy pośredniczącej gridu decydują w sposób autonomiczny o swoim położeniu i przydziale zasobów zgodnie z przyjętą strategią.

Proponowana metoda wspomaga inteligentne reagowanie na zmiany zachodzące w gridzie. W przypadku wystąpienia zdarzenia, które może wymagać rekonfiguracji systemu, agenty $AZ(\chi^{z})$, $MKS(\chi^{\beta})$ oraz agenty typów W i S ponownie wykorzystują otrzymane procedury programu do wyznaczania strategii χ. Nową strategię wyznacza się w zależności od aktualnego stanu gridu, po czym następuje jej wdrożenie. Reakcja na zaistniałe zdarzenie nie wymaga ponownego uruchomienia programowania genetycznego ani interwencji administratora.

Warto zauważyć, że zdolność programu χ do wyznaczania wysokiej jakości strategii zależy od tego, na ile reprezentatywny dla stanów gridu był scenariusz wykorzystany na etapie programowania genetycznego. Dostrajanie programu do wyznaczania strategii może odbywać się poprzez zapisywanie stanów gridu w trakcie jego pracy do wykorzystania w scenariuszu w kolejnej instancji metaheurystyki DMGP.

Na tym tle, warto rozważyć możliwość uczenia się przez agenty na podstawie par uczących. W tym wypadku zestaw danych wejściowych do problemu optymalizacji dla wybranego stanu gridu reprezentuje wejście do programowania genetycznego. Natomiast wyznaczona za pomocą kompromisowego programowania genetycznego strategia reprezentuje wzorzec wyjścia dla pary treningowej. Wprawdzie taki rodzaj uczenia maszynowego w odniesieniu do programowania genetycznego jest możliwy, to w wypadku omawianej problematyki wzorcowe pary uczące powinny być wyznaczone dla wielu instancji problemu optymalizacji kompromisowej. Rozwiązywane instancje powinny różnić się pod względem liczby agentów, liczby węzłów, liczby rodzajów komputerów, a także ich charakterystyk. W rezultacie należy oczekiwać kilkuset tysięcy par treningowych dla gridów „średniej" wielkości, aby proces treningu był reprezentatywny dla problemu. Przyjmując, że

czas wyznaczenia jednej strategii wynosi 10 minut, czas wstępnego przygotowania zbioru uczącego i zbioru testowego można szacować na tysiące godzin.

Tej klasy uczenie maszynowe z nauczycielem może być zastosowane w początkowej fazie treningu agentów dla mniejszej liczby par uczących. Następnie system może być trenowany bez nauczyciela. Interesująca koncepcja polega na wykorzystaniu w drugiej fazie uczenia metody treningu ze wzmocnieniem. W tym podejściu strategie agentów mogą być konfrontowane ze strategiami wyznaczonymi za pomocą metod optymalizacji przy uwzględnieniu nałożonego ograniczenia na czas obliczeń solwerów. Szczegółowo metody uczenia maszynowego Autor opisał w pracy [142].

5.5 Zastosowanie modelu agentowego do wspomagania wykonania zadań obliczeniowych w gridzie *Comcute*

Aplikacja Administratora Gridu AAG'16 zawiera implementacje opracowanych metod optymalizacji strategii zespołu agentów w systemach rozproszonych, w tym w gridzie *Comcute*. Aplikacja została zrealizowana w oparciu o platformę *Java Enterprise Edition 7*, a jej środowiskiem wykonawczym jest serwer *Wildfly 9*. Interfejs aplikacji (rysunek 5.24) jest dostępny za pośrednictwem przeglądarki internetowej. Szersze omówienie aplikacji AAG'16 zamieszczono w dodatku B.

Rysunek 5.24: Interfejs Aplikacji Administratora Gridu AAG'16
Źródło: opracowanie własne

W celu porównania opracowanej metody MOGPA z innymi metodami wyznaczania rozwiązań dla zagadnień wielokryterialnych, rozpatruje się problem dwukryterialnej optymalizacji obciążenia CPU i obciążenia komunikacyjnego newralgicznych węzłów [209]. W zagadnieniu nie uwzględnia się zadań obliczeniowych zlecanych przez użytkowników gridu. Zamiast tego wykorzystywano obciążenia zadane predefiniowanymi macierzami czasów zajętości CPU i obciążenia komunikacyjnego oraz wektorami zapotrzebowania na pamięć RAM i HDD. Ponadto nie jest brana pod uwagę dynamika mocy obliczeniowej oferowanej przez wolontariuszy.

Zestaw danych wejściowych dla zagadnienia optymalizacji dostosowano tak, aby odzwierciedlał warunki cytowanego eksperymentu. Przyjęto realizację jednego zadania, w którego wykonanie zaangażowane są wszystkie agenty zgodnie z zadanymi macierzami T i τ. W efekcie sformułowano zagadnienie optymalizacji, jak niżej:

Przy zadanych selektorach ograniczeń:

- $\mu_5 = 1$ – minimalna wydajność gridu $\vartheta_{\min} = 200$ TFLOPS,
- $\mu_6 = 1$ – maksymalny koszt zakupu komputerów $\xi_{\max} = 70000$ USD,
- $\mu_8 = 1$ – minimalny stopień rozproszenia $\eta_{\min} = 180$,
- $\mu_9 = 1$ – nieprzekroczenie wielkości dostępnej pamięci operacyjnej,
- $\mu_{10} = 1$ – nieprzekroczenie wielkości dostępnej pamięci HDD,
- $\mu_{15} = 1$ – maksymalne zapotrzebowanie energetyczne $\varepsilon_{\max} = 38$ kW,

oraz danych wejściowych:

$M = 1$, $V = 24$, $I = 9$, $J = 10$,

hdd = [8,256; 20; 38; 48,256; 70; 138; 184,256; 200; 380; 500] [GB],

h = [0,5;0,3;0,4;0,3;0,5;1,7;2,7;0,3;0,4;0,5;0,3;0,5;0,5;0,3;0,4;0,3;0,5;1,7;2,7;0,3;0,4;0,5;0,3;0,5] [GB],

ram = [0,5; 3; 6; 10,5; 12; 17; 20,5; 23; 30; 42] [GB],

r = [15; 350; 25; 25; 35; 35; 25; 35; 45; 25; 35; 25; 15; 350; 25; 25; 35; 35; 25; 35; 45; 25; 35; 20] [MB],

ϑ = [1,84; 4,848; 6,771; 12,84; 14,848; 16,771; 21,84; 24,848; 26,771; 40,2] [TFLOPS],

ξ = [400, 896, 1504, 1900, 2896, 3504, 4000, 4896, 5504, 7012] [USD],

ε = [250, 600, 650, 750, 800, 950, 1250, 1600, 1650, 2100] [W],

$$T = \begin{bmatrix}
1&2&3&1&1&2&1&1&1&1\\
3&2&1&2&2&1&1&2&1&1\\
2&2&3&1&1&2&2&1&1&1\\
4&3&3&3&1&3&2&1&3&1\\
3&2&3&2&2&2&1&1&1&1\\
1&2&1&1&1&1&1&1&1&1\\
2&3&2&2&3&2&2&2&2&2\\
1&2&1&1&2&1&1&2&1&1\\
2&1&2&2&1&2&2&1&2&2\\
3&4&2&2&4&2&1&1&2&2\\
1&2&2&1&2&2&1&2&1&2\\
3&2&1&3&1&1&3&2&1&1\\
1&2&3&1&2&2&1&1&1&1\\
3&2&1&2&2&1&3&2&1&1\\
2&2&3&2&1&2&2&2&3&1\\
4&3&3&3&3&2&1&1&3&1\\
3&2&3&3&2&2&3&2&1&1\\
1&2&1&1&2&1&1&2&1&1\\
2&3&2&2&3&2&2&3&2&2\\
1&2&1&1&1&1&1&2&1&1\\
2&1&2&2&1&2&2&1&2&2\\
3&4&2&3&4&1&1&2&1&1\\
1&2&2&1&2&2&1&2&2&2\\
3&2&1&2&1&1&3&2&1&1
\end{bmatrix}\ [\text{s}],\quad \tau = \begin{bmatrix}
0&1&2&1&0&3&2&3&2&0&1&2&2&0&0&0&1&0&0&0&2&1&1&0\\
3&0&1&2&3&4&0&0&0&0&1&0&0&1&2&1&0&3&2&3&0&0&1&2\\
1&0&0&1&1&0&1&2&2&1&0&0&0&1&0&0&0&3&2&0&2&0&1&0\\
0&2&0&0&0&1&7&4&7&7&2&5&3&1&2&1&0&0&0&3&0&0&1&2\\
0&0&0&0&0&8&1&5&2&5&0&0&0&1&2&1&0&3&2&3&2&0&1&0\\
4&0&0&2&0&0&3&2&7&4&0&2&0&0&0&1&0&3&2&0&2&1&1&2\\
0&4&0&0&5&5&0&7&2&7&0&1&4&1&2&0&0&0&0&3&2&0&1&2\\
0&1&2&1&4&7&1&0&0&0&4&1&0&1&2&1&0&3&2&3&0&0&0&0\\
2&1&0&5&1&1&4&9&0&0&1&4&2&0&0&0&0&0&2&0&2&0&1&2\\
0&0&1&0&0&2&0&1&7&0&0&1&0&1&2&1&0&3&0&3&0&0&1&2\\
4&0&4&1&5&0&0&0&0&0&0&1&0&1&2&1&0&3&2&0&2&0&0&2\\
0&1&0&0&1&0&0&1&0&4&1&0&2&0&0&0&0&0&2&3&2&0&1&0\\
0&1&2&1&0&3&2&3&2&0&1&2&0&1&2&1&0&3&2&3&2&0&1&2\\
3&0&1&2&3&4&0&0&0&0&1&0&3&0&1&2&3&4&0&0&0&0&1&0\\
1&0&0&1&1&0&1&2&2&1&0&0&0&0&0&0&3&0&0&0&0&0&1&0\\
0&2&0&0&0&1&7&4&7&7&2&5&3&0&1&0&0&4&0&0&0&0&0&0\\
0&0&0&0&0&8&1&5&2&5&0&0&0&0&0&0&0&0&0&0&0&1&1&0\\
4&0&0&2&0&0&3&2&7&4&0&2&3&0&1&2&3&0&0&1&0&0&1&0\\
0&4&0&0&5&5&0&7&2&7&0&1&0&0&1&2&3&0&0&0&0&0&0&0\\
0&1&2&1&4&7&1&0&0&0&4&1&3&0&0&0&3&4&0&0&0&1&1&0\\
2&1&0&5&1&1&4&9&0&0&1&4&0&0&1&2&0&0&0&1&0&0&1&0\\
0&0&1&0&0&2&0&1&7&0&0&1&3&0&1&2&3&4&0&0&0&0&0&0\\
4&0&4&1&5&0&0&0&0&0&0&1&0&0&0&0&3&4&1&0&0&0&0&0\\
0&1&0&0&1&0&0&1&0&4&1&0&3&0&1&2&0&0&0&0&0&0&1&0
\end{bmatrix}\ [\text{s}],$$

należy wyznaczyć zbiór $\boldsymbol{\mathcal{X}}^{\mathrm{ND}}_{\preccurlyeq}$ *Pareto*-optymalnych strategii dla problemu optymalizacji wielokryterialnej zadanego uporządkowaną trójką:

$$(\widehat{\boldsymbol{\mathcal{X}}}, F, \rho^{\preccurlyeq}), \tag{5.105}$$

1) $\widehat{\boldsymbol{\mathcal{X}}}$ – zbiór konfiguracji dopuszczalnych zdefiniowany, jak w zadaniu (4.66);

2) F – kryterium wektorowe:

$$F : \boldsymbol{\mathcal{X}} \to \mathbb{R}^2,$$

gdzie $F(x) = [\hat{Z}_{\max}(x), \widetilde{Z}_{\max}(x)]^T = y \in \mathbb{R}^2$ dla $x \in \widehat{\boldsymbol{\mathcal{X}}}$;

3) $\rho^{\leqslant}$ – relacja dominowania w przestrzeni $\mathbb{R}^2$.

Rozważany problem cechuje się 306 binarnymi zmiennymi decyzyjnymi, a zero-jedynkowa przestrzeń przeszukiwań zawiera $1{,}3 \cdot 10^{92}$ elementów. Całkowitoliczbowa przestrzeń przeszukiwań zawiera $7{,}98 \cdot 10^{36}$ rozwiązań spełniających wymagania formalne. Ze względu na uwzględnienie dodatkowych wymagań interesariusza nie wszystkie elementy całkowitoliczbowej przestrzeni reprezentują rozwiązania dopuszczalne.

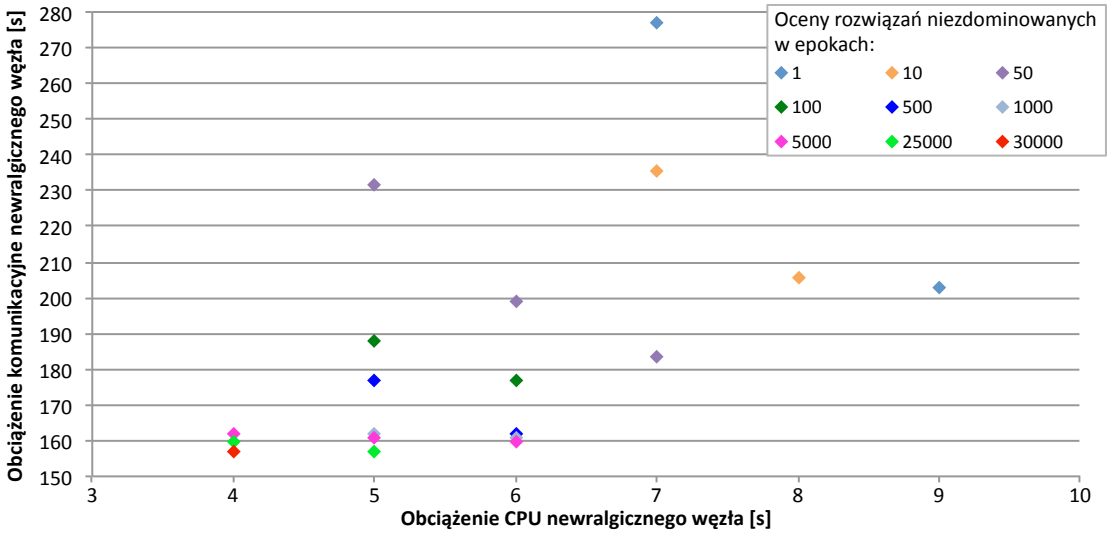
Do wyznaczenia rozwiązań zagadnienia (5.105) *Paluszak* zastosował algorytm harmoniczny (ang. *harmony search* – HS) [209]. W czasie około 95 minut wykonano 2×10^6 aktualizacji pamięci harmonicznej o rozmiarze 20 rozwiązań na komputerze wyposażonym w procesor klasy Intel Core i7-2670QM 2,20 GHz. W efekcie wyznaczono dwa rozwiązania niezdominowane x_1^{HS} i x_2^{HS}, które spełniają wszystkie ograniczenia i cechują się ocenami: $F(x_1^{\mathrm{HS}}) = (4\text{ s}; 178\text{ s})$ oraz $F(x_2^{\mathrm{HS}}) = (5\text{ s}; 157\text{ s})$.

Natomiast programowanie genetyczne dla populacji składającej się z 20 osobników przetworzyło 30000 generacji w czasie około 50 minut, przy czym symulacja odbywała się na komputerze klasy serwerowej wyposażonym w dwa procesory Intel Xeon E5640 2,67GHz. Przetworzenie jednej generacji w programowaniu genetycznym wymaga więcej czasu niż w przypadku algorytmów ewolucyjnych, które operują na fenotypowej reprezentacji osobników. Operacje na drzewach programów są bardziej kosztowne niż operacje na liczbowych zmiennych decyzyjnych. Ponadto zastosowana procedura *interpretacja* wymaga $MV_W + (V + J)I$ drzew do wyznaczenia całkowitoliczbowego rozwiązania X, podczas gdy fenotypowa reprezentacja wymaga $M + V + I$ zmiennych decyzyjnych.

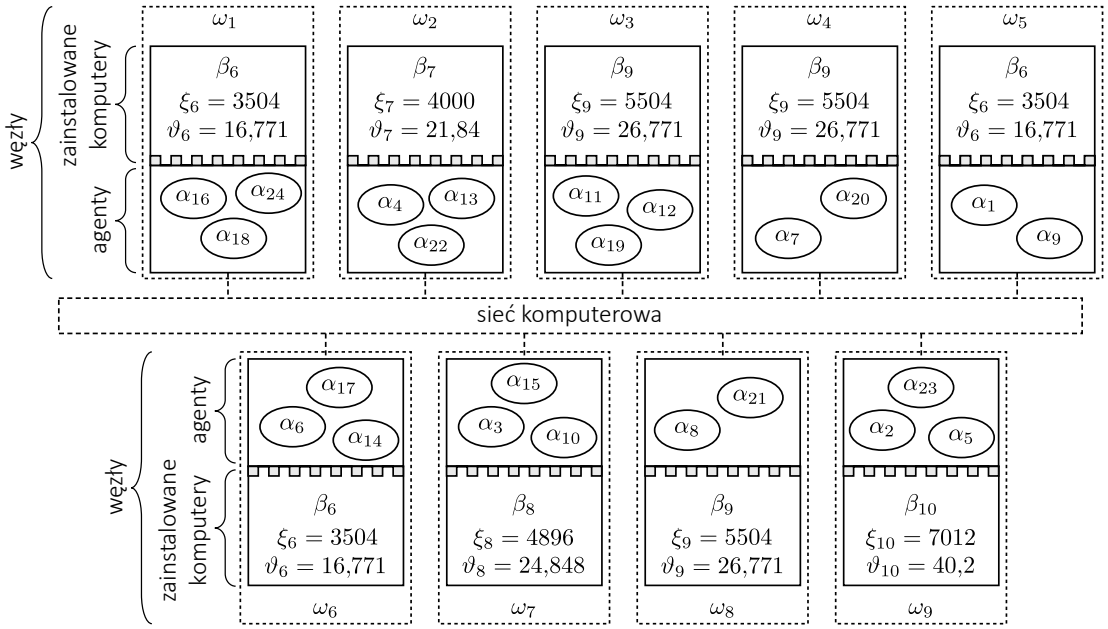
Pomimo mniejszej liczby analizowanych osobników programowanie genetyczne wyznaczyło rozwiązanie x_1^{GP} o ocenie $F(x_1^{\mathrm{GP}}) = (4\text{ s}; 157\text{ s})$. Rozwiązanie x_1^{GP} dominuje rozwiązania wyznaczone przez algorytm harmoniczny. Kolejne przybliżenia frontu *Pareto* uzyskane na różnych etapach programowania genetycznego przedstawiono na rysunku 5.25. Strategia x_1^{GP} jest pojedynczym rozwiązaniem niezdominowanym zwróconym po zakończeniu algorytmu MOGPA.

W tabeli nr 5.4 pokazano kolejne etapy dostrajania strategii prowadzące do uzyskania końcowego rozwiązania x_1^{GP}. Wyróżniono zmienne decyzyjne, które podlegały modyfikacji pomiędzy epokami. Pominięto wektor X^z, który opisuje przypisanie zadań do agentów początkowych W, ponieważ założono, że realizowane jest jedno zadanie, które angażuje wszystkie agenty gridu. Warto zauważyć, że poprawa oceny rozwiązania wymagała każdorazowo modyfikacji więcej niż jednej zmiennej decyzyjnej strategii.

Rozmieszczenie agentów i zasobów w gridzie zgodne z wynikową strategią x_1^{GP} pokazano na rysunku 5.26. Uzyskane rozwiązanie cechuje się parametrami: $\Theta(x_1^{\mathrm{GP}}) = 217{,}52$ TFLOPS, $\eta(x_1^{\mathrm{GP}}) = 255$, $\Xi(x_1^{\mathrm{GP}}) = 42932$ USD, $E(x_1^{\mathrm{GP}}) = 31{,}65$ kW, co oznacza, że spełniono wszystkie ograniczenia. Na rysunku 5.26 widoczny jest wysoki stopień rozproszenia agentów, który ma odzwierciedlenie w ocenie względem kryterium η.



Rysunek 5.25: Zbieżność programowania genetycznego w zagadnieniu dwukryterialnym
Źródło: opracowanie własne



Rysunek 5.26: Rozmieszczenie agentów i zasobów w gridzie zgodne ze strategią x_1^{GP}
Źródło: opracowanie własne

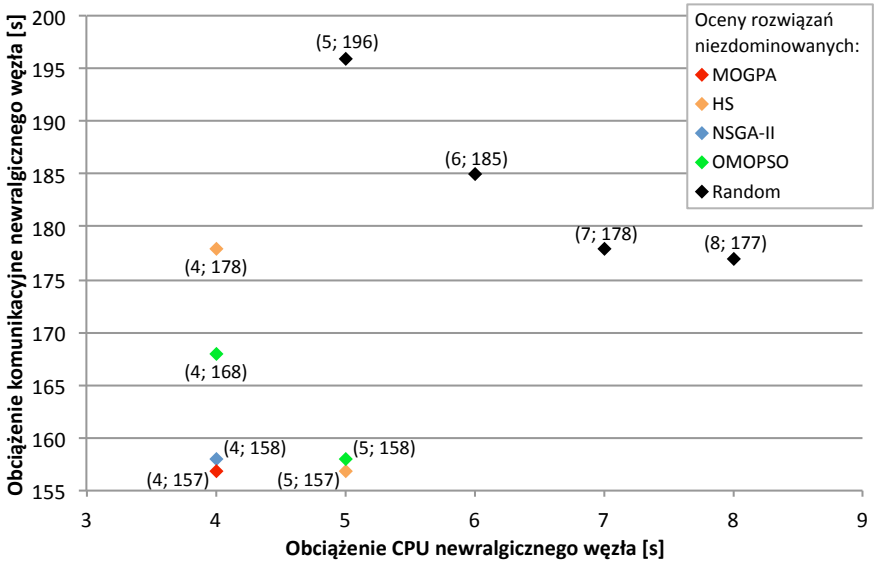
Tabela 5.4: Dostrajanie strategii w toku programowania genetycznego

Epoka	$\hat{Z}_{\max}(x)$	$\widetilde{Z}_{\max}(x)$	X^{α}	X^{β}
1000	5	162	[5, 9, 8, 2, 7, 6, 9, 8, 5, 4, 3, 3, 2, 6, 6, 1, 7, 1, 3, 4, 8, 2, 9, 1]	[10, 9, 9, 4, 4, 9, 8, 9, 7]
1500	4	162	[5, 9, 8, 2, 7, 6, 9, 8, 5, 4, 3, 3, 2, 6, 6, 1, 7, 1, 3, 4, 8, 2, 9, 1]	[10, **7**, 9, 4, 4, **6**, 8, 9, 7]
20000	4	161	[5, 9, 8, 2, **9**, 6, **4**, 8, 5, **7**, 3, 3, 2, 6, 6, 1, 7, 1, 3, 4, 8, 2, 9, 1]	[10, 7, 9, **9**, **6**, 6, 8, 9, 7]
25000	4	160	[5, 9, 8, 2, 9, 6, 4, 8, 5, 7, 3, 3, 2, 6, **7**, 1, **6**, 1, 3, 4, 8, 2, 9, 1]	[10, 7, 9, 9, 6, 6, 8, 9, 7]
30000	4	157	[5, 9, **7**, 2, 9, 6, 4, 8, 5, 7, 3, 3, 2, 6, 7, 1, 6, 1, 3, 4, 8, 2, 9, 1]	[**6**, 7, 9, 9, 6, 6, 8, 9, **10**]

Źródło: opracowanie własne

Wyniki uzyskane za pomocą programowania genetycznego porównano dodatkowo z rezultatami otrzymanymi za pomocą algorytmu ewolucyjnego NSGA-II [62] oraz wielokryterialnej wersji algorytmu roju cząstek OMOPSO [242]. Wykorzystano implementacje tych metaheurystyk dostępne w ramach pakietu oprogramowania *MOEA Framework* [113]. Metody NSGA-II i OMOPSO operowały na reprezentacjach osobników w postaci wektorów liczb rzeczywistych o $M+V+I$ elementach. W testach zastosowano rozmiar populacji na poziomie $G_{\text{size}} = 20$ osobników i limit $G_{\max} = 30000$ epok, co oznacza, że liczba wywołań funkcji oceny była taka sama, jak w wykonanym wcześniej przebiegu programowania genetycznego.

Na rysunku 5.27 porównano oceny strategii wyznaczonych przez algorytmy MOGPA, NSGA-II, OMOPSO oraz HS. Dodatkowo uwzględniono oceny rozwiązań wyznaczonych metodą losową po wygenerowaniu 600000 strategii. Widoczne jest, że wszystkie porównywane metaheurystyki wnoszą istotną poprawę wyników w stosunku do losowego przeszukiwania przestrzeni rozwiązań. Należy przy tym zwrócić uwagę, że algorytm HS wykonał większą liczę wywołań funkcji oceny.



Rysunek 5.27: Porównanie rozwiązań wyznaczonych przez wybrane metaheurystyki
Źródło: opracowanie własne

Strategia wyznaczona przez algorytm MOGPA dominuje pozostałe rozwiązania. Algorytm NSGA-II wyznaczył strategię o zbliżonej ocenie (4 s, 158 s), a kontynuując jego działanie poza początkowy limit 30000 epok udało się uzyskać rozwiązanie o cenie (4 s, 157 s) w epoce nr 45075. Dalsza ewolucja do zwiększonego limitu 60 tys. epok nie przyniosła poprawy wyników. Algorytmy OMOPSO oraz HS wyznaczyły rozwiązania o gorszych ocenach.

Algorytm MOGPA dostrojono następująco: drzewa populacji początkowej cechowały się wysokością w przedziale 6-12, prawdopodobieństwo krzyżowania wynosiło 80%, przy czym jako punkt krzyżowania w 90% przypadków wybierano węzeł wewnętrzny drzewa, a w 10% przypadków – liść. Mutacja węzła następowała z 5% prawdopodobieństwem, podobnie jak mutacja polegająca na zastąpieniu poddrzewa nowym losowo wygenerowanym poddrzewem. Ponadto zastosowano procedurę nadawania rang *Goldberga* na etapie oceny osobników. Rozmiar archiwum ustalono na

poziomie $\Lambda = 15$ osobników, intensywność weryfikacji populacji wynosiła $\lambda_{\text{size}} = 5$, a jej interwał $\lambda_{\text{epoch}} = 5$ epok. Wybrano miarę zagęszczenia DGCD do redukcji nadmiarowych rozwiązań z archiwum. Podczas działania programowania genetycznego rozmiar archiwum utrzymywał się poniżej limitu $\Lambda = 15$.

W kolejnym eksperymencie zrezygnowano z ograniczenia na wydajność gridu – selektor $\mu_5 = 0$. Odpowiada to sytuacji, gdy interesariusz po otrzymaniu możliwych strategii nie jest usatysfakcjonowany uzyskanymi kompromisami i decyduje o rozluźnieniu najmniej istotnego wymagania, licząc na poprawę ocen pod względem innych kryteriów. Pozostałe dane wejściowe są takie, jak w zagadnieniu (5.105), co prowadzi do sformułowania drugiego testowego problemu optymalizacji:

Przy nieaktywnym selektorze wymagania na minimalną wydajność gridu $\mu_5 = 0$ i pozostałych danych jak w zagadnieniu (5.105)
należy wyznaczyć zbiór $\boldsymbol{\mathcal{X}}^{\text{ND}}_{\preccurlyeq}$ *Pareto*-optymalnych strategii agentów dla problemu optymalizacji wielokryterialnej zadanego uporządkowaną trójką:

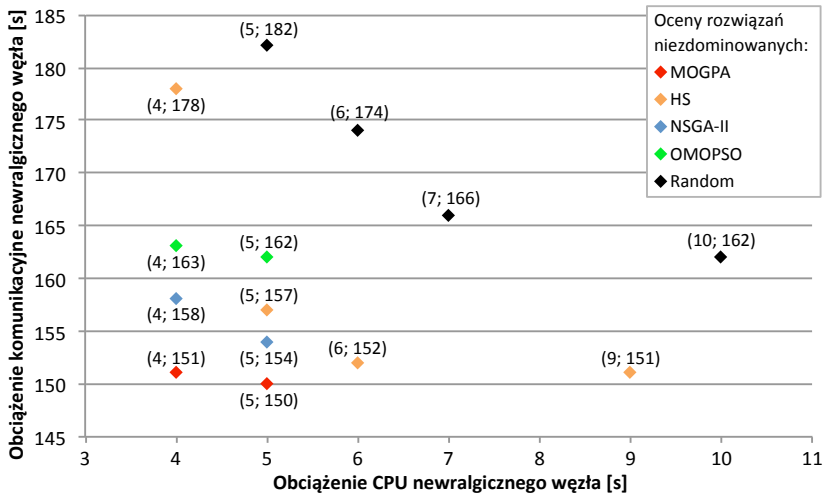
$$(\widehat{\boldsymbol{\mathcal{X}}}, F, \rho^{\preccurlyeq}). \tag{5.106}$$

1) $\widehat{\boldsymbol{\mathcal{X}}}$ – zbiór konfiguracji dopuszczalnych zdefiniowany, jak w zagadnieniu (4.66);

2) F – kryterium wektorowe, jak w zagadnieniu (5.105);

3) $\rho^{\preccurlyeq}$ – relacja dominowania w przestrzeni $\mathbb{R}^2$.

Do rozwiązania zmodyfikowanego problemu optymalizacji (5.106) zastosowano te same metaheurystyki, co w poprzednim eksperymencie. Zachowano również wartości dostrojonych parametrów. Porównanie uzyskanych wyników przedstawiono na rysunku 5.28. Spośród rozwiązań wyznaczonych przez algorytm harmoniczny, cztery strategie spełniają rozluźnione wymagania. Powoduje to szerszą reprezentację przybliżonego frontu *Pareto*.



Rysunek 5.28: Porównanie ocen strategii wyznaczonych po rozluźnieniu wymagań interesariusza
Źródło: opracowanie własne

Algorytm NSGA-II wyznaczył dwa rozwiązania o ocenach (4 s, 158 s) oraz (5 s, 154 s). Pierwsze z nich odpowiada wynikowi uzyskanemu w poprzednim eksperymencie. Druga strategia rozszerza front niezdominowany w kierunku obszaru, który wcześniej zawierał rozwiązania niedopuszczalne ze względu na bardziej restrykcyjne wymagania interesariusza. Metoda OMOPSO poprawiła jedną z ocen uzyskanych w poprzednim eksperymencie – z (4 s, 168 s) na (4 s, 163 s) – jednak pogorszeniu uległa druga ocena: z (4 s, 158 s) na (5 s, 162 s). Równocześnie nie zostało wyznaczone żadne rozwiązanie o ocenie, która nie była możliwa do uzyskania przy oryginalnym zestawie ograniczeń.

Algorytm MOGPA wyznaczył dwa nowe rozwiązania x_1^{GP} i x_2^{GP} cechujące się ocenami: $F(x_1^{\mathrm{GP}}) = (4\text{ s}, 151\text{ s})$ i $F(x_2^{\mathrm{GP}}) = (5\text{ s}, 150\text{ s})$. Żadne z tych rozwiązań nie spełnia ograniczeń poprzedniego zagadnienia optymalizacji (5.105), o czym świadczą oceny: $\Theta(x_1^{\mathrm{GP}}) = 185{,}591$ oraz $\Theta(x_2^{\mathrm{GP}}) = 181{,}80$. Przed rozluźnieniem wymagań obowiązywało ograniczenie: $\vartheta_{\min} = 200$. Widoczne jest, że rozluźnienie ograniczeń pozwoliło na wyznaczenie rozwiązań cechujących się lepszymi ocenami pod względem kryteriów optymalizacji.

Rozwiązanie x_1^{GP} odpowiada strategii: $X^z = [4]$, $X^\alpha = [6, 8, 9, 3, 9, 2, 4, 5, 1, 4, 1, 3, 8, 2, 8, 7, 2, 6, 9, 7, 5, 6, 3, 7]$, $X^\beta = [5, 6, 5, 9, 1, 9, 6, 10, 9]$. Natomiast rozwiązanie x_2^{GP} reprezentuje strategię: $X^z = [3]$, $X^\alpha = [1, 1, 1, 4, 6, 7, 5, 9, 8, 4, 9, 8, 2, 1, 2, 2, 7, 3, 3, 6, 5, 7, 6, 5]$, $X^\beta = [10, 8, 1, 7, 3, 9, 7, 4, 8]$. Warto zauważyć, że choć oceny obu rozwiązań są zbliżone, to odpowiadające im strategie różnią się znacząco. Podobnie jak poprzednio, rozwiązania wyznaczone przez programowanie genetyczne dominują strategie wyznaczone za pomocą pozostałych metaheurystyk.

W kolejnym eksperymencie rozważa się modernizację instancji laboratoryjnej systemu *Comcute*, która jest eksploatowana na Politechnice Gdańskiej. Dla porównania wyników z rezultatami algorytmu harmonicznego wykorzystano plan rozbudowy opracowany przez *Paluszaka* [209]. Przewiduje się zwiększenie liczby węzłów grida z 9 do 15 oraz zwiększenie liczby agentów z obecnych 6 ($2W+4S$) do 45 ($15W + 30S$). Koszt zakupu nowych komputerów nie może przekroczyć $\xi_{\max} = 350000$ zł, a zapotrzebowanie energetyczne systemu $\varepsilon_{\max} = 25$ kW. Brane pod uwagę są komputery zestawione w tabeli nr 3.1. Dodatkowo uwzględniane są ograniczenia na minimalny stopień rozproszenia oraz minimalną łączną moc obliczeniową systemu.

W eksperymencie *Paluszaka* rozważana była wersja gridu *Comcute* sprzed modernizacji oprogramowania warstwy pośredniczącej. W efekcie rozpatrywane obciążenia odbiegają od wartości przedstawionych w rozdziale 3.3 dla aktualnej wersji systemu. Aby porównanie wyników optymalizacji było możliwe, w programowaniu genetycznym wykorzystano czasy, które zastosowano dla algorytmu harmonicznego.

Badany przypadek testowy dotyczy przebiegu, dla którego zmierzono następujące obciążenia: $t_{v1} = 320{,}288$ s, dla $v = \overline{1, V_W}$, $t_{v1} = 306{,}83$ s, dla $v = \overline{V_W + 1, V}$, $\tau_{vu} = 27{,}602$ s oraz $\tau_{uv} = 337{,}926$ s dla $v = \overline{1,V_W}$ i $u = \overline{V_W + 1,V}$. Skumulowane czasy obciążenia CPU podano dla komputera typu β_1. Pozostałe wiersze macierzy T oszacowano w oparciu o wyniki benchmarku *CPU Mark*, które przedstawiono w tabeli nr 3.1. W efekcie sformułowano następujące zagadnienie dwukryterialnej optymalizacji kompromisowej obciążenia CPU i obciążenia komunikacyjnego newralgicznych węzłów.

Przy zadanych selektorach ograniczeń:

- $\mu_5 = 1$ – minimalna wydajność gridu $\vartheta_{\min} = 150000$ punktów benchmarku *CPU Mark*,
- $\mu_6 = 1$ – maksymalny koszt zakupu komputerów $\xi_{\max} = 350000$ zł,
- $\mu_8 = 1$ – minimalny stopień rozproszenia agentów $\eta_{\min} = 500$,
- $\mu_{15} = 1$ – maksymalne zapotrzebowanie energetyczne $\varepsilon_{\max} = 25$ kW,
- $\mu_{18} = 1$ – ograniczenie na liczbę dostępnych komputerów poszczególnych typów,

oraz danych wejściowych:
$M = 1$, $V = 45$, $V_W = 15$, $I = 15$, $J = 12$ (parametry komputerów w tabeli nr 3.1), $T = [t_{vj}]$, $\tau = [\tau_{vu}]$, przy czym:

$$t_{vj} = \begin{cases} 320{,}288, & \text{dla } v = \overline{1, V_W},\ j = 1, \\ 306{,}83, & \text{dla } v = \overline{V_W + 1, V},\ j = 1, \\ \dfrac{\vartheta_1}{\vartheta_j} t_{v,1}, & \text{dla } v = \overline{1,V},\ j > 1, \end{cases}$$

$$\tau_{vu} = \begin{cases} 27{,}602, & \text{dla } v = \overline{1,V_W},\ u = \overline{V_W + 1,V}, \\ 337{,}926, & \text{dla } v = \overline{V_W + 1,V},\ u = \overline{1,V_W}, \\ 0 & \text{w pozostałych przypadkach}, \end{cases}$$

należy dla kryteriów optymalizacji $\hat{Z}_{\max}$ i $\widetilde{Z}_{\max}$, $\boldsymbol{\mathcal{N}} = \{1, 2\}$ oraz punktów $y^{\inf}$, $y^{\sup}$ wyznaczyć rozwiązanie kompromisowe $x^{p=2}$, takie że:

$$\phi_2(x^{p=2}) = \min_{x \in \widehat{\boldsymbol{\mathcal{X}}}} \sqrt{\sum_{n \in \boldsymbol{\mathcal{N}}} \left(\frac{y_n(x) - y_n^{\inf}}{y_n^{\sup} - y_n^{\inf}} \right)^2}, \tag{5.107}$$

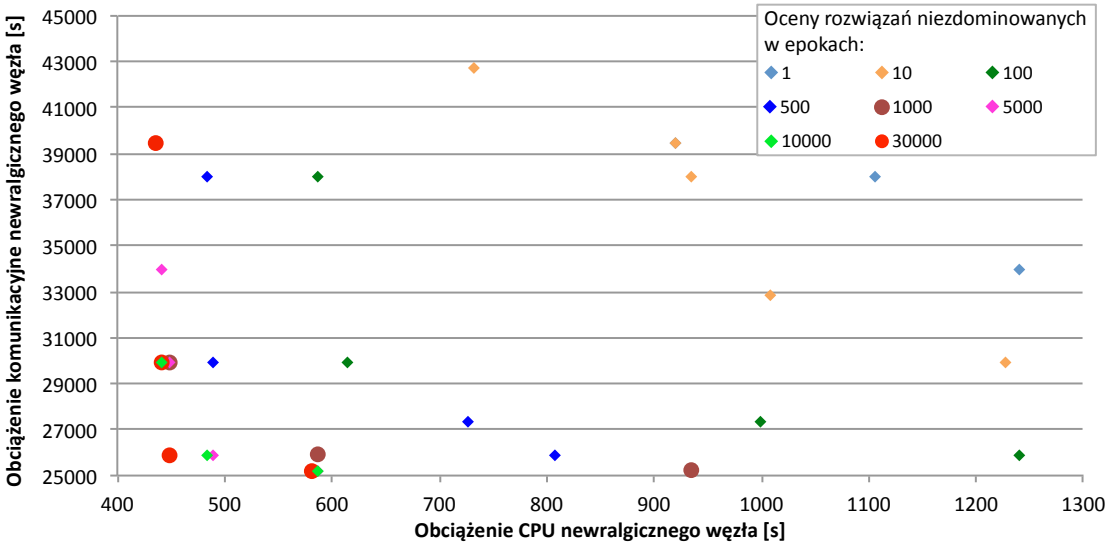
gdzie $\widehat{\boldsymbol{\mathcal{X}}}$ – zbiór rozwiązań dopuszczalnych, zdefiniowany w (4.66).

W problemie (5.107) występuje 855 binarnych zmiennych decyzyjnych, a przestrzeń przeszukiwań dla kodowania zero-jedynkowego zawiera $2{,}4 \cdot 10^{257}$ rozwiązań. W przypadku kodowania całkowitoliczbowego występuje 60 zmiennych decyzyjnych, a przestrzeń przeszukiwań zawiera $1{,}3 \cdot 10^{69}$ elementów.

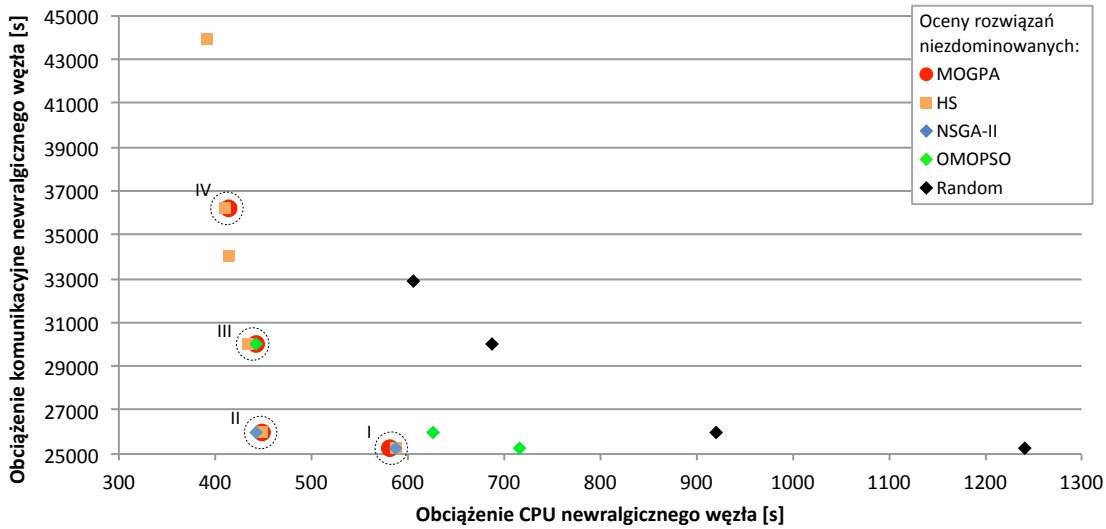
Do wyznaczenia rozwiązania zagadnienia (5.107) wykorzystano metodę CMGP, w której rozwiązanie kompromisowe wybierane jest ze zbioru niezdominowanego uzyskanego za pomocą metody MOGPA. Na rysunku 5.29 przedstawiono przybliżenia frontu *Pareto* uzyskane na różnych etapach algorytmu MOGPA. Widoczne jest, że następuje poprawa wyników wraz z upływem epok ewolucji aż do osiągnięcia limitu $G_{\max} = 30000$ generacji. Świadczy to o zbieżności metody w rozpatrywanym problemie optymalizacji. Rozmiar populacji ustalono na poziomie $G_{\text{size}} = 30$ osobników, a pozostałe parametry dostrojono, jak w poprzednim eksperymencie.

Końcowe wyniki wyznaczone przez algorytm MOGPA porównano z rezultatami otrzymanymi za pomocą algorytmu HS oraz algorytmów NSGA-II i OMOPSO. Efekty porównania zobrazowano na rysunku 5.30. Rezultaty czterech metaheurystyk są zbliżone. Trzy spośród porównywanych algorytmów – MOGPA, HS i NSGA-II – wyznaczyły przynajmniej jedno rozwiązanie, które nie

jest dominowane przez żadne z rozwiązań otrzymanych za pomocą innych algorytmów. Najwyższej jakości front *Pareto* można uzyskać, łącząc wyniki z porównywanych metod.



Rysunek 5.29: Oceny rozwiązań niezdominowanych na różnych etapach programowania genetycznego
Źródło: opracowanie własne



Rysunek 5.30: Porównanie wyników wyznaczonych za pomocą programowania genetycznego z rezultatami otrzymanymi za pomocą innych metaheurystyk
Źródło: opracowanie własne

Warto zauważyć, że badane algorytmy posiadają wspólne obszary zbieżności, które oznaczono liczbami rzymskimi na rys. 5.30. Oceny rozwiązań wyznaczonych w tych obszarach przedstawiono w tabeli nr 5.5. W obszarze I algorytm MOGPA należy ocenić wyżej, gdyż wyznaczył rozwiązanie o ocenie (581 s, 25221 s) w porównaniu do wyniku (587 s, 25221 s) z algorytmów NSGA-II i HS. Poprawiona została ocena pod względem kryterium $\hat{Z}_{\max}(x)$ bez pogorszenia wyniku dla $\widetilde{Z}_{\max}(x)$. W obszarze II wyżej należy ocenić algorytm NSGA-II, z kolei w obszarze III – algorytm HS.

Tabela 5.5: Wspólne obszary zbieżności algorytmów MOGPA, HS, NSGA-II i OMOPSO

Obszar	Oceny rozwiązań $F(x) = [\hat{Z}_{\max}(x), \widetilde{Z}_{\max}(x)]^T$			
	MOGPA	NSGA-II	OMOPSO	HS
I	x_1^{GP}: (581; 25221)	(587, 25221)	–	(587, 25221)
II	x_2^{GP}: (448; 25952)	(442; 25952)	–	(448; 25952)
III	x_3^{GP}: (442; 29973)	–	(442; 29973)	(435; 29973)
IV	x_4^{GP}: (414; 36187)	–	–	(411; 36187)

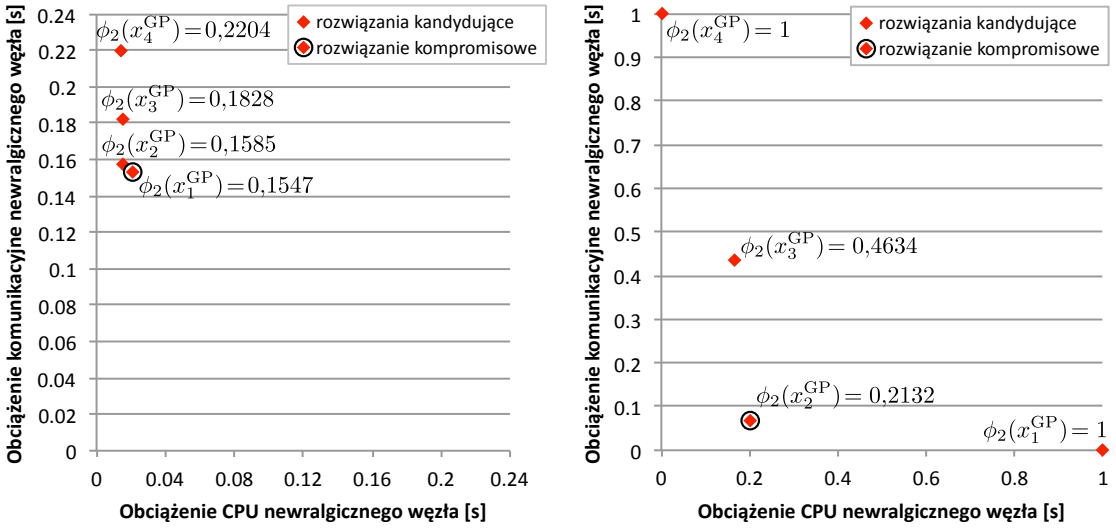
Źródło: opracowanie własne

Najszerszą reprezentację frontu *Pareto* uzyskano za pomocą algorytmu HS, który wyznaczył 6 strategii niezdominowanych. Algorytm MOGPA wskazał 4 rozwiązania, a OMOPSO – 3 rozwiązania, przy czym dwa z nich cechują się znacząco gorszą jakością w porównaniu do pozostałych wyników. Algorytm NSGA-II wyznaczył dwa rozwiązania. Należy przy tym zauważyć, że algorytm harmoniczny wykonał większą liczbę wywołań procedury oceny konfiguracji, realizując 10^6 aktualizacji pamięci harmonicznej o rozmiarze 30 osobników. Dla pozostałych metod rozmiar populacji $G_{\mathrm{size}} = 30$ i limit epok $G_{\max} = 30000$ były takie same.

W celu modernizacji grida *Comcute* konieczne jest wybranie jednego rozwiązania kompromisowego ze zbioru strategii wyznaczonego przez algorytm MOGPA. Na rysunku 5.31 pokazano rozmieszczenie ocen uzyskanych rozwiązań w przestrzeni kryterialnej znormalizowanej względem punktów $y^{\inf}$ i $y^{\sup}$. Rozważać można normalizację względem punktów charakterystycznych, których współrzędne obliczono w oparciu o oszacowania teoretyczne omówione w rozdziale 5.3 (rys. 5.31a). Alternatywnie współrzędne punktów $y^{\inf}$ i $y^{\sup}$ można wyznaczyć jako kresy dolne i górne zbioru ocen rozwiązań niezdominowanych, co prowadzi do normalizacji, jak na rysunku 5.31b.

Widoczne jest, że sposób doboru punktów charakterystycznych ma wpływ na wybór rozwiązania kompromisowego. W przypadku wykorzystania oszacowań teoretycznych uzyskano punkty $y^{\inf} = (71{,}99\text{ s}, 0\text{ s})$ i $y^{\sup} = (25245{,}11\text{ s}, 164487{,}6\text{ s})$. Najmniejszą odległością od punktu idealnego w znormalizowanej przestrzeni kryterialnej – $\phi_2(x_1^{\mathrm{GP}}) = 0{,}1547$ na rysunku 5.31a – cechuje się strategia: $X^z = [8]$, $X^\alpha = [9, 10, 6, 4, 4, 1, 9, 3, 2, 14, 7, 11, 7, 12, 15, 5, 2, 6, 2, 3, 3, 5, 12, 6, 8, 12, 15, 8, 13, 15, 8, 12, 13, 6, 1, 8, 13, 10, 5, 15, 5, 13, 11, 14, 11]$, $X^\beta = [3, 3, 3, 3, 3, 8, 5, 10, 3, 10, 3, 10, 3, 3, 8]$. Strategia cechuje się oceną $(581\text{ s}, 25221\text{ s})$ w przestrzeni kryterialnej. Jest to równocześnie rozwiązanie, które nie zostało zdominowane przez żaden wynik uzyskany za pomocą innych testowanych algorytmów.

Punkty charakterystyczne, które wyznaczono teoretycznie, nie spełniają ograniczeń z zagadnienia optymalizacji. W szczególności uzyskanie zerowego czasu komunikacji (druga współrzędna punktu $y^{\inf}$) wymaga ulokowania wszystkich agentów w jednym węźle, co jest sprzeczne z wymaganiem na minimalny stopień rozproszenia $\eta_{\min} = 500$. Punkty wyznaczone w oparciu o oceny niezdominowanych rozwiązań dopuszczalnych to $y^{\inf} = (414\text{ s}, 25221\text{ s})$ i $y^{\sup} = (581\text{ s}, 36187\text{ s})$. Przy ich zastosowaniu wybrano rozwiązanie cechujące się odległością $\phi_2(x_2^{\mathrm{GP}}) = 0{,}2132$ od punktu idealnego w przestrzeni znormalizowanej (rys. 5.31b).



(a) Po normalizacji względem punktów charakterystycznych oszacowanych teoretycznie

(b) Po normalizacji względem punktów wyznaczonych w oparciu o oceny rozwiązań niezdominowanych

Rysunek 5.31: Wybór rozwiązania kompromisowego ze zbioru *Pareto*
Źródło: opracowanie własne

Wybrane rozwiązanie kompromisowe cechuje się oceną $F(x_2^{GP}) = (448\text{ s}, 25\,952\text{ s})$ i odpowiada mu strategia: $X^z = [8]$, $X^\alpha = [13, 10, 6, 7, 4, 1, 9, 6, 2, 14, 7, 11, 3, 12, 2, 10, 14, 9, 11, 4, 15, 5, 12, 15, 3, 12, 6, 9, 13, 2, 8, 12, 1, 13, 1, 8, 3, 10, 5, 10, 5, 11, 11, 8, 15]$, $X^\beta = [5, 5, 5, 8, 5, 5, 11, 5, 5, 12, 11, 11, 5, 5, 12]$. Pozostałe parametry rozwiązania przedstawiają się następująco: $\Theta(x_2^{GP}) = 468\,786$ punktów wg benchmarku *CPU Mark*, $\Xi(x_2^{GP}) = 341\,112$ zł, $\eta(x_2^{GP}) = 952$, $E(x_2^{GP}) = 19{,}695$ kW. Warto zauważyć, że algorytm harmoniczny wytypował do wdrożenia w gridzie *Comcute* rozwiązanie o takiej samej ocenie (448 s, 25 952 s), ale przy innym rozdziale zasobów [209].

W kolejnym eksperymencie zastosowano metodę DMGP w celu zaprojektowania programu do wyznaczania strategii sieci agentów w gridzie, który realizuje dynamiczny scenariusz wykonania zadań przy zmiennej mocy obliczeniowej oferowanej przez wolontariuszy. Scenariusz składa się z $Q = 5$ etapów. W toku jego realizacji zlecono wykonanie 6 zadań obliczeniowych, które różnią się liczbą paczek danych wejściowych oraz momentem rozpoczęcia obliczeń. Niech przetwarzanie danych wejściowych odbywa się w węzłach obliczeniowych wolontariuszy, których liczba zmienia się w czasie. W tabeli nr 5.6 zamieszczono liczbę pozostałych do przetworzenia paczek danych wejściowych i liczbę aktywnych wolontariuszy na poszczególnych etapach scenariusza.

Przyjęto, że średni czas dostępności wolontariusza $\mathscr{T}_a = 16\,632$ s, a średni czas niedostępności $\mathscr{T}_u = 47\,484$ s. Parametry te charakteryzują najliczniejszą frakcję wśród wolontariuszy systemu *SETI@home* [127, 257], który jest jednym z najpopularniejszych gridów wolontariackich. Ponadto założono, że: czas obsługi paczki danych $P_T(z_m) = 5$ s, stopień replikacji $P_R(z_m) = 1$ oraz liczba agentów typu W uczestniczących w realizacji zadania $P_W(z_m) = 20$ dla wszystkich zadań obliczeniowych z_m, $m = \overline{1,6}$.

Tabela 5.6: Etapy scenariusza realizacji zadań

q	Liczba pozostałych paczek danych $L(z_m)$						Wolontariusze		
	z_1	z_2	z_3	z_4	z_5	z_6	Liczba	$\mathscr{T}_a$ [s]	$\mathscr{T}_u$ [s]
1	10^7	–	–	–	–	–	1000	16632	47484
2	5×10^6	10^7	$7{,}5 \times 10^6$	–	–	–	1500		
3	0	5×10^6	$2{,}5 \times 10^6$	$2{,}5 \times 10^6$	–	–	1500		
4	0	$2{,}5 \times 10^6$	0	0	5×10^6	5×10^6	1100		
5	0	0	0	0	$2{,}5 \times 10^6$	$2{,}5 \times 10^6$	1000		

Źródło: opracowanie własne

Ze względu na liczbę paczek danych w zadaniach obliczeniowych, rozważa się rozbudowę grida *Comcute* do 20 węzłów oraz zwiększenie liczby agentów z dotychczasowych 6 do 60 (20 agentów typu W + 40 agentów typu S). Liczba węzłów jest ograniczona liczbą wolnych miejsc w szafach serwerowych wykorzystywanych na potrzeby instancji laboratoryjnej systemu. Wybór komputerów następuje spośród systemów przedstawionych w tabeli nr 3.1. Należy przy tym zwrócić uwagę, że zakupione komputery wykorzystywane są w całym scenariuszu. Dane wejściowe $\boldsymbol{\mathcal{Z}}_q^{\mathrm{IN}}$ dla etapów $q = \overline{2{,}5}$ obejmują wyłącznie rodzaje komputerów wybrane w etapie $q = 1$.

Macierze skumulowanych czasów przetwarzania danych T oraz czasów komunikacji τ wyznacza się dla każdego etapu scenariusza w oparciu o liczbę aktywnych zadań i pozostałych do przetworzenia paczek danych. Wartości obciążeń szacowane są zgodnie z zależnościami, które wyznaczono na podstawie pomiarów w gridzie *Comcute* (tabela nr 3.4). W efekcie zdefiniować można sekwencję stanów $\boldsymbol{\mathcal{Z}}_1^{\mathrm{IN}}, \ldots, \boldsymbol{\mathcal{Z}}_q^{\mathrm{IN}}, \ldots, \boldsymbol{\mathcal{Z}}_5^{\mathrm{IN}}$, która opisuje rozpatrywany scenariusz wykonania zadań.

Zagadnienie optymalizacji programu do wyznaczania strategii sieci agentów w dynamicznym środowisku grida *Comcute* odnosi się do równoczesnej minimalizacji trzech kryteriów na każdym etapie scenariusza: obciążenia procesorów węzła newralgicznego pod tym względem, obciążenia komunikacyjnego węzła newralgicznego pod względem komunikacji oraz kosztu wykonania zadań obliczeniowych. Koszty pracy komputerów poszczególnych typów oszacowano w oparciu o ich zapotrzebowanie energetyczne i stawkę dostawcy energii elektrycznej na poziomie $c_\varepsilon = 0{,}2376$ zł/kWh. W efekcie sformułowano następujące zagadnienie optymalizacji:

Dla scenariusza opisanego sekwencją stanów $\boldsymbol{\mathcal{Z}}_1^{\mathrm{IN}}, \ldots, \boldsymbol{\mathcal{Z}}_q^{\mathrm{IN}}, \ldots, \boldsymbol{\mathcal{Z}}_5^{\mathrm{IN}}$ oraz kryterium wektorowego $F(x) = [\hat{Z}_{\max}(x), \widetilde{Z}_{\max}(x), K(x)]$, należy wyznaczyć program do specyfikacji strategii χ^*, taki że:

$$\phi_2^{\mathrm{suma}}(\chi^*) = \min_{\chi \in \widehat{\boldsymbol{\mathcal{X}}}_{\boldsymbol{\mathcal{P}}}} \phi_2^{\mathrm{suma}}(\chi). \tag{5.108}$$

Algorytm DMGP wyznacza program χ^* w oparciu o odległości ocen wyznaczanych przez niego strategii od punktów idealnych na poszczególnych etapach scenariusza. Wykorzystano oszacowania teoretyczne dla współrzędnych punktów $y^{\inf}$ i $y^{\sup}$, ponieważ umożliwiają porównanie zbieżności proponowanego algorytmu z innymi podejściami.

Do porównania wybrano algorytm NSGA-II, za pomocą którego uzyskano wysokiej jakości rozwiązania we wcześniejszych eksperymentach. Metoda ta zwraca jednak strategie, a nie programy

do ich wyznaczania. Nie jest efektywnym podejście polegające na uruchomieniu algorytmu NSGA-II niezależnie dla każdego stanu scenariusza, ponieważ nie jest w takim przypadku możliwe uwzględnienie ograniczeń dotyczących całego przebiegu wykonania zadań. W szczególności zadania, których realizacja już się rozpoczęła, nie mogą zostać przeniesione do innego agenta początkowego W, a raz wybrane typy komputerów muszą być wykorzystane w całym scenariuszu.

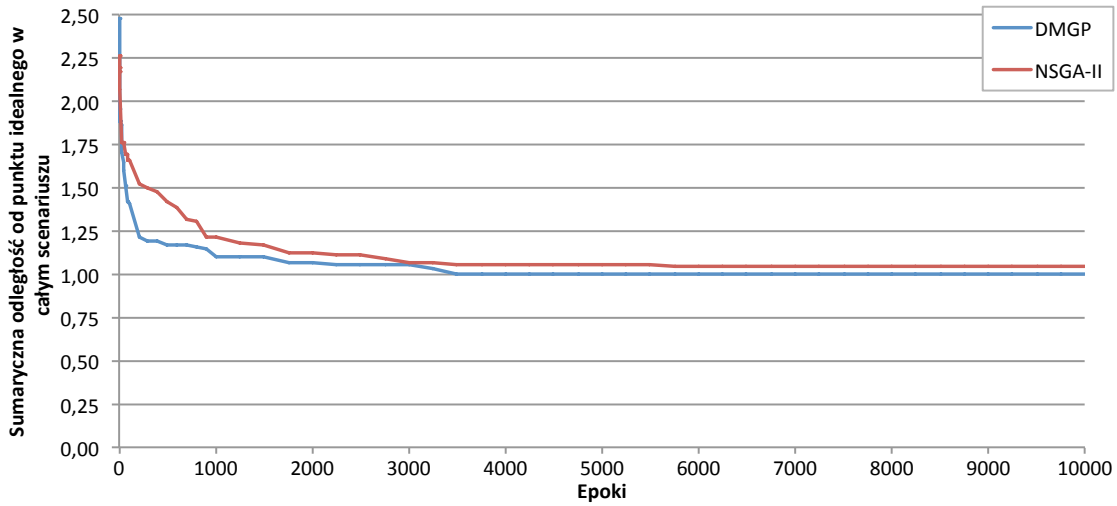
Konfiguracja sprzętowa X^{β}, która zostanie wybrana przez instancję NSGA-II dla pierwszego etapu $\boldsymbol{\mathcal{Z}}_1^{\mathrm{IN}}$ może zostać narzucona w dalszych instancjach metaheurystyki dla pozostałych etapów scenariusza. Takie podejście prowadzi jednak do niskiej jakości wyników w kontekście całego scenariusza, ponieważ wybrane typy komputerów odzwierciedlają wyłącznie warunki pierwszego etapu. Dla kontrastu algorytm DMGP ocenia wybierane konfiguracje sprzętowe we wszystkich etapach scenariusza. W rezultacie istnieje możliwość przypisania typów komputerów X^{β}, które nie jest optymalne w pierwszym etapie, ale skutkuje najlepszymi ocenami dla całego scenariusza.

Aby taką możliwość posiadał również algorytm NSGA-II, konieczne jest jego zmodyfikowanie w stosunku do wersji, którą wykorzystano w poprzednich eksperymentach. Proponuje się genotypową reprezentację osobników w postaci wektora liczb rzeczywistych, który koduje wartości wektora X^{z}, wektora X^{β} oraz Q wektorów X^{α} – po jednym dla każdego etapu scenariusza. Na osobnika składa się zatem $M + I + QV$ zmiennych decyzyjnych. Genotyp osobnika jest interpretowany jako sekwencja pięciu strategii zespołu agentów warstwy pośredniczącej.

W trakcie oceny elementów populacji wyznaczane są wartości kryterium wektorowego dla wszystkich etapów scenariusza. Dla trzech kryteriów optymalizacji i pięciu etapów odpowiada to przeszukiwaniu 15-wymiarowej przestrzeni kryterialnej. Tak skonstruowany algorytm wyznacza przypisanie typów komputerów X^{β} i przypisanie zadań do agentów X^{z} z uwzględnieniem wszystkich etapów scenariusza.

Na rysunku 5.32 porównano zbieżność metod DMGP i NSGA-II. Wykreślono wartości funkcji $\phi_2^{\mathrm{suma}}(\chi)$ dla najlepszych programów od wyznaczania strategii na różnych etapach programowania genetycznego DMGP oraz wartości sum normy ϕ_2 dla sekwencji strategii kodowanych przez najlepsze rozwiązania algorytmu ewolucyjnego NSGA-II. W obu metodach wykorzystano rozmiar populacji $G_{\mathrm{size}} = 50$ i limit epok $G_{\max} = 10000$. Widoczne jest, że algorytm DMGP cechuje się szybszą zbieżnością, przy czym końcowe rozwiązanie ma sumaryczną odległość od punktów idealnych $\phi_2^{\mathrm{suma}}(\chi^*) = 1{,}0008$ w porównaniu do 1,0453 dla najlepszego osobnika uzyskanego przez algorytm NSGA-II. W tabeli nr 5.7 pokazano oceny i wartości miary ϕ_2 dla strategii wyznaczanych przez program χ^* na poszczególnych etapach scenariusza.

W ostatnim eksperymencie rozważa się dzierżawę węzłów obliczeniowych w chmurze jako alternatywę dla zakupu komputerów i utrzymywania fizycznej instancji grida *Comcute*. Kryteria optymalizacji, ograniczenia oraz scenariusz realizacji zadań pozostają takie same, jak w poprzednim eksperymencie. Zamiast wykorzystywanych dotychczas typów komputerów z tabeli nr 3.1 rozpatrywane są profile maszyn wirtualnych dostępne w chmurze obliczeniowej *Amazon Elastic Compute Cloud*, które przedstawiono w tabeli nr 5.3. Pozostałe dane wejściowe nie podlegają modyfikacji w stosunku do wcześniejszego eksperymentu.



Rysunek 5.32: Porównanie zbieżności metod DMGP i NSGA-II
Źródło: opracowanie własne

Tabela 5.7: Oceny strategii wyznaczonych przez program χ^*

q	Strategie wyznaczone przez program χ^*			
	ϕ_2	$\hat{Z}_{\max}[s]$	$\widetilde{Z}_{\max}$ [s]	K [zł]
1	0,2049	2726,27	241,28	120,45
2	0,1940	5260,61	510,48	181,49
3	0,2023	2415,15	284,88	80,75
4	0,1880	2913,28	238,87	136,74
5	0,2107	1358,23	160,48	60,25

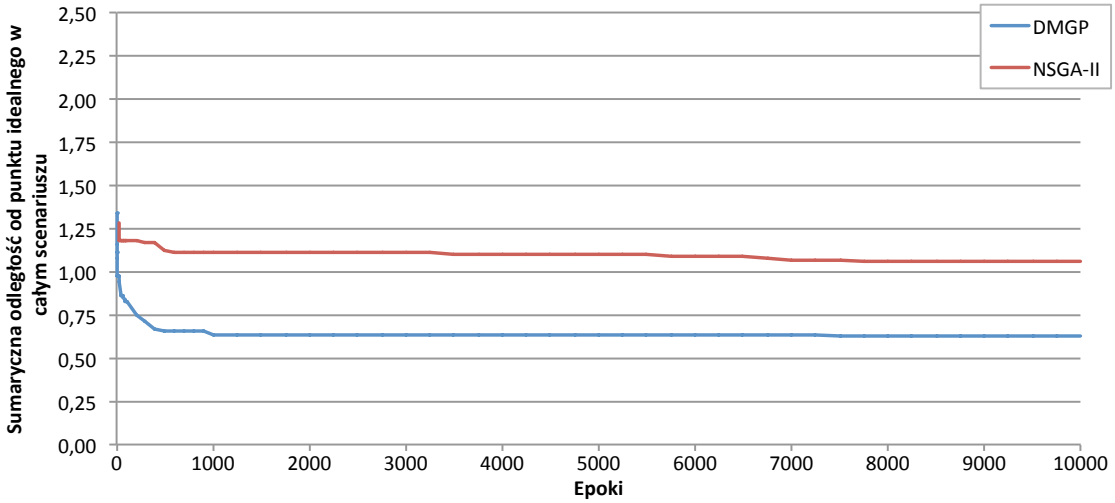
Źródło: opracowanie własne

Dzierżawa zasobów w chmurze umożliwia zmianę konfiguracji węzłów, którą opisuje wektor X^β pomiędzy etapami scenariusza. Wynika to z faktu, że węzły są w tym wypadku maszynami wirtualnymi. Rekonfigurację umożliwia oprogramowanie nadzorcy wirtualizacji (ang. *VM Hypervisor*) dostępne w chmurze *Amazon EC2*. W ten sposób profile maszyn wirtualnych mogą zostać dostrojone dla każdego etapu scenariusza. Możliwości takiej nie stwarza zakup fizycznych komputerów. Zagadnienie optymalizacji programu do wyznaczania strategii zespołu agentów w systemie *Comcute* dla instancji działającej w chmurze obliczeniowej formułuje się następująco:

Dla scenariusza opisanego sekwencją stanów $\boldsymbol{\mathcal{Z}}_1^{\mathrm{IN}}, \ldots, \boldsymbol{\mathcal{Z}}_q^{\mathrm{IN}}, \ldots, \boldsymbol{\mathcal{Z}}_5^{\mathrm{IN}}$ jak w zagadnieniu (5.108), przy czym typy komputerów wybierane są z tabeli nr 5.3 i nie obowiązuje ograniczenie dotyczące takiego samego przypisania typów komputerów X^β w całym scenariuszu oraz przy zadanym kryterium wektorowym $F(x) = [\hat{Z}_{\max}(x), \widetilde{Z}_{\max}(x), K(x)]$, należy wyznaczyć program do specyfikacji strategii χ^*, taki że:

$$\phi_2^{\mathrm{suma}}(\chi^*) = \min_{\chi \in \widehat{\boldsymbol{\mathcal{X}}}_{\boldsymbol{\mathcal{P}}}} \phi_2^{\mathrm{suma}}(\chi). \tag{5.109}$$

Do wyznaczenia rozwiązania zagadnienia (5.109) wykorzystano metodę DMGP. Dla porównania zastosowano zmodyfikowaną metodę NSGA-II, która wyznacza pięć strategii działania agentów – po jednej dla każdego etapu scenariusza – rozwiązując 15-kryterialny problem optymalizacji. W tym przypadku wektor wybranych typów komputerów X^{β} (dla dzierżawy w chmurze – komputerów wirtualnych) może podlegać zmianom pomiędzy etapami. Osobnik w metodzie NSGA-II koduje zatem jeden wektor przydziału zadań X^{z}, Q wektorów X^{β} i Q wektorów X^{α}. W efekcie liczba zmiennych decyzyjnych $M + Q(V + I) = 406$. Porównanie zbieżności obu algorytmów przedstawiono na rysunku 5.33.



Rysunek 5.33: Porównanie zbieżności metod DMGP i NSGA-II w wypadku dzierżawy węzłów w chmurze
Źródło: opracowanie własne

Zauważa się, że zwiększenie liczby zmiennych decyzyjnych utrudnia przeszukiwanie przestrzeni rozwiązań w metodzie NSGA-II. W przypadku metody MOGPA osobnik wykorzystuje taką samą liczbę drzew, jak w poprzednim eksperymencie. Oceny strategii wyznaczanych przez rozwiązania końcowe zaprezentowano w tabeli nr 5.8.

Tabela 5.8: Oceny wyznaczonych strategii

q	Strategie wyznaczone przez program χ^*				NSGA-II			
	ϕ_2	$\hat{Z}_{\max}[s]$	$\widetilde{Z}_{\max}$ [s]	K [zł]	ϕ_2	$\hat{Z}_{\max}[s]$	$\widetilde{Z}_{\max}$ [s]	K [zł]
1	0,1440	6279,71	216,88	1057,66	0,2319	3786,62	301,52	2804,93
2	0,1119	7800,46	379,98	1662,42	0,2502	7800,46	510,48	5608,29
3	0,1282	3463,24	168,08	1089,71	0,1780	3616,76	192,48	1699,80
4	0,1247	5140,78	260,63	1161,45	0,1414	5950,87	291,28	1322,24
5	0,1192	2048,46	117,48	495,99	0,2797	2976,53	129,38	1862,22

Źródło: opracowanie własne

Warto zauważyć, że dla stanów $q = \overline{2,5}$ program χ^* zwraca strategie, które dominują rozwiązania wyznaczone przez algorytm NSGA-II. Dla pierwszego stanu scenariusza wytypowane strategie

nie dominują się wzajemnie, jednak ta wyznaczona przez program χ^* cechuje się mniejszą wartością miary ϕ_2.

Dodatkową zaletą metody DMGP jest to, że program χ^* można wykorzystać do wyznaczenia strategii zespołu agentów również dla stanów grida, które nie zostały uwzględnione w scenariuszu. Własność ta jest szczególnie ważna w dynamicznych systemach, takich jak grid *Comcute*. Wytypowano trzy prawdopodobne stany systemu, dla których wyznaczono strategie zespołu agentów warstwy pośredniczącej przy użyciu programu χ^*. Parametry stanów testowych oraz oceny strategii przedstawiono w tebeli nr 5.9.

Widoczne jest, że dla stanów testowych nr 1 i 2 program χ^* wyznacza wysokiej jakości rozwiązania, o czym świadczą niskie wartości miary ϕ_2. Nieco gorszy rezultat uzyskano dla trzeciego stanu testowego. Dla każdego przypadku program χ^* wyznaczył inną strategię, co świadczy o tym, że następuje dostrajanie rozwiązania do danych wejściowych opisujących stan systemu. Takiej własności nie ma metoda NSGA-II, która wyznacza Q strategii i wymaga ponownego uruchomienia w przypadku zmiany warunków działania gridu.

Tabela 5.9: Testowe stany gridu *Comcute* i oceny strategii wyznaczonych przez program χ^*

Test	Liczba paczek danych $L(z_m)$						Liczba wolontariuszy	Program χ^*			
	z_1	z_2	z_3	z_4	z_5	z_6		ϕ_2	$\hat{Z}_{\max}$ [s]	$\widetilde{Z}_{\max}$ [s]	K [zł]
1	$2{,}5 \times 10^6$	–	–	–	$2{,}5 \times 10^6$	$2{,}5 \times 10^6$	1000	0,0970	3079,23	173,13	764,53
2	5×10^6	–	5×10^6	–	–	$2{,}5 \times 10^6$	1500	0,0694	5280,88	260,63	812,83
3	5×10^6	–	–	–	5×10^6	5×10^6	900	0,2229	6658,26	304,38	1788,75

Źródło: opracowanie własne

Porównanie ocen rozwiązań przedstawionych w tabelach nr 5.8 i 5.7 prowadzi do wniosku, że zakup komputerów pozwala na uzyskanie krótszych czasów obciążenia procesora newralgicznego węzła i niższych kosztów użytkowania systemu w porównaniu do dzierżawy węzłów w chmurze obliczeniowej. Z drugiej strony koszt zakupu komputerów wskazanych przez program χ^* w zagadnieniu (5.108) wynosi 111 319 zł, podczas gdy dzierżawa nie wymaga żadnych inwestycji poprzedzających uruchomienie węzłów w chmurze. Niemniej w przypadku długotrwałego użytkowania systemu lub ze względów bezpieczeństwa rekomendowany jest zakup komputerów. Wyższe koszty użytkowania węzłów w chmurze przekroczą w dłuższej perspektywie czasu początkowy koszt zakupu serwerów.

W instancji laboratoryjnej grida *Comcute* zainstalowano oprogramowanie *OpenStack*, które umożliwiło konfigurację chmury prywatnej obejmującej wszystkie węzły fizyczne systemu. Agenty typu W i S funkcjonują w środowisku maszyn wirtualnych, w których uruchomione są aplikacje do zarządzania obliczeniami oraz do dystrybucji zadań i danych. Migracja agentów pomiędzy węzłami jest możliwa przy użyciu mechanizmu migracji maszyn wirtualnych platformy *OpenStack*. Dzięki wykorzystaniu sieciowego systemu plików, który współdzielą wszystkie komputery, migracja agenta wymaga jedynie replikacji jego pamięci operacyjnej do węzła docelowego.

Zaimplementowane w ramach rozprawy *agenty-solwery* również wdrożono w laboratoryjnej wersji systemu *Comcute*. Posługując się interfejsem aplikacji AAG'16, można dokonać wyboru kryteriów oceny jakości oraz ograniczeń w celu sformułowania problemu optymalizacji, który zostanie

rozwiązany przez *agenty-solwery*. We wrześniu 2016 roku wdrożenie nowej strategii i migracja agentów warstwy pośredniczącej wymagały interwencji administratora gridu, który inicjuje ten proces. Interesującym kierunkiem dalszej modernizacji systemu *Comcute* jest całkowita automatyzacja interakcji pomiędzy *agentami-solwerami* a agentami warstwy pośredniczącej.

5.6 Wnioski i uwagi

Do wyznaczania rozwiązań w zagadnieniu optymalizacji strategii sieci agentów warstwy pośredniczącej gridu wykorzystano programowanie genetyczne. Zastosowano funkcje kary, które pogarszają wartość *fitness* strategii prowadzących do niedopuszczalnych alokacji agentów i zasobów w systemie. W ten sposób preferowane są rozwiązania dopuszczalne przed niedopuszczalnymi. Natomiast zastosowanie rankingów w optymalizacji wielokryterialnej gwarantuje preferowanie rozwiązań niezdominowanych wśród rozwiązań dopuszczalnych.

Ze względu na maksymalizację sprawności konstruowanych strategii w wypadku minimalizowanej funkcji celu zwracana przez nią wartość jest odejmowana od wartości maksymalnej rozpatrywanej funkcji. Natomiast, jeśli funkcja celu jest maksymalizowana, to jej wartość pozostaje bez zmian w wyrażeniu na ocenę sprawności. Warto zauważyć, że wartości rozważanych w modelu funkcji kryterialnych są nieujemne dla strategii dopuszczalnych.

Zaproponowano metodę mapowania genotyp-fenotyp, która pozwala na wyznaczenie strategii sieci agentów przy użyciu programów konstruowanych za pomocą programowania genetycznego. Wprowadzona procedura *interpretacja* gwarantuje wyznaczanie strategii, które spełniają wymagania formalne w rozpatrywanych zagadnieniach optymalizacji. W zaproponowanej wersji programowania genetycznego generowanie drzew może odbywać się według metody konstruowania pełnych drzew, metody przyrostowego konstruowania drzew lub metody hybrydowej.

Przetestowano podstawowe zbiory procedur stosowane w programowaniu genetycznym. Poza wykorzystaniem operatorów arytmetycznych analizowano procedury warunkowe, procedury umożliwiające budowanie wyrażeń logicznych oraz procedury pętli. Wprowadzono limit liczby iteracji dla pętli oraz limit czasu wykonania programu w populacji, aby ograniczyć czas oceny osobników.

Kluczowy problem w wielokryterialnym programowaniu genetycznym polega na transformacji zagadnienia polioptymalizacji w funkcję sprawności. W algorytmach ewolucyjnych rozwiązano go za pomocą procedury nadawania rang w odniesieniu do alternatyw dopuszczalnych. Jeśli wybrane kryteria są w konflikcie, to poprawa jednego kryterium powoduje pogorszenie drugiego. Oznacza to, że dla rozpatrywanego problemu zazwyczaj istnieje zbiór rozwiązań *Pareto*-optymalnych. Rangi umożliwiające wyznaczenie wartości *fitness* obliczane są za pomocą procedury *Goldberga* w oparciu o liczbę poziomów niezdominowania. W alternatywnej procedurze *Fonseci-Fleminga* ranga osobnika odpowiada liczbie rozwiązań, które nad nim dominują. Niezależnie od wybranej procedury, wartości *fitness* wyznaczane na podstawie rang będą takie same dla strategii dopuszczalnych o równych rangach.

Przeprowadzono eksperymenty, w których badano przebieg eksploracji przestrzeni rozwiązań dopuszczalnych w dwukryterialnym zagadnieniu optymalizacji (5.41). Potwierdzono, że w kolejnych

epokach presja selekcyjna kieruje populację ocen strategii w stronę frontu *Pareto*. Zaobserwowana cecha procesu obliczeniowego programowania genetycznego pokazuje, że selekcja oparta o funkcję *fitness*, którą zdefiniowano zależnością (5.40), wytwarza wystarczającą presję, aby właściwie ukierunkować poszukiwanie.

Większa presja selekcyjna przekłada się na szybsze przesuniecie ocen populacji w stronę frontu *Pareto*. Z drugiej strony może doprowadzić do przedwczesnej stagnacji procesu wyszukiwania w wyniku zmniejszenia różnorodności w populacji. Mniejsza presja ułatwia utrzymanie zróżnicowanej populacji, ale zwiększa też prawdopodobieństwo utraty rozwiązań niezdominowanych. Poziom presji wynika z przyjętej procedury nadawania rang oraz wybranego operatora selekcji i jego parametrów, np. rozmiaru turnieju w selekcji turniejowej.

Wprawdzie wielokryterialne programowanie genetyczne opiera się na procedurze nadawania rang, to może nastąpić utrata rozwiązań niezdominowanych z bieżącej populacji w wyniku utworzenia populacji potomnej. Aby uniknąć utraty najlepszych strategii, można zastosować archiwum zewnętrzne, w którym przechowywane są rozwiązania wyznaczające alternatywy lokalnie niezdominowane.

Opracowano wielokryterialne programowanie genetyczne MOGPA, które opiera się na wykorzystaniu archiwum o zadanym rozmiarze. Ponadto w rozprawie proponuje się *dynamiczną geometryczną miarę zagęszczenia* DGCD, która odzwierciedla kompromis między odległością oceny rozpatrywanego rozwiązania od innych ocen a równomiernym rozmieszczeniem osobników w przestrzeni kryterialnej. W metaheurystyce MOGPA dostępne są opcje pozwalające na wybór procedury nadawania rang, sposobu redukcji rozszerzonego archiwum, sposobu aktualizacji populacji programów za pomocą archiwum, a także wartości parametrów archiwum.

W szczególności w algorytmie MOGPA/G/R wykorzystuje się procedurę *Goldberga*, a redukcja nadmiarowych rozwiązań w archiwum odbywa się losowo. Z kolei w algorytmie MOGPA/G/DGCD stosuje się miarę zagęszczenia DGCD. Algorytm MOGPA/FF/R cechuje się uwzględnieniem procedury *Fonseci-Flemminga*. Natomiast w algorytmie MOGPA/FF/DGCD stosuje się dodatkowo miarę DGCD. Aplikacja AAG'16 pozwala na konfigurację poszczególnych aspektów działania metaheurystyki.

Złożoność metaheurystyki MOGPA dla rozważanego w rozprawie problemu wynosi $O(n^6)$, gdzie $n = \max\{I, V, J, G_{\max}, G_{\text{size}}\}$. W oparciu o opisane metody polioptymalizacji strategii agentów warstwy pośredniczącej gridu opracowano *agenty-solwery* wyznaczające reprezentacje *Pareto*-optymalnych rozwiązań. *Agenty-solwery* wykorzystują algorytmy MOGPA/xx/yy, przy czym xx oznacza procedurę nadawania rang, a yy – sposób redukcji przepełnionego archiwum.

Oprócz strategii *Pareto*-optymalnych, istotną rolę odgrywają rozwiązania kompromisowe zagadnienia (5.87). Alternatywy tej klasy mogą być wyznaczane za pomocą zaproponowanej metody CMGP, która w pierwszym etapie wykorzystuje algorytm MOGPA/xx/yy w celu wyznaczenia zbioru rozwiązań niezdominowanych. Z kolei w opracowanej metodzie CGP zastosowano programowanie genetyczne, w którym rolę funkcji celu pełni p-norma.

Do zalet programowania genetycznego należy zaliczyć fakt, że strategia zespołu agentów warstwy pośredniczącej gridu jest wyznaczana przez automatycznie wytworzony program

obejmujący zbiór procedur programistycznych, których wyniki zależą od stanu systemu. Pożądane jest, aby wytworzone programy wyznaczały optymalne strategie nie tylko dla jednego zestawu danych wejściowych, ale dla szeregu stanów odzwierciedlających zdarzenia, jakie mogą zachodzić w gridzie. Dzięki temu reakcja na zmiany w systemie będzie wymagała jedynie ponownego wyznaczenia strategii przy pomocy aktualnego programu. Eliminuje to konieczność oczekiwania na wyznaczenie nowego rozwiązania przez algorytm MOGPA, skracając znacząco czas reakcji na zdarzenia zachodzące w systemie.

Scenariusz realizacji zadań w dynamicznym środowisku jest sekwencją uporządkowanych chronologicznie stanów grida, które są efektem występujących w nim zdarzeń. Każdy ze stanów może być rozpatrywany jako osobne zagadnienie optymalizacji w modelu statycznym. Sformułowano problem optymalizacji programu do wyznaczania strategii zespołu agentów w dynamicznym środowisku (5.103), który to problem uwzględnia cały scenariusz zdarzeń. Do jego rozwiązywania zaproponowano metaheurystykę DMGP, której złożoność wynosi $O(n^7)$, gdzie $n = \max\{I, V, J, G_{\max}, G_{\text{size}}, Q\}$. Szczególne możliwości rekonfiguracji zasobów pomiędzy etapami scenariusza stwarza dzierżawa węzłów w chmurze obliczeniowej. W tym przypadku zmianie może ulegać nie tylko rozmieszczenie agentów, ale również przypisanie typów komputerów do węzłów.

Warto podkreślić, że przeprowadzono szereg eksperymentów numerycznych, które potwierdziły możliwość wyznaczania strategii *Pareto*-optymalnych i kompromisowych za pomocą programowania genetycznego. Stwierdzono, że jakość wyznaczonych rozwiązań jest zazwyczaj wyższa niż wyników uzyskanych za pomocą algorytmu harmonicznego oraz algorytmów ewolucyjnych. Implementacja opracowanych metod w pakiecie AAG'16 umożliwia powtórzenie uzyskanych wyników, a także analizę nowych instancji.

Ponieważ optymalizacja strategii sieci agentów przy rekonfiguracji wybranych zasobów może być zastosowana w odniesieniu do innych gridów, aplikację AAG'16 udostępniono w Sieci na czas naukowej dyskusji nad rozprawą. Dzięki temu administratorzy systemów tej klasy mogą ją zastosować do rekonfiguracji innych gridów. Konieczne jest w takim przypadku dostosowanie wartości parametrów wejściowych modelu optymalizacji do charakterystyk rozpatrywanego systemu.

W celu wykonania pomiarów wybranych danych wejściowych do rozpatrywanych problemów optymalizacji dokonano istotnych modyfikacji podstawowej wersji laboratoryjnego gridu *Comcute*. W szczególności zainstalowano oprogramowanie *OpenStack*, które umożliwiło konfigurację chmury prywatnej. Agenty typu W i S zrealizowano w postaci maszyn wirtualnych, w których uruchomione są odpowiednio aplikacje do zarządzania obliczeniami oraz do dystrybucji zadań i danych. Migracja agentów pomiędzy węzłami jest możliwa przy użyciu mechanizmu migracji maszyn wirtualnych platformy *OpenStack*.

Zaimplementowane *agenty-solwery* wdrożono w laboratoryjnej wersji systemu *Comcute*. Posługując się interfejsem aplikacji AAG'16 można sformułować problem optymalizacji, który zostanie rozwiązany przez *agenty-solwery* w gridzie. Wprawdzie wdrożenie nowej strategii i migracja agentów warstwy pośredniczącej wymaga interwencji administratora gridu, to interesującym kierunkiem dalszej modernizacji systemu *Comcute* jest całkowita automatyzacja interakcji pomiędzy *agentami-solwerami* a agentami warstwy pośredniczącej.

Podsumowanie

Przedstawiony w dysertacji system metodologiczny do wyznaczania strategii zespołu agentów programistycznych w gridach jest oryginalnym dorobkiem autora. Opracowano modele funkcjonowania agentów w systemie rozproszonym, a także omówiono kryteria oceny jakości strategii ich działania oraz nakładane ograniczenia. Na ich podstawie sformułowano zagadnienia optymalizacji wektorowej w odniesieniu do wyznaczania reprezentacji rozwiązań *Pareto*-optymalnych oraz alternatyw kompromisowych. Sformułowano także zagadnienie polioptymalizacji polegające na poszukiwaniu programów komputerowych do wyznaczania strategii sieci agentów w dynamicznym środowisku.

Problemy optymalizacji strategii sieci agentów opierają się na siedemnastu kryteriach wybieranych przez interesariuszy. Optymalizacja może dotyczyć minimalizacji obciążenia procesorów w newralgicznym komputerze, minimalizacji obciążenia komunikacyjnego w newralgicznym hoście, maksymalizacji wydajności gridu, minimalizacji kosztów zakupu komputerów bądź kosztów realizacji zadań, maksymalizacji stopnia rozproszenia agentów czy też maksymalizacji wielkości dostępnej pamięci RAM w newralgicznym komputerze. Równoważenie obciążeń może odnosić się również do innych zasobów, takich jak pamięć dyskowa HDD czy pamięć półprzewodnikowa SSD. Brane pod uwagę są wymagania związane z maksymalizacją poziomu dostępności gridu czy minimalizacją zużycia energii elektrycznej. Można także rozważać minimalizację łącznego obciążenia procesorów w węzłach gridu, łącznego obciążenia komunikacyjnego lub łącznego czasu realizacji zadań. Na wartości każdego z kryteriów optymalizacji mogą być nakładane ograniczenia.

Do wyznaczania rozwiązań dla problemów optymalizacji strategii sieci agentów opracowano cztery grupy algorytmów. Pierwszą grupę stanowią algorytmy do optymalizacji jednokryterialnej, w której interesariusze wybierają funkcję celu oraz zestaw ograniczeń. Druga grupa obejmuje algorytmy do wyznaczania rozwiązań *Pareto*-optymalnych. Trzecia grupa to algorytmy poszukujące strategii kompromisowych. Do czwartej grupy należą algorytmy programowania genetycznego do implementacji programów wyznaczających strategie zespołu agentów w dynamicznym środowisku w zależności od stanu systemu. Algorytmy zaimplementowano w języku *Java* w ramach pakietu AAG'16, który pozwala na powtórzenie wykonanych eksperymentów i uzyskanych dla nich wyników badań. Opracowane metody mogą być zastosowane do wyznaczania strategii, które odzwierciedlają preferencje interesariusza odnośnie kryteriów i ograniczeń w sformułowanych zagadnieniach optymalizacji wielokryterialnej.

W zaproponowanej wersji programowania genetycznego generowanie drzew może odbywać się według wybranej przez interesariusza metody. Procedurę budowania populacji początkowej rozszerzono o możliwość weryfikacji, czy wszystkie dane wejściowe wykorzystano w drzewach programistycznych stosowanych do wyznaczania strategii zespołu agentów. Ponadto odbywa się weryfikacja, czy liczba wierzchołków drzew programów nie przekracza założonego limitu. Omówiono także podstawowe zbiory procedur stosowane w programowaniu genetycznym i porównano zbieżność metody w zależności od wybranego zbioru. Dodatkowo opracowano procedurę agregacji drzew programistycznych i procedurę *interpretacja*, która pozwala na wyznaczanie strategii w zależności od stanu systemu.

Warto podkreślić, że zaproponowane algorytmy programowania genetycznego składają się na oryginalny dorobek naukowy autora zamieszczony w rozprawie. Przy użyciu opracowanych algorytmów wielokryterialnych klasy MOGPA/*xx*/*yy* możliwe jest wyznaczanie rozwiązań optymalnych w sensie *Pareto*. Z kolei za pomocą skonstruowanych algorytmów kompromisowych CMGP i CGP wyznaczane są strategie kompromisowe. Algorytm CMGP wybiera je spośród wyników otrzymanych za pomocą algorytmu MOGPA, natomiast CGP – wyznacza strategie bezpośrednio. Z kolei algorytm DMGP pozwala na konstruowanie programów do wyznaczania strategii uwzględniających dynamiczny charakter gridu.

Wykonano szereg eksperymentów dla testowych instancji sformułowanych zagadnień optymalizacji z wykorzystaniem opracowanych algorytmów. Otrzymane rezultaty potwierdziły prawidłowość rozważań dotyczących modelu systemu, sformułowanych zadań optymalizacji i skonstruowanych algorytmów programowania genetycznego. Opracowane metody wykorzystano do rozwiązania problemów optymalizacji w odniesieniu do laboratoryjnej instancji gridu *Comcute*, a także w odniesieniu do chmury obliczeniowej. Świadczy to o uniwersalności rozważanego modelu i możliwościach jego adaptacji do różnorodnych zagadnień optymalizacji.

Szczególną cechą programowania genetycznego DMGP jest to, że uzyskiwane programy mogą wyznaczać strategie sieci agentów dla wielu stanów gridu. Umożliwia to skrócenie czasu reakcji na zmiany zachodzące w systemie i jego rekonfigurację w trakcie działania. Funkcjonalności tej nie mają inne metody optymalizacji, które zwracają statyczne strategie do wykorzystania przy zadanych warunkach pracy systemu.

Opracowane metody optymalizacji, które opierają się na programowaniu genetycznym oraz wyniki przeprowadzonych eksperymentów stanowią przesłanki do stwierdzenia, że sformułowany na wstępie problem badawczy został poprawnie rozwiązany. W konsekwencji postawiona hipoteza robocza została naukowo zweryfikowana z pozytywnym skutkiem, a zatem można uznać ją za tezę naukową. Powyższe uzasadnia, że cel rozprawy został osiągnięty.

Istotnym wkładem autora w dziedzinach sztucznej inteligencji oraz przetwarzania rozproszonego są opracowane modele, a także zdefiniowane zagadnienia optymalizacji i metody ich rozwiązywania. Należy jednak stwierdzić, że rozprawa nie wyczerpuje rozległej tematyki optymalizacji strategii sieci agentów w systemie rozproszonym, lecz stanowi podstawę do dalszych zaawansowanych badań. Jednym z interesujących problemów jest generalizacja programów do wyznaczania strategii dla nieprzewidzianych stanów grida i ewolucyjne dostrajanie rozwiązań w trakcie działania systemu.

Wyniki eksperymentów wskazują na potencjał drzew programistycznych do uogólniania wyznaczanych strategii na stany systemu, które nie występowały w scenariuszu wykorzystanym na etapie programowania genetycznego.

Sformułowany model systemu rozproszonego typu grid oraz opracowane metody optymalizacji były wykorzystane przez autora w ramach pracy naukowo-badawczej na Politechnice Gdańskiej. Obejmowała ona wdrożenie eksperymentalnego gridu *Comcute* oraz jego późniejszą eksploatację i modernizację. W szczególności podczas przygotowywania rozprawy wdrożono w systemie *agenty-solwery* do optymalizacji strategii sieci agentów warstwy pośredniczącej, które zmodyfikowano pod kątem ich mobilności w gridzie. Wybrane zagadnienia autor opisał także w kilkunastu recenzowanych publikacjach.

Dalsze kierunki badań obejmują rozwój metod programowania genetycznego w celu rozwiązywania innych zagadnień polioptymalizacji. Interesującym obszarem zastosowań jest tzw. przetwarzanie we mgle (ang. *fog computing*) w miejskich infrastrukturach wspierających inteligentne procesy obliczeniowe. Umożliwia ono realizację obliczeń, które wspomagają funkcjonowanie inteligentnego miasta czy regionu.

Podsumowując, należy podkreślić, że rozwój teorii i zastosowań metod wyznaczania strategii zespołów agentów w systemach rozproszonych jest ważnym kierunkiem badawczym w dziedzinach przetwarzania rozproszonego i sztucznej inteligencji. Wyniki badań mają istotne znaczenie praktyczne dla systemów klasy grid, a także dla aplikacji rozproszonych, które działają w obrębie chmury obliczeniowej.

Bibliografia

[1] B. Abrahams, J. Zeleznikow. *Agent and Multi-Agent Systems: Technologies and Applications: 4th KES International Symposium, KES-AMSTA 2010, Gdynia, Poland, June 23-25, 2010, Proceedings. Part I*, Including Notions of Fairness in Development of an Integrated Multi-agent Online Dispute Resolution Environment, str. 102–111. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-13480-7. doi:10.1007/978-3-642-13480-7_12.

[2] R. Agarwal, A. Deo, S. Das. *Intelligent Agents in E-learning.* SIGSOFT Softw. Eng. Notes, 29(2):1–1, III 2004. ISSN 0163-5948. doi:10.1145/979743.979755.

[3] M. T. Al-Hajri, M. A. Abido. *Multiobjective optimal power flow using Improved Strength Pareto Evolutionary Algorithm (SPEA2). Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, str. 1097–1103. Nov 2011. ISSN 2164-7143. doi:10.1109/ISDA.2011.6121805.

[4] N. Al-Madi, S. Ludwig. *Adaptive genetic programming applied to classification in data mining. Nature and Biologically Inspired Computing (NaBIC), 2012 Fourth World Congress on*, str. 79–85. Nov 2012. doi:10.1109/NaBIC.2012.6402243.

[5] M. S. Alam, M. M. Islam, X. Yao, K. Murase. *Diversity Guided Evolutionary Programming: A novel approach for continuous optimization.* Applied Soft Computing, 12(6):1693 – 1707, 2012. ISSN 1568-4946. doi:http://dx.doi.org/10.1016/j.asoc.2012.02.002.

[6] I. Alberto, C. Azcarate, F. Mallor, P. Mateo. *Multiobjective Evolutionary Algorithms. Pareto Rankings.* Monografias del Semin. Matem. Garcia de Galdeano, 27:27–35, 2003.

[7] R. Allan. *Survey of Agent Based Modelling and Simulation Tools.* Rap. tech., Computational Science and Engineering Department, STFC Daresbury Laboratory, 2011. `http://www.grids.ac.uk/Complex/ABMS/ABMS.html`. Dostęp: 2016-01-20.

[8] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster. *The Globus Striped GridFTP Framework and Server. Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC '05, str. 54–64. IEEE Computer Society, Washington, DC, USA, 2005. ISBN 1-59593-061-2. doi:10.1109/SC.2005.72.

[9] B. Allen. *Einstein@Home.* `http://einsteinathome.org/`. Dostęp: 2016-01-20.

[10] R. M. Almeida, E. E. N. Macau. *Percolation model for wildland fire spread dynamics.* E. E. N. Macau, L. F. R. Turci, L. S. Martins Filho, red., *Proceedings of Dynamics Days South America 2010: International Conference on Chaos and Nonlinear Dynamics*. Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2010.

[11] T. Altameem, M. Amoon. *An agent-based approach for dynamic adjustment of scheduled jobs in computational grids.* Journal of Computer and Systems Sciences International, 49(5):765–772, 2010. ISSN 1555-6530. doi:10.1134/S1064230710050114.

[12] Amazon Web Services, Inc. *Amazon Elastic Compute Cloud.* `https://aws.amazon.com/ec2/`. Dostęp: 2016-03-14.

[13] A. Ameljańczyk. *Teoria gier i optymalizacja wektorowa.* WAT, Warszawa, 1980.

[14] A. Ameljańczyk. *Optymalizacja wielokryterialna w problemach sterowania i zarządzania.* PAN, Warszawa, 1984.

[15] D. P. Anderson. *BOINC: a system for public-resource computing and storage. Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, str. 4–10. Nov 2004. ISSN 1550-5510. doi:10.1109/GRID.2004.14.

[16] P. J. Angeline. *Evolutionary Optimization Versus Particle Swarm Optimization: Philosophy and Performance Differences. Proceedings of the 7th International Conference on Evolutionary Programming VII*, EP '98, str. 601–610. Springer-Verlag, London, UK, UK, 1998. ISBN 3-540-64891-7.

[17] M. Anik, S. Ahmed. *A mixed mutation approach for evolutionary programming based on guided selection strategy. Informatics, Electronics Vision (ICIEV), 2013 International Conference on*, str. 1–6. May 2013. doi:10.1109/ICIEV.2013.6572647.

[18] Apache Software Foundation. *Apache JMeter.* `http://jmeter.apache.org/`. Dostęp: 2016-01-20.

[19] A. Arcuri, X. Yao. *A novel co-evolutionary approach to automatic software bug fixing. Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, str. 162–168. June 2008. doi:10.1109/CEC.2008.4630793.

[20] T. Bäck. *Selective pressure in evolutionary algorithms: a characterization of selection mechanisms. Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, str. 57–62 vol.1. Jun 1994. doi: 10.1109/ICEC.1994.350042.

[21] T. Bäck, G. Rudolph, H. P. Schwefel. *Evolutionary Programming and Evolution Strategies: Similarities and Differences.* D. B. Fogel, J. W. Atmar, red., *Proceedings of the 2nd Annual Conference on Evolutionary Programming*, str. 11–22. Evolutionary Programming Society, La Jolla, CA, II 1993.

[22] J. Balicki. *Algorytmy ewolucyjne oraz algorytmy przeszukiwania tabu do optymalizacji przydziałów modułów programów w rozproszonych systemach komputerowych.* Wydawnictwo AMW, Gdynia, 2001.

[23] J. Balicki. *Genetic programming for finding trajectories of underwater vehicle. Robot Motion and Control, 2002. RoMoCo '02. Proceedings of the Third International Workshop on*, str. 217–222. Nov 2002. doi:10.1109/ROMOCO.2002.1177110.

[24] J. Balicki. *Computational Science – ICCS 2006: 6th International Conference, Reading, UK, May 28-31, 2006. Proceedings, Part III*, Negative Selection with Ranking Procedure in Tabu-Based Multi-criterion Evolutionary Algorithm for Task Assignment, str. 863–870. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-34384-4. doi:10.1007/11758532_112.

[25] J. Balicki. *Multi-criterion Tabu Programming for Pareto-optimal Task Assignment in Distributed Computer Systems. Proceedings of the 12th WSEAS International Conference on Computers*, ICCOMP'08, str. 142–147. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 2008. ISBN 978-960-6766-85-5.

[26] J. Balicki. *An Adaptive Quantum-based Multiobjective Evolutionary Algorithm for Efficient Task Assignment in Distributed Systems. Proceedings of the WSEAES 13th International Conference on Computers*, ICCOMP'09, str. 417–422. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 2009. ISBN 978-960-474-099-4.

[27] J. Balicki, Z. Kitowski. *Multicriteria optimization of computer resource allocations with using genetic algorithms and artificial neural networks. Proceedings of the 12th International Conference on Systems Science*, tom 3, str. 11–18. 1995.

[28] J. Balicki, Z. Kitowski. *Evolutionary Multi-Criterion Optimization: First International Conference, EMO 2001 Zurich, Switzerland, March 7–9, 2001 Proceedings*, Multicriteria Evolutionary Algorithm with Tabu Search for Task Assignment, str. 373–384. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. ISBN 978-3-540-44719-1. doi:10.1007/3-540-44719-9_26.

[29] J. Balicki, H. Krawczyk, E. Nawarecki. *Grid and Volunteer Computing.* Wydawnictwo Politechniki Gdańskiej, 2012. ISBN 9788360779170.

[30] J. Balicki, J. Kuchta. *Obliczenia rozproszone w systemach komputerowych o architekturze klasy grid.* Wydawnictwo Politechniki Gdańskiej, Gdańsk, 2012. ISBN 978-83-60779-14-9.

[31] J. Balicki, J. Szymański, M. Kępa, K. Draszawka, W. Korłub. *Artificial Intelligence and Soft Computing: 14th International Conference, ICAISC 2015, Zakopane, Poland, June 14-18, 2015, Proceedings, Part I*, Improving Effectiveness of SVM Classifier for Large Scale Data, str. 675–686. Springer International Publishing, Cham, 2015. ISBN 978-3-319-19324-3. doi:10.1007/978-3-319-19324-3_60.

[32] J. Balicki, *et al. Comcute system utrzymania wielkiej mocy obliczeniowej.* Raport końcowy z projektu rozwojowego OR00010811, Politechnika Gdańska, Gdańsk, 2012.

[33] E. T. Barr, M. Harman, Y. Jia, A. Marginean, J. Petke. *Automated Software Transplantation. Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, str. 257–269. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3620-8. doi:10.1145/2771783.2771796.

[34] T. Bartz-Beielstein, J. Branke, J. Mehnen, O. Mersmann. *Evolutionary Algorithms.* Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 4(3):178–195, 2014. ISSN 1942-4795. doi:10.1002/widm.1124.

[35] F. L. Bellifemine, G. Caire, D. Greenwood. *Developing Multi-Agent Systems with JADE.* Wiley, 2007.

[36] F. L. Bellifemine, G. Caire, T. Trucco, G. Rimassa. *JADE Programmer's Guide.* Telecom Italia Lab, April 2010.

[37] F. L. Bellifemine, G. Caire, T. Trucco, G. Rimassa, R. Mungenast. *JADE Administrator's Guide.* Telecom Italia Lab, April 2010.

[38] H. G. Beyer. *The theory of evolution strategies.* Springer, 2001.

[39] H. G. Beyer, H. P. Schwefel. *Evolution strategies – A comprehensive introduction.* Natural Computing, 1(1):3–52, 2002. ISSN 1567-7818. doi:10.1023/A:1015059928466.

[40] J. Blazewicz, M. Kovalyov, M. Machowiak, D. Trystram, J. Weglarz. *Preemptable malleable task scheduling problem.* Computers, IEEE Transactions on, 55(4):486–490, April 2006. ISSN 0018-9340. doi:10.1109/TC.2006.58.

[41] S. Bleuler, M. Brack, L. Thiele, E. Zitzler. *Multiobjective genetic programming: reducing bloat using SPEA2. Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, tom 1, str. 536–543 vol. 1. 2001. doi:10.1109/CEC.2001.934438.

[42] W. Blewitt, G. Ushaw, G. Morgan. *Applicability of GPGPU Computing to Real-Time AI Solutions in Games.* Computational Intelligence and AI in Games, IEEE Transactions on, 5(3):265–275, Sept 2013. ISSN 1943-068X. doi:10.1109/TCIAIG.2013.2258156.

[43] M. Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, 1987.

[44] R. A. Brooks. *A Robust Layered Control System for a Mobile Robot.* IEEE Journal of Robotics and Automation, RA-2(1):14–23, 1986.

[45] I. Buck. *GPU computing with NVIDIA CUDA. ACM SIGGRAPH 2007 courses*, SIGGRAPH '07. ACM, New York, NY, USA, 2007. doi:10.1145/1281500.1281647.

[46] A. Byrski, R. Debski, M. Kisiel-Dorohinicki. *Agent-based computing in an augmented cloud environment.* International Journal of Computer Systems Science & Engineering, 27(1):7–18, 2012.

[47] J. Cao, D. Kerbyson, E. Papaefstathiou, G. R. Nudd. *Performance modeling of parallel and distributed computing using PACE. Performance, Computing, and Communications Conference, 2000. IPCCC '00. Conference Proceeding of the IEEE International*, str. 485–492. Feb 2000. doi:10.1109/PCCC.2000.830354.

[48] J. Cao, D. J. Kerbyson, G. R. Nudd. *High Performance Service Discovery in Large-Scale MultiAgent and Mobile-Agent Systems.* J. Software Engineering and Knowledge Engineering, Special Issue on Multi-Agent Systems and Mobile Agents, str. 621–641, 2001.

[49] J. Cao, D. Spooner, J. D. Turner, S. Jarvis, D. J. Kerbyson, S. Saini, G. Nudd. *Agent-Based Resource Management for Grid Computing. Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*, str. 350–350. May 2002. doi:10.1109/CCGRID.2002.1017159.

[50] L. Cao, V. Gorodetsky, P. Mitkas. *Agent Mining: The Synergy of Agents and Data Mining.* Intelligent Systems, IEEE, 24(3):64–72, May 2009. ISSN 1541-1672. doi:10.1109/MIS.2009.45.

[51] G. Chen, C. P. Low, Z. Yang. *Preserving and Exploiting Genetic Diversity in Evolutionary Programming Algorithms.* Evolutionary Computation, IEEE Transactions on, 13(3):661–673, June 2009. ISSN 1089-778X. doi:10.1109/TEVC.2008.2011742.

[52] A. Chien, B. Calder, S. Elbert, K. Bhatia. *Entropia: Architecture and Performance of an Enterprise Desktop Grid System.* J. Parallel Distrib. Comput., 63(5):597–610, V 2003. ISSN 0743-7315. doi:10.1016/S0743-7315(03)00006-6.

[53] M. Clerc, J. Kennedy. *The particle swarm - explosion, stability, and convergence in a multidimensional complex space.* IEEE Transactions on Evolutionary Computation, 6(1):58–73, Feb 2002. ISSN 1089-778X. doi:10.1109/4235.985692.

[54] T. Clohessy, T. Acton, L. Morgan. *Smart City as a Service (SCaaS): A Future Roadmap for E-Government Smart City Cloud Computing Initiatives. Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, str. 836–841. Dec 2014. doi:10.1109/UCC.2014.136.

[55] C. A. C. Coello, G. B. Lamont, D. A. V. Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387332545.

[56] S. Conry, K. Kuwabara, V. Lesser, R. A. Meyer. *Multistage negotiation for distributed constraint satisfaction.* Systems, Man and Cybernetics, IEEE Transactions on, 21(6):1462–1477, Nov 1991. ISSN 0018-9472. doi:10.1109/21.135689.

[57] D. Corne, J. Knowles, M. Oates. *The Pareto Envelope-Based Selection Algorithm for Multiobjective Optimization.* M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, H.-P. Schwefel, red., *Parallel Problem Solving from Nature PPSN VI*, tom 1917

z serii *Lecture Notes in Computer Science*, str. 839–848. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-41056-0. doi:10.1007/3-540-45356-3_82.

[58] G. Coulouris, J. Dollimore, T. Kindberg, G. Blair. *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th wyd., 2011. ISBN 0132143011, 9780132143011.

[59] N. L. Cramer. *A Representation for the Adaptive Generation of Simple Sequential Programs. Proceedings of the 1st International Conference on Genetic Algorithms*, str. 183–187. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1985. ISBN 0-8058-0426-9.

[60] H. A. Dau, F. D. Salim, A. Song, L. Hedin, M. Hamilton. *Phone Based Fall Detection by Genetic Programming. Proceedings of the 13th International Conference on Mobile and Ubiquitous Multimedia*, MUM '14, str. 256–257. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-3304-7. doi:10.1145/2677972.2678010.

[61] K. A. De Jong. *Evolutionary Computation: a Unified Approach*. MIT press, 2006.

[62] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan. *A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II*. M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, H.-P. Schwefel, red., *Parallel Problem Solving from Nature PPSN VI*, tom 1917 z serii *Lecture Notes in Computer Science*, str. 849–858. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-41056-0. doi:10.1007/3-540-45356-3_83.

[63] K. Deb, H. Jain. *An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints*. Evolutionary Computation, IEEE Transactions on, 18(4):577–601, Aug 2014. ISSN 1089-778X. doi:10.1109/TEVC.2013.2281535.

[64] K. Deb, D. Kalyanmoy. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 2001. ISBN 047187339X.

[65] K. Deb, J. Sundar. *Reference Point Based Multi-objective Optimization Using Evolutionary Algorithms. Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, str. 635–642. ACM, New York, NY, USA, 2006. ISBN 1-59593-186-4. doi:10.1145/1143997.1144112.

[66] Deutsche Elektronen-Synchrotron DESY. *dCache*. `https://www.dcache.org/`. Dostęp: 2016-03-20.

[67] DigitalOcean Inc. *DigitalOcean Pricing*. `https://www.digitalocean.com/pricing/`. Dostęp: 2016-03-14.

[68] V. T. Dunin-Kęplicz B. *Teamwork in Multi-Agent Systems A formal Approach*. John Wiley & Sons, 2010.

[69] Earth System Grid Federation. *Earth System Grid*. `https://www.earthsystemgrid.org/`. Dostęp: 2016-03-20.

[70] J. Eggermont. *Data mining using genetic programming: classification and symbolic regression.* Rozprawa doktorska, Leiden University, 2005.

[71] EGI.eu. *European Grid Infrastructure.* `http://www.egi.eu/`. Dostęp: 2016-03-20.

[72] A. E. Eiben, J. E. Smith. *Introduction to evolutionary computing.* Springer, 2003.

[73] Enabling Grids for E-sciencE (EGEE) Project. *gLite.* `http://grid-deployment.web.cern.ch/grid-deployment/glite-web/`. Dostęp: 2016-03-20.

[74] M. Farina, P. Amato. *Fuzzy Optimality and Evolutionary Multiobjective Optimization.* C. Fonseca, P. Fleming, E. Zitzler, L. Thiele, K. Deb, red., *Evolutionary Multi-Criterion Optimization*, tom 2632 z serii *Lecture Notes in Computer Science*, str. 58–72. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-01869-8. doi:10.1007/3-540-36970-8_5.

[75] FIPA – Foundations for Intelligent Physical Agents. *FIPA Abstract Architecture Specification.* `http://www.fipa.org/specs/fipa00001/`, December 2002. Dostęp: 2016-01-20.

[76] FIPA – Foundations for Intelligent Physical Agents. *FIPA ACL Message Structure Specification.* `http://www.fipa.org/specs/fipa00061/`, December 2002. Dostęp: 2016-01-20.

[77] FIPA – Foundations for Intelligent Physical Agents. *FIPA Communicative Act Library Specification.* `http://www.fipa.org/specs/fipa00037/`, December 2002. Dostęp: 2016-01-20.

[78] FIPA – Foundations for Intelligent Physical Agents. *FIPA SL Content Language Specification.* `http://www.fipa.org/specs/fipa00008/`, December 2002. Dostęp: 2016-01-20.

[79] FIPA – Foundations for Intelligent Physical Agents. *FIPA Standard Status Specifications.* `http://www.fipa.org/repository/standardspecs.html`, December 2002. Dostęp: 2016-01-20.

[80] FIPA – Foundations for Intelligent Physical Agents. *FIPA Agent Management Specification.* `http://www.fipa.org/specs/fipa00023/`, March 2004. Dostęp: 2016-01-20.

[81] L. J. Fogel, A. J. Owens, M. J. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley, 1966.

[82] C. M. Fonseca, P. J. Fleming. *Genetic Algorithms for Multiobjective Optimization: Formulation Discussion and Generalization. Proceedings of the 5th International Conference on Genetic Algorithms*, str. 416–423. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-299-2.

[83] C. M. Fonseca, P. J. Fleming. *An Overview of Evolutionary Algorithms in Multiobjective Optimization.* Evolutionary Computation, 3:1–16, 1995.

[84] I. Foster, C. Kesselman, S. Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. Int. J. High Perform. Comput. Appl., 15(3):200–222, VIII 2001. ISSN 1094-3420. doi:10.1177/109434200101500302.

[85] Frontline Systems, Inc. *The Frontline solvers*. `http://www.solver.com/`. Dostęp: 2016-01-20.

[86] F. Fu, C. Zhang. *A Modified Harmony Search for Multi-mode Resource Constrained Project Scheduling Problem. Computational Intelligence and Design (ISCID), 2011 Fourth International Symposium on*, tom 1, str. 181–184. Oct 2011. doi:10.1109/ISCID.2011.54.

[87] D. Gâlea, F. Leon, M. H. Zaharia. *E-learning Distributed Framework Using Intelligent Agents*. C. M, G. Dan, V. A, red., *New Trends in Computer Science and Engineering, Anniversary Volume, Department of Computer Engineering, Faculty of Automatic Control and Computer Engineering, Technical University "Gh. Asachi", Polirom Press, Iąsi*, str. 159–163. Citeseer, 2003.

[88] J. Gao, J. Wang, B. Wang, X. Song. *A Papr Reduction Algorithm Based on Harmony Research for Ofdm Systems*. Procedia Engineering, 15(0):2665 – 2669, 2011. ISSN 1877-7058. doi:http://dx.doi.org/10.1016/j.proeng.2011.08.501. {CEIS} 2011.

[89] K. Gao, H. Li, Q. Pan, J. Li, J. Liang. *Hybrid heuristics based on harmony search to minimize total flow time in no-wait flow shop. Control and Decision Conference (CCDC), 2010 Chinese*, str. 1184–1188. May 2010. doi:10.1109/CCDC.2010.5498155.

[90] L. Gao, D. Zou, Y. Ge, W. Jin. *Solving pressure vessel design problems by an effective global harmony search algorithm. Control and Decision Conference (CCDC), 2010 Chinese*, str. 4031–4035. May 2010. doi:10.1109/CCDC.2010.5498438.

[91] X. Gao, X. Wang, S. Ovaska. *Harmony Search Methods for Multi-modal and Constrained Optimization*. Z. Geem, red., *Music-Inspired Harmony Search Algorithm*, tom 191 z serii *Studies in Computational Intelligence*, str. 39–51. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-00184-0. doi:10.1007/978-3-642-00185-7_3.

[92] X.-Z. Gao, T. Jokinen, X. Wang, S. Ovaska, A. Arkkio. *A New Harmony Search method in optimal wind generator design. Electrical Machines (ICEM), 2010 XIX International Conference on*, str. 1–6. Sept 2010. doi:10.1109/ICELMACH.2010.5608219.

[93] T. Gea, J. Paradells, M. Lamarca, D. Roldan. *Smart Cities as an Application of Internet of Things: Experiences and Lessons Learnt in Barcelona. Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on*, str. 552–557. July 2013. doi:10.1109/IMIS.2013.158.

[94] Z. Geem. *Improved Harmony Search from Ensemble of Music Players*. B. Gabrys, R. Howlett, L. Jain, red., *Knowledge-Based Intelligent Information and Engineering Systems*, tom 4251 z serii *Lecture Notes in Computer Science*, str. 86–93. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-46535-5. doi:10.1007/11892960_11.

[95] Z. Geem. *Harmony Search Algorithm for Solving Sudoku*. B. Apolloni, R. Howlett, L. Jain, red., *Knowledge-Based Intelligent Information and Engineering Systems*, tom 4692 z serii *Lecture Notes in Computer Science*, str. 371–378. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-74817-5. doi:10.1007/978-3-540-74819-9_46.

[96] Z. Geem, J.-Y. Choi. *Music Composition Using Harmony Search Algorithm*. M. Giacobini, red., *Applications of Evolutionary Computing*, tom 4448 z serii *Lecture Notes in Computer Science*, str. 593–600. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-71804-8. doi:10.1007/978-3-540-71805-5_65.

[97] Z. W. Geem. *Multiobjective Optimization of Time-Cost Trade-Off Using Harmony Search*. Journal of Construction Engineering and Management, 136(6):711–716, 2010. doi:10.1061/(ASCE)CO.1943-7862.0000167.

[98] Z. W. Geem. *Effects of initial memory and identical harmony in global optimization using harmony search algorithm*. Applied Mathematics and Computation, 218(22):11337 – 11343, 2012. ISSN 0096-3003. doi:http://dx.doi.org/10.1016/j.amc.2012.04.070.

[99] Z. W. Geem. *Recent Advances in Harmony Search Algorithm*. Springer Publishing Company, Incorporated, 2012. ISBN 3642263178, 9783642263170.

[100] Z. W. Geem, J. H. Kim, G. Loganathan. *A new heuristic optimization algorithm: harmony search*. Simulation, 76(2):60–68, 2001.

[101] Z. W. Geem, Y. Park. *Optimal Layout for Branched Networks Using Harmony Search. Proceedings of the 5th WSEAS International Conference on Applied Computer Science*, ACOS'06, str. 364–367. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 2006. ISBN 960-8457-43-2.

[102] A. M. George, A. Razak, N. Wilson. *The Comparison of Multi-objective Preference Inference Based on Lexicographic and Weighted Average Models. Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on*, str. 88–95. Nov 2015. ISSN 1082-3409. doi:10.1109/ICTAI.2015.26.

[103] S. Gil-Lopez, I. Landa-Torres, J. Del Ser, S. Salcedo-Sanz, D. Manjarres, J. Portilla-Figueras. *A Novel Grouping Heuristic Algorithm for the Switch Location Problem Based on a Hybrid Dual Harmony Search Technique*. J. Cabestany, I. Rojas, G. Joya, red., *Advances in Computational Intelligence*, tom 6691 z serii *Lecture Notes in Computer Science*, str. 17–24. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-21500-1. doi:10.1007/978-3-642-21501-8_3.

[104] Globus Alliance. *Globus Toolkit*. `http://toolkit.globus.org/`. Dostęp: 2016-01-20.

[105] Globus Alliance. *Globus Toolkit 6.0 Release Manuals*. `http://toolkit.globus.org/toolkit/docs/6.0/`. Dostęp: 2016-03-20.

[106] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st wyd., 1989. ISBN 0201157675.

[107] J. Gomez-Sanz, R. Fuentes, J. Pavón. *INGENIAS Development Kit*. `http://ingenias.sourceforge.net/`. Dostęp: 2016-02-15.

[108] Google Inc. *Android*. `http://developer.android.com/`. Dostęp: 2016-02-20.

[109] Google Inc. *Google Cloud Platform*. `https://cloud.google.com/`. Dostęp: 2016-03-14.

[110] C. Goues, S. Forrest, W. Weimer. *Current challenges in automatic software repair*. Software Quality Journal, 21(3):421–443, 2013. ISSN 1573-1367. doi:10.1007/s11219-013-9208-0.

[111] J. Granatyr, V. Botelho, O. R. Lessing, E. E. Scalabrin, J.-P. Barthès, F. Enembreck. *Trust and Reputation Models for Multiagent Systems*. ACM Comput. Surv., 48(2):27:1–27:42, X 2015. ISSN 0360-0300. doi:10.1145/2816826.

[112] O. Gutknecht, F. Michel. *MaDKit*. `http://www.madkit.org/`. Dostęp: 2016-02-15.

[113] D. Hadka. *MOEA Framework*. `http://moeaframework.org`. Dostęp: 2016-02-15.

[114] M. Harman, Y. Jia, W. B. Langdon. *Search-Based Software Engineering: 6th International Symposium, SSBSE 2014, Fortaleza, Brazil, August 26-29, 2014. Proceedings*, Babel Pidgin: SBSE Can Grow and Graft Entirely New Functionality into a Real World System, str. 247–252. Springer International Publishing, Cham, 2014. ISBN 978-3-319-09940-8. doi:10.1007/978-3-319-09940-8_20.

[115] M. Harman, W. Langdon, W. Weimer. *Genetic programming for Reverse Engineering. Reverse Engineering (WCRE), 2013 20th Working Conference on*, str. 1–10. Oct 2013. doi:10.1109/WCRE.2013.6671274.

[116] F. Hermenier, N. Loriant, J.-M. Menaud. *Power Management in Grid Computing with Xen*. G. Min, B. Di Martino, L. Yang, M. Guo, G. Rünger, red., *Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops*, tom 4331 z serii *Lecture Notes in Computer Science*, str. 407–416. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-49860-5. doi:10.1007/11942634_43.

[117] C. Hewitt. *Viewing Control Structures as Patterns of Passing Messages*. Artificial Intelligence, 8(3):323–364, VI 1977.

[118] C. Hewitt, J. Inman. *DAI betwixt and between: from 'intelligent agents' to open systems science*. Systems, Man and Cybernetics, IEEE Transactions on, 21(6):1409–1419, Nov 1991. ISSN 0018-9472. doi:10.1109/21.135685.

[119] A. Hirales-Carbajal, A. Tchernykh, R. Yahyapour, J. L. González-García, T. Röblitz, J. M. Ramírez-Alcaraz. *Multiple Workflow Scheduling Strategies with User Run Time Estimates on a Grid*. J. Grid Comput., 10(2):325–346, VI 2012. ISSN 1570-7873. doi:10.1007/s10723-012-9215-6.

[120] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. Second edition, 1992.

[121] J. Horn, N. Nafpliotis, D. Goldberg. *A niched Pareto genetic algorithm for multiobjective optimization. Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, str. 82–87 vol.1. Jun 1994. doi: 10.1109/ICEC.1994.350037.

[122] HP. *Benchmark report for SAS Grid Manager on HP ProLiant BL460c G7 Blades and 3PAR T800 Storage System*. `http://h20195.www2.hp.com/v2/getpdf.aspx/4AA3-8251ENW.pdf?ver=1.0`. Dostęp: 2016-01-20.

[123] Y. Huang, S. Tripathi. *Resource allocation for primary-site fault-tolerant systems*. Software Engineering, IEEE Transactions on, 19(2):108–119, Feb 1993. ISSN 0098-5589. doi:10.1109/32.214829.

[124] Z. Huiyu. *Data mining and classification for traffic systems using genetic network programming*. Rozprawa doktorska, Waseda University, Japan, 2011.

[125] D. Isla. *Handling Complexity in the Halo 2 AI*. Game Developers Conference, 2005.

[126] D. Jara-Roa, P. Valdiviezo-Díaz, M. Agila-Palacios, C. Sarango-Lapo, M. Rodriguez-Artacho. *An adaptive multi-agent based architecture for engineering education. Education Engineering (EDUCON), 2010 IEEE*, str. 217–222. April 2010. doi:10.1109/EDUCON.2010.5492574.

[127] B. Javadi, D. Kondo, J.-M. Vincent, D. P. Anderson. *Discovering Statistical Models of Availability in Large Distributed Systems: An Empirical Study of SETI@home*. IEEE Transactions on Parallel and Distributed Systems, 22(11):1896–1903, 2011. ISSN 1045-9219. doi:http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.50.

[128] D. Kafura, J. Briot. *Actors And Agents*. IEEE Concurrency, 6(2):24–29, April 1998. ISSN 1092-3063. doi:10.1109/MCC.1998.678786.

[129] H. Kameda, J. Li, C. Kim, Y. Zhang. *Optimal Load Balancing in Distributed Computer Systems*. Springer-Verlag, London, UK, UK, 1997. ISBN 3-540-76130-6.

[130] J. Kaplan, N. Yankelovich. *Open Wonderland: An Extensible Virtual World Architecture*. IEEE Internet Computing, 15(5):38–45, Sept 2011. ISSN 1089-7801. doi:10.1109/MIC.2011.76.

[131] Katedra Architektury Systemów Komputerowych, Wydział ETI, Politechnika Gdańska. *Comcute*. `http://comcute.eti.pg.gda.pl/`. Dostęp: 2016-01-20.

[132] K. K. Khajwaniya, V. Tiwari. *Satellite image denoising using Weiner filter with SPEA2 algorithm. Intelligent Systems and Control (ISCO), 2015 IEEE 9th International Conference on*, str. 1–6. Jan 2015. doi:10.1109/ISCO.2015.7282324.

[133] Khronos Group. *OpenCL*. `http://www.khronos.org/opencl/`. Dostęp: 2016-02-20.

[134] M. Kim, T. Hiroyasu, M. Miki, S. Watanabe. *SPEA2+: Improving the Performance of the Strength Pareto Evolutionary Algorithm 2.* X. Yao, E. Burke, J. Lozano, J. Smith, J. Merelo-Guervós, J. Bullinaria, J. Rowe, P. Tiňo, A. Kabán, H.-P. Schwefel, red., *Parallel Problem Solving from Nature - PPSN VIII*, tom 3242 z serii *Lecture Notes in Computer Science*, str. 742–751. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-23092-2. doi:10.1007/978-3-540-30217-9_75.

[135] S.-H. Kim, W.-S. Yoo, K.-J. Oh, I.-S. Hwang, J.-E. Oh. *Transient Analysis and Leakage Detection Algorithm using GA and HS algorithm for a Pipeline System.* Journal of Mechanical Science and Technology, 20(3):426–434, 2006. ISSN 1738-494X. doi:10.1007/BF02917526.

[136] L. Kleinrock, W. Korfhage. *Collecting Unused Processing Capacity: An Analysis of Transient Distributed Systems.* IEEE Transactions on Parallel and Distributed Systems, str. 535–546, May 1993.

[137] J. Knowles, D. Corne. *The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation. Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, tom 1, str. –105 Vol. 1. 1999. doi:10.1109/CEC.1999.781913.

[138] Z. Kobti, S. Sharma. *A Multi-Agent Architecture for Game Playing. Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, str. 276–281. April 2007. doi:10.1109/CIG.2007.368109.

[139] Konsorcjum PL-Grid. *PL-Grid.* `http://www.plgrid.pl/`. Dostęp: 2016-01-20.

[140] J. Korczak, M. Hernes, M. Bac. *Fuzzy logic in the multi-agent financial decision support system. Computer Science and Information Systems (FedCSIS), 2015 Federated Conference on*, str. 1367–1376. Sept 2015. doi:10.15439/2015F155.

[141] W. Korłub, H. Balicka, J. Balicki, P. Dryja, P. Przybyłek, M. Tyszka, M. Zadroga, M. Zakidalski. *Metoda ewolucyjno-neuronowa oraz metoda wektorów nośnych w bankowości.* K. Kreft, red., *Wyzwania społeczeństwa informacyjnego*, str. 73–87. Wydawnictwo Uniwersytetu Gdańskiego, Gdańsk, 2015.

[142] W. Korłub, J. Balicki. *Uczenie maszynowe do samoorganizacji systemów rozproszonych w zastosowaniach gospodarczych.* Współczesna Gospodarka, 2016 (w recenzji).

[143] W. Korłub, J. Balicki, M. Beringer, P. Dryja, J. Paluszak, P. Przybyłek, M. Tyszka, M. Zadroga. *Inteligentne systemy agentowe w systemach zdalnego nauczania.* EduAkcja. Magazyn edukacji elektronicznej, 1(9):51–64, 2015.

[144] W. Korłub, J. Balicki, M. Beringer, P. Dryja, M. Tyszka, M. Zadroga. *Inteligentne superkomputery wirtualne do prognozowania trendów w finansa.* K. Kreft, red., *Wyzwania społeczeństwa informacyjnego*, str. 59–72. Wydawnictwo Uniwersytetu Gdańskiego, Gdańsk, 2015.

[145] W. Korłub, J. Balicki, M. Beringer, P. Przybylek, M. Tyszka, M. Zadroga. *Collective citizens' behavior modelling with support of the Internet of Things and Big Data. Human System Interactions (HSI), 2015 8th International Conference on*, str. 61–67. June 2015. doi:10.1109/HSI.2015.7170644.

[146] W. Korłub, J. Balicki, T. Bieliński, J. Paluszak. *Volunteer Computing System Comcute with Smart Scheduler.* J. Balicki, red., *Applications of Information Systems in Engineering and Bioscience*, Proceeding of SEPADS'14 – 13th International Conference on Software Engineering, Parallel and Distributed Systems, str. 54–60. WSEAS Press, 2014.

[147] W. Korłub, J. Balicki, T. Boiński, J. Paluszak, A. Polak. *Optymalizacja równoważenia obciążeń w systemach klasy grid.* J. Balicki, J. Kuchta, red., *Obliczenia rozproszone w systemach komputerowych o architekturze klasy grid*, str. 145–158. Wydawnictwo Politechniki Gdańskiej, Gdańsk, 2012.

[148] W. Korłub, J. Balicki, H. Krawczyk. *Genetic Positioning of Fire Stations Utilizing Grid-computing Platform.* J. Balicki, H. Krawczyk, E. Nawarecki, red., *Grid and Volunteer Computing*, str. 116–131. GUT, Gdańsk, 2012.

[149] W. Korłub, J. Balicki, H. Krawczyk, J. Paluszak. *Genetic programming with negative selection for volunteer computing system optimization. Human System Interaction (HSI), 2013 The 6th International Conference on*, str. 271–278. June 2013. ISSN 2158-2246. doi:10.1109/HSI.2013.6577835.

[150] W. Korłub, J. Balicki, J. Masiejczyk, J. Paluszak. *Mersenne Number Finding and Collatz Hypothesis Verification in the Comcute Grid System.* J. Balicki, H. Krawczyk, E. Nawarecki, red., *Grid and Volunteer Computing*, str. 148–162. GUT, Gdańsk, 2012.

[151] W. Korłub, J. Balicki, J. Masiejczyk, J. Paluszak. *Techniki audytowania zabezpieczeń sieci bezprzewodowej z wykorzystaniem systemów klasy grid.* J. Balicki, J. Kuchta, red., *Obliczenia rozproszone w systemach komputerowych o architekturze klasy grid*, str. 161–174. Wydawnictwo Politechniki Gdańskiej, Gdańsk, 2012.

[152] W. Korłub, J. Balicki, J. Paluszak. *Efektywne zrównoleglanie obliczeń w systemie klasy grid na przykładzie hipotezy Collatza.* J. Balicki, J. Kuchta, red., *Obliczenia rozproszone w systemach komputerowych o architekturze klasy grid*, str. 243–254. Wydawnictwo Politechniki Gdańskiej, Gdańsk, 2012.

[153] W. Korłub, J. Balicki, J. Paluszak. *Pattern Recognition and Machine Intelligence: 6th International Conference, PReMI 2015, Warsaw, Poland, June 30 - July 3, 2015, Proceedings*, Big Data Processing by Volunteer Computing Supported by Intelligent Agents, str. 268–278. Springer International Publishing, Cham, 2015. ISBN 978-3-319-19941-2. doi:10.1007/978-3-319-19941-2_26.

[154] W. Korłub, J. Balicki, J. Paluszak, A. Zacniewski. *Genetic Programming for Workload Balancing in the Comcute Grid System.* J. Balicki, H. Krawczyk, E. Nawarecki, red., *Grid and Volunteer Computing*, str. 97–115. GUT, Gdańsk, 2012.

[155] W. Korłub, J. Balicki, J. Szymanski, M. Zakidalski. *Artificial Intelligence and Soft Computing: 13th International Conference, ICAISC 2014, Zakopane, Poland, June 1-5, 2014, Proceedings, Part I*, Big Data Paradigm Developed in Volunteer Grid System with Genetic Programming Scheduler, str. 771–782. Springer International Publishing, Cham, 2014. ISBN 978-3-319-07173-2. doi:10.1007/978-3-319-07173-2_66.

[156] W. Korłub, J. Balicki, M. Tyszka. *Harmony Search to Self-Configuration of Fault-Tolerant Grids for Big Data*. Z. Kowalczuk, red., *Advanced and Intelligent Computations in Diagnosis and Control*, tom 386 z serii *Advances in Intelligent Systems and Computing*, str. 411–424. Springer International Publishing, 2016. ISBN 978-3-319-23179-2. doi:10.1007/978-3-319-23180-8_30.

[157] W. Korłub, J. Balicki, M. Zakidalski. *Some Optimization Methods for Simulations in Volunteer and Grid Systems*. P. Czarnul, red., *Modeling Large-Scale Computing Systems: Concepts and Models*, str. 161–172. Wydawnictwo Politechniki Gdańskiej, Gdańsk, 2013.

[158] W. Korłub, M. Kania, J. Krajewski. *A proposition for integrating elements of game universe by means of behavioral trees*. Zeszyty Naukowe Wydziału ETI Politechniki Gdańskiej, 2011.

[159] W. Korłub, M. Kania, J. Krajewski. *Integrating heterogeneous systems with high dependability requirements by means of web services. ICT Young 2012: II konferencja Studentów i Doktorantów Elektroniki, Telekomunikacji, Informatyki, Automatyki i Robotyki: materiały konferencyjne, 26-27.05.2012*, str. 249–253. Wydawnictwo Politechniki Gdańskiej, Gdańsk, 2012.

[160] W. Korłub, M. Wójcik. *Possible uses of crisis situation aiding system in virtual world simulation*. T. Boiński, J. Lebiedź, M. Wójcik, red., *Wytwarzanie Gier Komputerowych: Materiały konferencyjne Krajowej Konferencji Wytwarzania Gier Komputerowych, 31.08-2.09.2016*, 2, str. 133–142. Wydawnictwo Politechniki Gdańskiej, Gdańsk, 2012.

[161] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-11170-5.

[162] J. R. Koza. *Genetic programming IV: routine human-competitive intelligence*. Springer, New York, 2003. ISBN 1-4020-7446-8.

[163] J. R. Koza, D. Andre, F. H. Bennett III, M. Keane. *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman, IV 1999. ISBN 1-55860-543-6.

[164] S. Kraus. *Automated Negotiation and Decision Making in Multiagent Environments*. M. Luck, V. Mařík, O. Štěpánková, R. Trappl, red., *Multi-Agent Systems and Applications*, tom 2086 z serii *Lecture Notes in Computer Science*, str. 150–172. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-42312-6. doi:10.1007/3-540-47745-4_7.

[165] K. Kravari. *EMERALD*. `http://lpis.csd.auth.gr/systems/emerald/index.html`. Dostęp: 2016-01-20.

[166] K. Kravari, N. Bassiliades. *A Survey of Agent Platforms*. Journal of Artificial Societies and Social Simulation, 18(1):11, 2015. ISSN 1460-7425. doi:10.18564/jasss.2661.

[167] A. N. Laboratory. *Repast*. `http://repast.sourceforge.net/`. Dostęp: 2016-02-15.

[168] M. Lammie, P. Brenner, D. Thain. *Scheduling Grid Workloads on Multicore Clusters to Minimize Energy and Maximize Performance. 10th IEEE/ACM International Conference on Grid Computing*. 2009.

[169] S. Lander, V. Lesser. *Customizing Distributed Search Among Agents with Heterogeneous Knowledge*. Proceedings of the First International Conference on Information and Knowledge Management, str. 335–344, January 1992.

[170] W. B. Langdon. *Performance of genetic programming optimised Bowtie2 on genome comparison and analytic testing (GCAT) benchmarks*. BioData Mining, 8(1):1–7, 2015. ISSN 1756-0381. doi:10.1186/s13040-014-0034-0.

[171] W. B. Langdon, A. P. Harrison. *GP on SPMD parallel graphics hardware for mega Bioinformatics data mining*. Soft Computing, 12(12):1169–1183, 2008. ISSN 1433-7479. doi:10.1007/s00500-008-0296-x.

[172] W. B. Langdon, M. Modat, J. Petke, M. Harman. *Improving 3D Medical Image Registration CUDA Software with Genetic Programming. Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, str. 951–958. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2662-9. doi:10.1145/2576768.2598244.

[173] C. Le Goues, M. Dewey-Vogt, S. Forrest, W. Weimer. *A Systematic Study of Automated Program Repair: Fixing 55 out of 105 Bugs for $8 Each. Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, str. 3–13. IEEE Press, Piscataway, NJ, USA, 2012. ISBN 978-1-4673-1067-3.

[174] T. Leppänen, J. Riekki, M. Liu, E. Harjula, T. Ojala. *Internet of Things Based on Smart Objects: Technology, Middleware and Applications*, Mobile Agents-Based Smart Objects for the Internet of Things, str. 29–48. Springer International Publishing, Cham, 2014. ISBN 978-3-319-00491-4. doi:10.1007/978-3-319-00491-4_2.

[175] B. Li, E. Zhou, B. Huang, J. Duan, Y. Wang, N. Xu, J. Zhang, H. Yang. *Large scale recurrent neural network on GPU. Neural Networks (IJCNN), 2014 International Joint Conference on*, str. 4062–4069. July 2014. doi:10.1109/IJCNN.2014.6889433.

[176] K. Li. *Optimal Load Distribution in Nondedicated Heterogeneous Cluster and Grid Computing Environments*. J. Syst. Archit., 54(1-2):111–123, I 2008. ISSN 1383-7621. doi:10.1016/j.sysarc.2007.04.003.

[177] P. Lichodzijewski, M. I. Heywood, A. Nur Zincir-Heywood. *Cascaded GP models for data mining. Evolutionary Computation, 2004. CEC2004. Congress on*, tom 2, str. 2258–2264 Vol.2. June 2004. doi:10.1109/CEC.2004.1331178.

[178] B. J. Liu T. *An Intuitive and Flexible Architecture for Intelligent Mobile Robots.* 2nd International Conference on Autonomous Robots and Agents, 2004.

[179] M. Luck, P. McBurney, O. Shehory, S. Willmott. *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing).* AgentLink, 2005.

[180] S. A. Ludwig, S. Roos. *Knowledge-Based and Intelligent Information and Engineering Systems: 14th International Conference, KES 2010, Cardiff, UK, September 8-10, 2010, Proceedings, Part IV*, Prognosis of Breast Cancer Using Genetic Programming, str. 536–545. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-15384-6. doi: 10.1007/978-3-642-15384-6_57.

[181] B. Luo, J. Zheng, J. Xie, J. Wu. *Dynamic Crowding Distance: A New Diversity Maintenance Strategy for MOEAs. Natural Computation, 2008. ICNC '08. Fourth International Conference on*, tom 1, str. 580–585. Oct 2008. doi:10.1109/ICNC.2008.532.

[182] R. Maddegoda, A. S. Karunananda. *Multi agent based approach to assist the design process of 3D game environments. Advances in ICT for Emerging Regions (ICTer), 2012 International Conference on*, str. 36–44. Dec 2012. doi:10.1109/ICTer.2012.6422829.

[183] H. H. Maheta, V. K. Dabhi. *An improved SPEA2 Multi objective algorithm with non dominated elitism and Generational Crossover. Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014 International Conference on*, str. 75–82. Feb 2014. doi:10.1109/ICICICT.2014.6781256.

[184] D. Manjarres, I. Landa-Torres, S. Gil-Lopez, J. Del Ser, M. N. Bilbao, S. Salcedo-Sanz, Z. W. Geem. *A Survey on Applications of the Harmony Search Algorithm.* Eng. Appl. Artif. Intell., 26(8):1818–1831, IX 2013. ISSN 0952-1976. doi:10.1016/j.engappai.2013.05.008.

[185] Z. Manna, R. J. Waldinger. *Toward Automatic Program Synthesis.* Commun. ACM, 14(3):151–165, III 1971. ISSN 0001-0782. doi:10.1145/362566.362568.

[186] N. Marz, J. Warren. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems.* Manning Publications Company, 2015. ISBN 9781617290343.

[187] J. Masiejczyk. *Wielokryterialne algorytmy przeszukiwania tabu do optymalizacji trajektorii autonomicznego pojazdu podwodnego.* Rozprawa doktorska, Wydział Elektroniki, Telekomunikacji i Informatyki, Politechnika Gdańska, 2013.

[188] A. Maslow. *Motivation and personality.* Harper & Row, New York, 2. wyd., 1970.

[189] MAXON Computer. *CINEBENCH.* `http://www.maxon.net/products/cinebench/overview.html`. Dostęp: 2016-01-20.

[190] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan, M. O'Neill. *Grammar-based Genetic Programming: a survey.* Genetic Programming and Evolvable Machines, 11(3):365–396, 2010. ISSN 1573-7632. doi:10.1007/s10710-010-9109-y.

[191] Mersenne Research, Inc. *GIMPS: the Great Internet Mersenne Prime Search*. `http://www.mersenne.org/`. Dostęp: 2016-03-20.

[192] Microsoft Corporation. *Microsoft Azure*. `http://azure.microsoft.com/`. Dostęp: 2016-03-14.

[193] J. F. Miller. *An empirical study of the efficiency of learning boolean functions using a Cartesian Genetic Programming approach*. W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, R. E. Smith, red., *Proceedings of the Genetic and Evolutionary Computation Conference*, tom 2, str. 1135–1142. Morgan Kaufmann, Orlando, Florida, USA, 13-17 VII 1999. ISBN 1-55860-611-4.

[194] J. F. Miller. *Cartesian Genetic Programming*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-17310-3. doi:10.1007/978-3-642-17310-3_2.

[195] T. Moehlman, V. Lesser, B. Buteau. *Decentralized negotiation: An approach to the distributed planning problem*. Group Decision and Negotiation, 1(2):161–191, 1992. ISSN 0926-2644. doi:10.1007/BF00406753.

[196] D. J. Montana. *Strongly Typed Genetic Programming*. Evol. Comput., 3(2):199–230, VI 1995. ISSN 1063-6560. doi:10.1162/evco.1995.3.2.199.

[197] J. Montusiewicz. *Ewolucyjna analiza wielokryterialna w zagadnieniach technicznych*. Wyd. Instytut Podstawowych Problemów Techniki PAN, Warszawa, 2004.

[198] N. Nedjah, A. Abraham, L. de Macedo Mourelle. *Genetic Systems Programming: Theory and Experiences*. Springer Publishing Company, Incorporated, 1st wyd., 2009. ISBN 3540298495, 9783540298496.

[199] C. Nikolai, G. Madey. *Tools of the Trade: A Survey of Various Agent Based Modeling Platforms*. Journal of Artificial Societies and Social Simulation, 12(2):2, 2009. ISSN 1460-7425.

[200] NorduGrid. *ARC middleware*. `http://www.nordugrid.org/`. Dostęp: 2016-03-20.

[201] H. S. Nwana. *Software Agents: An Overview*. Knowledge Engineering Review, 11(3):1–40, IX 1996.

[202] J. J. Odell. *Objects and Agents Compared*. Journal of Object Technology, 1(1):41–53, May-June 2002.

[203] OMG – Object Management Group. *MASIF – Mobile Agent System Interoperability Facilities Specification*, November 1997. `http://www.omg.org/cgi-bin/doc?orbos/97-10-05`.

[204] M. O'Neil, C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*, Grammatical Evolution, str. 33–47. Springer US, Boston, MA, 2003. ISBN 978-1-4615-0447-4. doi:10.1007/978-1-4615-0447-4_4.

[205] M. O'Neill, L. Vanneschi, S. Gustafson, W. Banzhaf. *Open Issues in Genetic Programming.* Genetic Programming and Evolvable Machines, 11(3-4):339–363, IX 2010. ISSN 1389-2576. doi:10.1007/s10710-010-9113-2.

[206] Open Wonderland Foundation. *Open Wonderland.* `http://openwonderland.org/`. Dostęp: 2016-03-20.

[207] Oracle. *Java Online Documentation.* `http://www.oracle.com/technetwork/java/index.html`. Dostęp: 2016-01-20.

[208] Oracle Corporation. *Glassfish.* `https://glassfish.java.net/documentation.html`. Dostęp: 2016-01-20.

[209] J. Paluszak. *Optymalizacja wykorzystania zasobów w systemach rozproszonych o architekturze typu grid.* Rozprawa doktorska, Wydział Elektroniki, Telekomunikacji i Informatyki, Politechnika Gdańska, 2015.

[210] V. Pande. *Folding@Home.* `https://folding.stanford.edu/`. Dostęp: 2016-01-20.

[211] V. Pandey, W.-K. Ng, E.-P. Lim. *Financial advisor agent in a multi-agent financial trading system. Database and Expert Systems Applications, 2000. Proceedings. 11th International Workshop on*, str. 482–486. 2000. ISSN 1529-4188. doi:10.1109/DEXA.2000.875070.

[212] PassMark Software. *CPU Benchmarks.* `http://www.cpubenchmark.net/`. Dostęp: 2016-03-20.

[213] T. K. Paul, H. Iba. *Prediction of Cancer Class with Majority Voting Genetic Programming Classifier Using Gene Expression Data.* IEEE/ACM Transactions on Computational Biology and Bioinformatics, 6(2):353–367, April 2009. ISSN 1545-5963. doi:10.1109/TCBB.2007.70245.

[214] J. Petke, M. Harman, W. B. Langdon, W. Weimer. *Using Genetic Improvement and Code Transplants to Specialise a C++ Program to a Problem Class.* M. Nicolau, K. Krawiec, M. I. Heywood, M. Castelli, P. Garcia-Sanchez, J. J. Merelo, V. M. Rivas Santos, K. Sim, red., *17th European Conference on Genetic Programming*, tom 8599 z serii *LNCS*, str. 137–149. Springer, Granada, Spain, 23-25 IV 2014. doi:doi:10.1007/978-3-662-44303-3_12.

[215] M. Philips. *Knight Shows How to Lose $440 Million in 30 Minutes.* `http://www.bloomberg.com/news/articles/2012-08-02/knight-shows-how-to-lose-440-million-in-30-minutes`, 2012. Dostęp: 2016-01-20.

[216] A. Pimenta, F. G. Guimarães, E. G. Carrano, C. A. L. Nametala, R. H. C. Takahashi. *GoldMiner: A genetic programming based algorithm applied to Brazilian Stock Market. Computational Intelligence and Data Mining (CIDM), 2014 IEEE Symposium on*, str. 397–402. Dec 2014. doi:10.1109/CIDM.2014.7008695.

[217] P. Plaszczak, R. Wellner. *Grid Computing: The Savvy Manager's Guide.* Morgan Kaufmann, first wyd., 2005. ISBN 0127425039.

[218] A. Pokahr, L. Braubach, W. Lamersdorf. *A flexible BDI architecture supporting extensibility. Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, str. 379–385. Sept 2005. doi:10.1109/IAT.2005.9.

[219] R. Poli, W. B. Langdon, N. F. McPhee. *A field guide to genetic programming.* Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[220] Postdot Technologies, Inc. *Postman.* `https://www.getpostman.com/`, 2016. Dostęp: 2016-01-20.

[221] J. Powers, M. Sen. *Mathematical Methods in Engineering.* Cambridge University Press, 2015. ISBN 9781107037045.

[222] A. Pugliese, D. Talia, R. Yahyapour. *Modeling and Supporting Grid Scheduling.* Journal of Grid Computing, 6(2):195–213, 2008. ISSN 1570-7873. doi:10.1007/s10723-007-9083-7.

[223] A. S. Rao, M. P. George. *BDI agents: From theory to practice. Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, str. 312–319. 1995.

[224] H. Rasiowa. *Wstęp do matematyki współczesnej.* PWN, Warszawa, xiv wyd., 2007.

[225] M. L. Raymer, W. F. Punch, E. D. Goodman, L. A. Kuhn. *Genetic Programming for Improved Data Mining: Application to the Biochemistry of Protein Interactions. Proceedings of the 1st Annual Conference on Genetic Programming*, str. 375–380. MIT Press, Cambridge, MA, USA, 1996. ISBN 0-262-61127-9.

[226] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Problemata, 15. Frommann-Holzboog, 1973. ISBN 9783772803734.

[227] S. Refat, M. El-Telbany, H. Hefny, A. Bahnasawi. *Discovering the classification rules for Egyptian stock market using genetic programming. Circuits and Systems, 2003 IEEE 46th Midwest Symposium on*, tom 2, str. 952–955 Vol. 2. Dec 2003. ISSN 1548-3746. doi:10.1109/MWSCAS.2003.1562444.

[228] P. Resnick, K. Kuwabara, R. Zeckhauser, E. Friedman. *Reputation Systems.* Commun. ACM, 43(12):45–48, XII 2000. ISSN 0001-0782. doi:10.1145/355112.355122.

[229] T. Rings, H. Neukirchen, J. Grabowski. *Testing Grid Application Workflows Using TTCN-3. Software Testing, Verification, and Validation, 2008 1st International Conference on*, str. 210–219. April 2008. doi:10.1109/ICST.2008.24.

[230] J. S. Rosenschein, G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers.* MIT Press, Cambridge, MA, USA, 1994. ISBN 0-262-18159-2.

[231] S. J. Russell, P. Norvig. *Artificial Intelligence a modern approach.* Prentice Hall, Upper Saddle River, N.J., 2nd international edition wyd., 2003.

[232] C. Ryan. *Automatic Re-engineering of Software Using Genetic Programming*. Springer US, Boston, MA, 2000. ISBN 978-1-4615-4631-3.

[233] C. Ryan, K. Krawiec, U.-M. O'Reilly, J. Fitzgerald, D. Medernach. *Genetic Programming: 17th European Conference, EuroGP 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers*, Building a Stage 1 Computer Aided Detector for Breast Cancer Using Genetic Programming, str. 162–173. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-662-44303-3. doi:10.1007/978-3-662-44303-3_14.

[234] B. Sareni, L. Krahenbuhl. *Fitness sharing and niching methods revisited*. IEEE Transactions on Evolutionary Computation, 2(3):97–106, Sep 1998. ISSN 1089-778X. doi:10.1109/4235.735432.

[235] J. D. Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. Proceedings of the 1st International Conference on Genetic Algorithms*, str. 93–100. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1985. ISBN 0-8058-0426-9.

[236] H. Schwefel. *Evolutionsstrategie und numerische Optimierung*. Technische Universität Berlin, 1975.

[237] Science and Technology Facilities Council. *FLAME – Flexible Large-scale Agent Modelling Environment*. `http://www.flame.ac.uk/`. Dostęp: 2016-01-20.

[238] H. Seada, K. Deb. *Evolutionary Multi-Criterion Optimization: 8th International Conference, EMO 2015, Guimarães, Portugal, March 29 –April 1, 2015. Proceedings, Part II*, U-NSGA-III: A Unified Evolutionary Optimization Procedure for Single, Multiple, and Many Objectives: Proof-of-Principle Results, str. 34–49. Springer International Publishing, Cham, 2015. ISBN 978-3-319-15892-1. doi:10.1007/978-3-319-15892-1_3.

[239] C. R. Shalizi. *Complex Systems Science in Biomedicine*, Methods and Techniques of Complex Systems Science: An Overview, str. 33–114. Springer US, Boston, MA, 2006. ISBN 978-0-387-33532-2. doi:10.1007/978-0-387-33532-2_2.

[240] S. Shangxiong. *A Particle Swarm Optimization (PSO) Algorithm Based on Multi-agent System. Intelligent Computation Technology and Automation (ICICTA), 2008 International Conference on*, tom 2, str. 802–805. Oct 2008. doi:10.1109/ICICTA.2008.367.

[241] Y. Shoham, K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521899435.

[242] M. R. Sierra, C. A. Coello Coello. *Evolutionary Multi-Criterion Optimization: Third International Conference, EMO 2005, Guanajuato, Mexico, March 9-11, 2005. Proceedings*, Improving PSO-Based Multi-objective Optimization Using Crowding, Mutation and -Dominance, str. 505–519. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31880-4. doi:10.1007/978-3-540-31880-4_35.

[243] SmartBear Software. *SoapUI*. `https://www.soapui.org/`. Dostęp: 2016-01-20.

[244] M. Sobczyk. *Statystyka*. Wydawnictwo Naukowe PWN, Warszawa, 2007.

[245] N. Srinivas, K. Deb. *Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms*. Evol. Comput., 2(3):221–248, IX 1994. ISSN 1063-6560. doi:10.1162/evco.1994.2.3.221.

[246] Standard Performance Evaluation Corporation. *SPEC's Benchmarks*. `https://www.spec.org/benchmarks.html`. Dostęp: 2016-01-20.

[247] C. Szabo, Q. Z. Sheng, T. Kroeger, Y. Zhang, J. Yu. *Science in the Cloud: Allocation and Execution of Data-Intensive Scientific Workflows*. J. Grid Comput., 12(2):245–264, VI 2014. ISSN 1570-7873. doi:10.1007/s10723-013-9282-3.

[248] W. A. Tackett. *Genetic Programming for Feature Discovery and Image Discrimination. Proceedings of the 5th International Conference on Genetic Algorithms*, str. 303–311. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-299-2.

[249] Telecom Italia Lab. *Java Agent DEvelopment Framework Documentation*. `http://jade.tilab.com/doc/index.html`. Dostęp: 2016-01-20.

[250] T. Terzidou, T. Tsiatsos, C. Miliou, A. Sourvinou. *Agent Supported Serious Game Environment*. IEEE Transactions on Learning Technologies, PP(99):1–1, 2016. ISSN 1939-1382. doi:10.1109/TLT.2016.2521649.

[251] TOP500.org. *LINPACK Benchmark*. `http://www.top500.org/project/linpack/`. Dostęp: 2016-01-20.

[252] M. Tornow, A. Al-Hamadi, V. Borrmann. *A multi-agent mobile robot system with environment perception and HMI capabilities. Signal and Image Processing Applications (ICSIPA), 2013 IEEE International Conference on*, str. 252–257. Oct 2013. doi:10.1109/ICSIPA.2013.6708013.

[253] B. Twardowski, D. Ryzko. *Multi-agent Architecture for Real-Time Big Data Processing. Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014 IEEE/WIC/ACM International Joint Conferences on*, tom 3, str. 333–337. Aug 2014. doi:10.1109/WI-IAT.2014.185.

[254] Typesafe Inc. *Akka*. `http://akka.io/`. Dostęp: 2016-01-20.

[255] E. Ulungu, J. Teghem, C. Ost. *Efficiency of interactive multi-objective simulated annealing through a case study*. Journal of the Operational Research Society, 49(10):1044–1050, 1998.

[256] UNICORE Forum e.V. *UNICORE*. `https://www.unicore.eu/`. Dostęp: 2016-03-20.

[257] University of California. *SETI@home*. `http://setiathome.ssl.berkeley.edu/`. Dostęp: 2016-01-20.

[258] US National Science Foundation. *Network for Earthquake Engineering and Simulation (NEES)*. `https://nees.org/`. Dostęp: 2016-03-20.

[259] R. F. Van der Wijngaart, M. Frumkin. *NAS Grid Benchmarks Version 1.0*. Rap. tech., NASA Advanced Supercomputing (NAS) Division, 2002. `https://www.nas.nasa.gov/assets/pdf/techreports/2002/nas-02-005.pdf`. Dostęp: 2016-01-20.

[260] J. A. Walker, J. F. Miller. *Solving Real-valued Optimisation Problems Using Cartesian Genetic Programming. Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, str. 1724–1730. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-697-4. doi:10.1145/1276958.1277295.

[261] J. A. Walker, J. F. Miller, R. Cavill. *A Multi-chromosome Approach to Standard and Embedded Cartesian Genetic Programming. Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, str. 903–910. ACM, New York, NY, USA, 2006. ISBN 1-59593-186-4. doi:10.1145/1143997.1144153.

[262] T. Wang, M. Harman, Y. Jia, J. Krinke. *Searching for Better Configurations: A Rigorous Approach to Clone Evaluation. Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, str. 455–465. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-2237-9. doi:10.1145/2491411.2491420.

[263] W. Weimer, T. Nguyen, C. Le Goues, S. Forrest. *Automatically Finding Patches Using Genetic Programming. Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, str. 364–374. IEEE Computer Society, Washington, DC, USA, 2009. ISBN 978-1-4244-3453-4. doi:10.1109/ICSE.2009.5070536.

[264] G. Weiss, red. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, 1999. ISBN 978-0-262-23203-6.

[265] P. A. Whigham. *Grammatically-based Genetic Programming*. J. P. Rosca, red., *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, str. 33–41. 1995.

[266] D. White, A. Arcuri, J. A. Clark. *Evolutionary Improvement of Programs*. Evolutionary Computation, IEEE Transactions on, 15(4):515–538, Aug 2011. ISSN 1089-778X. doi:10.1109/TEVC.2010.2083669.

[267] D. Wieers. *Dstat*. `http://dag.wiee.rs/home-made/dstat/`. Dostęp: 2016-03-20.

[268] R. J. Wilson. *The European DataGrid project. Proceedings of APS / DPF / DPB Summer Study on the Future of Particle Physics (Snowmass 2001), Snowmass, Colorado, 30 Jun - 21 Jul 2001*. 2001.

[269] M. Wooldridge. *Introduction to MultiAgent Systems*. John Wiley & Sons, June 2002. ISBN 047149691X.

[270] M. J. Wooldridge, N. R. Jennings. *Intelligent Agents: Theory and Practice*. The Knowledge Engineering Review, 10(2):115–152, 1995.

[271] World Wide Web Consortium (W3C). *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. `http://www.w3.org/TR/soap12-part1/`, April 2007. Dostęp: 2016-01-20.

[272] World Wide Web Consortium (W3C). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. `http://www.w3.org/TR/REC-xml/`, November 2008. Dostęp: 2016-01-20.

[273] F. Xie, A. Song, V. Ciesielski. *Applications of Evolutionary Computation: 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3-5, 2013. Proceedings*, Human Action Recognition from Multi-Sensor Stream Data by Genetic Programming, str. 418–427. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-37192-9. doi:10.1007/978-3-642-37192-9_42.

[274] H. Yu, Z. Shen, C. Leung. *From Internet of Things to Internet of Agents. Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, str. 1054–1057. Aug 2013. doi:10.1109/GreenCom-iThings-CPSCom.2013.179.

[275] J. Yu, R. Buyya. *A Taxonomy of Scientific Workflow Systems for Grid Computing*. SIGMOD Rec., 34(3):44–49, IX 2005. ISSN 0163-5808. doi:10.1145/1084805.1084814.

[276] A. Zacniewski. *Optymalizacja alokacji modułów programistycznych w rozproszonym systemie szkolenia wojskowego*. Rozprawa doktorska, Wydział Elektroniki, Telekomunikacji i Informatyki, Politechnika Gdańska, 2011.

[277] L. Zha, W. Li, H. Yu, X. Xie, N. Xiao, Z. Xu. *System Software for China National Grid*. H. Jin, D. Reed, W. Jiang, red., *Network and Parallel Computing*, tom 3779 z serii *Lecture Notes in Computer Science*, str. 14–21. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-29810-6. doi:10.1007/11577188_3.

[278] R. Zheng, N. Chakraborty, T. Dai, K. Sycara. *Multiagent Negotiation on Multiple Issues with Incomplete Information: Extended Abstract. Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, str. 1279–1280. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2013. ISBN 978-1-4503-1993-5.

[279] E. Zitzler, M. Laumanns, L. Thiele. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. Rap. tech., Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland, 2001.

[280] E. Zitzler, L. Thiele. *Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach*. Evolutionary Computation, IEEE Transactions on, 3(4):257–271, Nov 1999. ISSN 1089-778X. doi:10.1109/4235.797969.

Wykaz rysunków

1.1 Podstawowe rodzaje metaheurystyk [184] 11
1.2 Wybrane rodzaje krzyżowania w algorytmach genetycznych [61] 14
1.3 Selekcja ruletki 15
1.4 Przykłady drzew reprezentujących wybrane S-wyrażenia 19
1.5 Diagram algorytmu ewolucyjnego 23
1.6 Wybrane procedury nadawania rang [276] 25
1.7 Wyznaczanie wartości miary oddalenia i miary zagęszczenia osobników w algorytmach z rodziny NSGA 26
1.8 Diagram procedury aktualizacji populacji w NSGA-II i NSGA-III [62, 63] 28
1.9 Reguła trójwartościowej ruletki dla parametrów $HMCR = 0{,}7$ oraz $PAR = 0{,}2$ [209] 31
1.10 Diagram algorytmu harmonicznego dla zagadnień optymalizacji jednokryterialnej [209] 32

2.1 Przepływ informacji z perceptorów i decyzji przy wyborze akcji w architekturach warstwowych 38
2.2 Wybrane strategie realizacji zachowań [158] 39
2.3 Wstępne przetwarzanie nowych informacji przez agenta [169] 41
2.4 Zarządzanie zasobami za pomocą agentów [49] 46
2.5 Doradczy monitor wydajności PMA [49] 46
2.6 Agentowe przetwarzanie *Big Data* w czasie rzeczywistym w architekturze lambda [253] 47
2.7 Zastosowanie systemów agentowych w inteligentnych miastach [145] 48

3.1 Podmioty zrzeszone w wirtualnych organizacjach 51
3.2 Kluczowy fragment strony głównej projektu *Comcute* [131] 55
3.3 Zasada działania systemu *Comcute* 56
3.4 Przykłady powiązań między agentami warstwy pośredniczącej w gridzie *Comcute* . 59

3.5 Czas przetwarzania wybranej liczby paczek danych przez agenta klasy S w zależności od liczby paczek oraz od łącznej liczby agentów klasy S przetwarzających taką samą liczbę paczek w ramach zadania obliczeniowego 63

3.6 Obciążenie komunikacyjne agenta klasy S w zależności od liczby obsługiwanych paczek danych i liczby agentów klasy S przetwarzających taką samą liczbę paczek w ramach zadania obliczeniowego 64

3.7 Obciążenie agenta klasy S w zależności od liczby żądań na sekundę od węzłów obliczeniowych 65

3.8 Czas przetwarzania agenta zarządzającego klasy W w zależności od liczby żądań napływających do powiązanych agentów dystrybucyjnych typu S 66

3.9 Czasy przetwarzania wybranej liczby paczek przez agenta klasy W w zależności od liczby powiązanych agentów typu S oraz od liczby paczek 67

3.10 Czasy komunikacji pomiędzy agentami klasy W w zależności od czasu realizacji zadania 69

3.11 Obciążenie węzła, w którym działa agent klasy W 70

3.12 Obciążenie węzła, w którym działa agent klasy S 71

3.13 Obciążenie sterty maszyny wirtualnej *Javy* przez agenta klasy S 72

4.1 Okresy dostępności i niedostępności komputera wolontariusza [136] 87

4.2 Wybrane oceny strategii zespołu 12 agentów w przestrzeni kryterialnej ($\hat{Z}_{\max}$, Ξ) dla gridu o sześciu węzłach i realizacji dwóch zadań obliczeniowych 98

5.1 Mapowanie genotyp-fenotyp w programowaniu genetycznym do wyznaczania strategii zespołu agentów 110

5.2 Wartości funkcji celu dla rozwiązań wyznaczonych przez agenta $GP(F_1, \mu, \boldsymbol{\mathcal{F}})$ . . . 111

5.3 Uśredniona zbieżność wartości funkcji celu wyznaczonych za pomocą agentów $GP(F_1, \mu, \boldsymbol{\mathcal{F}})$ 112

5.4 Zastosowanie operatorów *OR*, *AND* i *NOT* do implementacji operatora *XOR* . . . 113

5.5 Przykładowa procedura warunkowa *IF* w programowaniu genetycznym 113

5.6 Porównanie zbieżności agentów $GP(F_1, \mu, \boldsymbol{\mathcal{F}})$ i $GP(F_1, \mu, \boldsymbol{\mathcal{F}}^+)$ 114

5.7 Operatory obsługi zmiennych w drzewie programu 115

5.8 Przykładowa procedura *WHILE* w programowaniu genetycznym 116

5.9 Drzewo programu wyznaczającego NWD i odpowiadający mu pseudokod 117

5.10 Porównanie zbieżności programowania genetycznego dla trzech zbiorów procedur: $\boldsymbol{\mathcal{F}}$, $\boldsymbol{\mathcal{F}}^+$ oraz $\boldsymbol{\mathcal{F}}^{++}$ 117

5.11 Przykładowa procedura *FOR* w programowaniu genetycznym 118

5.12 Porównanie zbieżności programowania genetycznego dla czterech zbiorów procedur: $\boldsymbol{\mathcal{F}}$, $\boldsymbol{\mathcal{F}}^{+}$, $\boldsymbol{\mathcal{F}}^{++}$ oraz $\boldsymbol{\mathcal{F}}^{+++}$ 118
5.13 Rangi wyznaczone za pomocą wybranych procedur 119
5.14 Oceny rozwiązań na różnych etapach programowania genetycznego 122
5.15 Zbieżność programowania genetycznego podczas 100 epok 122
5.16 Diagram wielokryterialnego programowania genetycznego z archiwum MOGPA . . 124
5.17 Oceny rozwiązań zakwalifikowane do archiwum przy różnych miarach zagęszczenia 126
5.18 Utrata rozwiązań niezdominowanych i ich przywrócenie z archiwum 128
5.19 Wykorzystanie archiwum do utrzymania różnorodności w populacji 129
5.20 Schemat wielokryterialnej metaheurystyki programowania genetycznego MOGPA . 130
5.21 Punkty charakterystyczne w przestrzeni kryterialnej dla zagadnienia optymalizacji strategii sieci agentów w gridzie 135
5.22 Schemat blokowy metody DMGP 144
5.23 Rozproszone wdrożenie strategii przez zespół agentów warstwy pośredniczącej w gridzie *Comcute* 146
5.24 Interfejs Aplikacji Administratora Gridu AAG'16 148
5.25 Zbieżność programowania genetycznego w zagadnieniu dwukryterialnym 151
5.26 Rozmieszczenie agentów i zasobów w gridzie zgodne ze strategią x_1^{GP} 151
5.27 Porównanie rozwiązań wyznaczonych przez wybrane metaheurystyki 152
5.28 Porównanie ocen strategii wyznaczonych po rozluźnieniu wymagań interesariusza . 153
5.29 Oceny rozwiązań niezdominowanych na różnych etapach programowania genetycznego 156
5.30 Porównanie wyników wyznaczonych za pomocą programowania genetycznego z rezultatami otrzymanymi za pomocą innych metaheurystyk 156
5.31 Wybór rozwiązania kompromisowego ze zbioru *Pareto* 158
5.32 Porównanie zbieżności metod DMGP i NSGA-II 161
5.33 Porównanie zbieżności metod DMGP i NSGA-II w wypadku dzierżawy węzłów w chmurze 162

A.1 Zbieżność programowania genetycznego dla czterech problemów *De Jonga* 199
A.2 Wartości funkcji celu dla kolejnych osobników w zagadnieniu (A.3), $G_{\mathrm{size}} = 20$ . . . 201
A.3 Zbieżność programowania genetycznego dla problemów o 30-wymiarowej przestrzeni przeszukiwań 203

B.1 Wybór kryteriów optymalizacji w aplikacji AAG'16 205
B.2 Widok szczegółowy realizowanej symulacji dla zagadnienia dwukryterialnego 206

Wykaz algorytmów

1.1 Diagram programowania ewolucyjnego 13
1.2 Diagram algorytmu genetycznego 16
1.3 Diagram strategii ewolucyjnej 18
1.4 Diagram programowania genetycznego 21
2.1 Diagram interpretera BDI 38
3.1 Diagram symulatora przetwarzania paczek danych przez agenty z grupy $\mathcal{S}_m$ 61

Wykaz tabel

3.1 Parametry wybranych rodzajów serwerów dostępnych na rynku lub zainstalowanych w gridzie *Comcute* 57

3.2 Obciążenie agenta klasy S w zależności od liczby paczek danych do przetworzenia oraz od łącznej liczby agentów klasy S w gridzie 63

3.3 Obciążenie agenta klasy S w zależności od liczby żądań na sekundę od węzłów obliczeniowych w czasie przetwarzania 32 tys. paczek danych 65

3.4 Metody szacowania obciążeń agentów typu W i S w gridzie *Comcute* 74

4.1 Indeksy selektorów ograniczeń dla strategii zespołu agentów warstwy pośredniczącej gridu 93

4.2 Kryteria oceny jakości strategii zespołu agentów warstwy pośredniczącej gridu . . . 94

4.3 Oceny rozwiązań niezdominowanych wyznaczonych za pomocą metody losowej . . 99

5.1 Wartości miar zagęszczenia GCD i DGCD dla przykładowego zbioru strategii kandydujących do archiwum 126

5.2 Przykładowy scenariusz wykonania zadań w gridzie 140

5.3 Wybrane profile sprzętowe dla maszyn wirtualnych w chmurze *Amazon EC2* 142

5.4 Dostrajanie strategii w toku programowania genetycznego 151

5.5 Wspólne obszary zbieżności algorytmów MOGPA, HS, NSGA-II i OMOPSO 157

5.6 Etapy scenariusza realizacji zadań 159

5.7 Oceny strategii wyznaczonych przez program χ^* 161

5.8 Oceny wyznaczonych strategii 162

5.9 Testowe stany gridu *Comcute* i oceny strategii wyznaczonych przez program χ^* . . 163

A.1 Porównanie programowania genetycznego z metodą PSO i algorytmem ewolucyjnym 203

Dodatek A. Porównanie programowania genetycznego z innymi algorytmami

W celu weryfikacji jakości wyników wyznaczanych za pomocą zaimplementowanego programowania genetycznego, zastosowano je do testowych problemów optymalizacji sformułowanych przez *De Jonga* [61]. Odnoszą się one do liniowych oraz nieliniowych zagadnień z ciągłymi zmiennymi decyzyjnymi. Wyniki uzyskane za pomocą programowania genetycznego porównano z rezultatami otrzymanymi za pomocą algorytmu ewolucyjnego *De Jonga* oraz algorytmu harmonicznego [209]. Na rysunku A.1 zaprezentowano porównanie zbieżności rozpatrywanych metod dla czterech zagadnień optymalizacji.

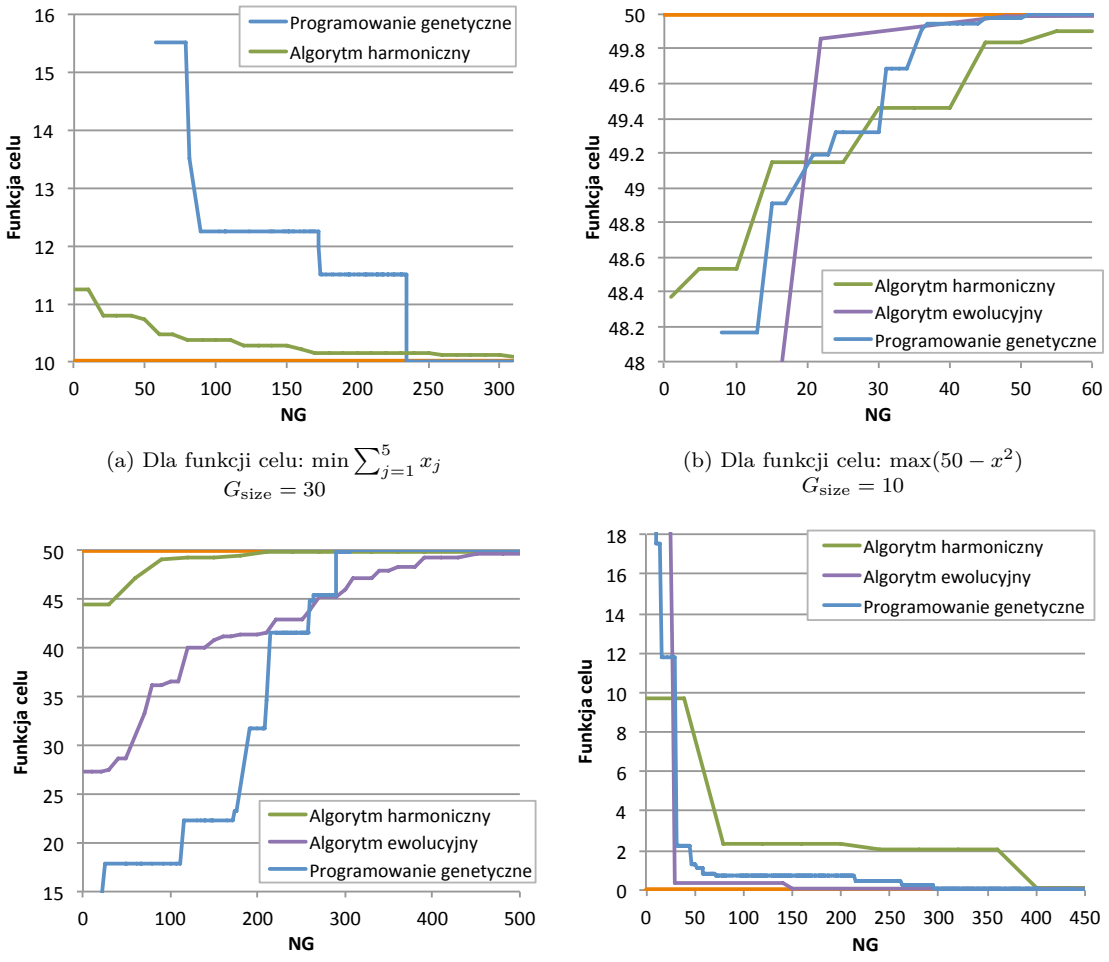
Na rysunku A.1a przedstawiono rezultaty minimalizacji funkcji celu dla problemu programowania liniowego z pięcioma zmiennymi decyzyjny, który sformułowano następująco:

$$\min \sum_{j=1}^{5} x_j, \tag{A.1}$$

przy ograniczeniach: $1 \leqslant x_1 \leqslant 4;\ 4 \leqslant x_2 \leqslant 7;\ 2 \leqslant x_3 \leqslant 3;\ 2 \leqslant x_4 \leqslant 3;\ 1 \leqslant x_5 \leqslant 2;$

$x_1, x_2, x_3, x_4, x_5 \in \mathbb{R}.$

Na rysunku A.1a zobrazowano wartości funkcji celu dla najlepszych osobników w zależności od liczby mutacji *NG*. Wartość optymalna w rozważanym zagadnieniu wynosi 10 dla rozwiązania $x = [1, 4, 2, 2, 1]^T$. Wykres wartości funkcji celu dla programowania genetycznego rozpoczyna się od osobnika nr 57, co wynika z faktu, że żaden z wcześniejszych osobników nie spełniał wszystkich ograniczeń problemu (A.1). Zarządzanie ograniczeniami jest trudniejsze w programowaniu genetycznym niż w algorytmach ewolucyjnych wykorzystujących fenotypowe reprezentacje rozwiązań. Przykładowo, w algorytmie harmonicznym na etapie generowania początkowych improwizacji zmienne decyzyjne losowane są wyłącznie z dopuszczalnych zakresów. Wszystkie rozwiązania z populacji początkowej spełniają zatem ograniczenia. Osobniki, które nie spełniają wymagań, mogą zostać skonstruowane za pomocą krzyżowania lub mutacji w kolejnych epokach.



(a) Dla funkcji celu: $\min \sum_{j=1}^{5} x_j$
$G_{\text{size}} = 30$

(b) Dla funkcji celu: $\max(50 - x^2)$
$G_{\text{size}} = 10$

(c) Dla funkcji celu: $\max(x_1^2 + x_2^2)$
$G_{\text{size}} = 20$

(d) Dla funkcji celu: $\min \left[100(x_1^2 - x_2)^2 + (1 - x_1)^2\right]$
$G_{\text{size}} = 20$

Rysunek A.1: Zbieżność programowania genetycznego dla czterech problemów *De Jonga*
Źródło: opracowanie własne na podstawie [61, 209]

W przypadku programowania genetycznego w populacji początkowej konstruowane są losowe drzewa wyrażeń, które stanowią genotypową reprezentację rozwiązań. Weryfikacja, czy zwracane przez drzewa wartości są w dopuszczalnych zakresach, wymaga mapowania genotyp-fenotyp (wyznaczenia wartości wyrażeń), które odbywa się dopiero podczas oceny osobników. Nie ma zatem gwarancji, że osobniki z populacji początkowej będą spełniały ograniczenia – dopiero presja selekcyjna ukierunkowuje wyszukiwanie na obszar rozwiązań dopuszczalnych. Z tego powodu programowanie genetyczne w początkowych epokach zazwyczaj cechuje się gorszymi wartościami funkcji celu w porównaniu do wartości otrzymanych za pomocą innych metod.

Dla rozważanego zagadnienia wykorzystano zbiór operatorów: $\mathcal{F} = \{+, -, *, /\}$. Zbiór symboli terminalnych zawierał wartości ograniczeń dla zmiennych decyzyjnych oraz wartości losowe. Przyjęto rozmiar populacji $G_{\text{size}} = 30$, drzewa populacji początkowej generowane były

za pomocą metody hybrydowej, a ich maksymalna głębokość wynosiła 5. Za pomocą programowania genetycznego uzyskano rozwiązanie optymalne po wyznaczeniu 235 osobników w czasie 0,039 s. Dla porównania algorytm harmoniczny potrzebował 310 improwizacji, aby zwrócić rozwiązanie z założoną dokładnością 0,1 w czasie 0,047 s [209]. Algorytm *simplex* zaimplementowany przez *Frontline Systems* [85] w podobnym czasie wyznaczył rozwiązanie dokładne [209].

Na rysunku A.1b przedstawiono maksymalizację funkcji celu dla problemu programowania nieliniowego sformułowanego, jak niżej: [61]:

$$\max(50 - x^2), \tag{A.2}$$

przy ograniczeniach: $-5 \leqslant x \leqslant 5;\ x \in \mathbb{R}$.

W programowaniu genetycznym wykorzystano ten sam zbiór operatorów, co dla problemu (A.1). Zbiór terminali zawierał wartości ograniczeń oraz liczby losowe. Rozmiar populacji ustalono na $G_{\text{size}} = 10$. Rozwiązanie optymalne $x = 0$ cechuje się wartością 50 dla funkcji celu. Programowanie genetyczne po 36 mutacjach zwróciło rozwiązanie z dokładnością do 0,1, dla którego funkcja celu przyjęła wartość 49,91. Po wygenerowaniu 51 osobników uzyskano dokładność do dwóch miejsc po przecinku, a po 200 mutacjach – do 10^{-8}. Algorytm harmoniczny dokładność na poziomie 0,1 uzyskał po 55 improwizacjach, a po wykonaniu 1000 kroków zakończył działanie z rozwiązaniem o dokładności 10^{-6} [209].

Algorytm ewolucyjny *De Jonga* wykorzystywał populację 10 osobników i operator mutacji oparty o rozkład *Gaussa* $G(0,\ s/\sqrt{2/\pi})$, gdzie s oznacza średnie tempo mutacji ustalone na poziomie $s = 1{,}0$. Po 48 mutacjach algorytm zwrócił rozwiązanie z dokładnością do 0,1, a po wygenerowaniu 131 osobników poprawił ją do 0,01. Przy limicie 1000 mutacji najlepszy wynik $x = 0{,}00275$ uzyskano dla osobnika nr 762 przy funkcji celu o wartości 49,999992 [61].

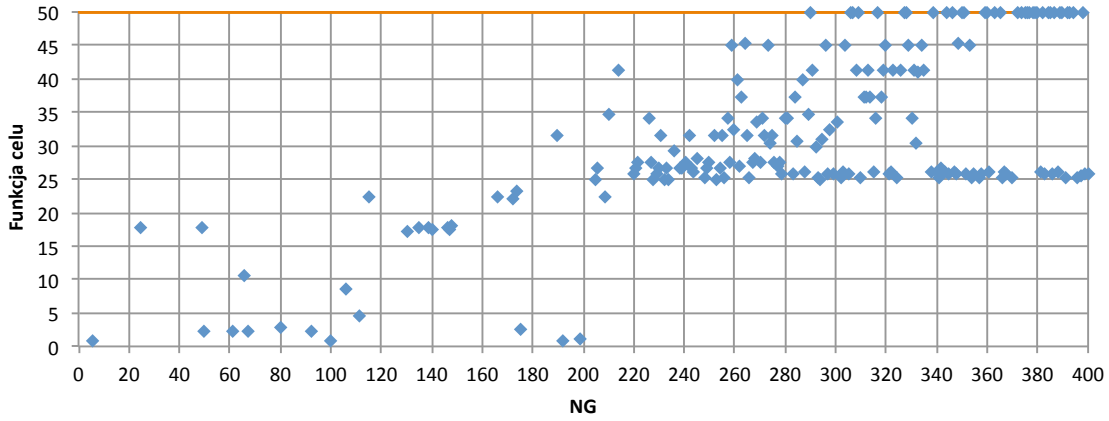
Rezultaty maksymalizacji funkcji celu dla kolejnego zagadnienia programowania nieliniowego zaprezentowano na rysunku A.1c. Problem optymalizacji sformułowany jest następująco [61]:

$$\max(x_1^2 + x_2^2), \tag{A.3}$$

przy ograniczeniach: $-5 \leqslant x_1 \leqslant 5;\ -5 \leqslant x_2 \leqslant 5;\ x_1, x_2 \in \mathbb{R}$.

Rozpatrywane zagadnienie cechuje się czterema rozwiązaniami optymalnymi o współrzędnych: $(-5; -5)$, $(5, 5)$, $(5; -5)$, $(5, 5)$. Maksymalna wartość funkcji celu wynosi 50. Widoczne jest, że programowanie genetyczne w początkowym etapie ewolucji zwraca znacząco gorsze wyniki od pozostałych metod. Dopiero po wygenerowaniu 220 osobników uzyskiwane wartości funkcji celu zbliżają się do wyznaczonych przez algorytm ewolucyjny *De Jonga*. Równocześnie wyniki z programowania genetycznego szybciej dążą do wartości optymalnej niż wyniki zwracane przez porównywane algorytmy. Rozwiązanie z dokładnością do 0,1 uzyskano dla 290-tego osobnika (funkcja celu: 49,902), a osobnik nr 306 reprezentuje rozwiązanie optymalne z funkcją celu o wartości 50. Algorytm harmoniczny potrzebował 390 improwizacji, aby uzyskać rozwiązanie z dokładnością do 0,1 [209], a algorytm ewolucyjny zwrócił wynik tej jakości dopiero dla 617-tego osobnika [61].

Przebieg eksploracji przestrzeni rozwiązań przez programowanie genetyczne zobrazowano na rysunku A.2. Przedstawiono wartości funkcji celu dla kolejnych osobników. Punkty wykreślono jedynie dla tych rozwiązań, które spełniają ograniczenia w problemie (A.3). Widoczne jest, że w początkowym etapie ewolucji tylko nieliczne osobniki należą do zbioru rozwiązań dopuszczalnych. W miarę postępu obliczeń presja selekcyjna ukierunkowuje populację na obszar przestrzeni przeszukiwań obejmujący rozwiązania, które spełniają przyjęte ograniczenia. Równocześnie uzyskiwane wartości funkcji celu ulegają systematycznej poprawie. Od dwusetnego osobnika systematycznie rośnie również średnia wartość funkcji celu. W końcowej części wykresu większość osobników jest zgromadzona w okolicach rozwiązania optymalnego.



Rysunek A.2: Wartości funkcji celu dla kolejnych osobników w zagadnieniu (A.3), $G_{\text{size}} = 20$
Źródło: opracowanie własne

Na rysunku A.1d przedstawiono wyniki działania porównywanych algorytmów dla instancji zagadnienia minimalizacji funkcji *Rosenbrocka*, które sformułowano, jak niżej [61]:

$$\min \left[100(x_1^2 - x_2)^2 + (1 - x_1)^2\right], \tag{A.4}$$

przy ograniczeniach: $-10 \leqslant x_1 \leqslant 10;\ -10 \leqslant x_2 \leqslant 10;\ x_1, x_2 \in \mathbb{R}.$

Globalne minimum funkcji *Rosenbrocka* położone jest w punkcie o współrzędnych (1,1), dla którego funkcja przyjmuje wartość 0. Programowanie genetyczne dostrojono podobnie, jak we wcześniejszych przykładach i uzyskano rozwiązanie z dokładnością do 0,1 po wygenerowaniu 296 osobników (funkcja celu: 0,027). Algorytm harmoniczny wynik podobnej jakości wyznaczył po 400 improwizacjach [209]. Z kolei algorytm ewolucyjny *De Jonga* potrzebował zaledwie 150 mutacji [61].

Na podstawie przeprowadzonych eksperymentów stwierdza się, że programowanie genetyczne zazwyczaj wyznacza gorszej jakości rozwiązania na wczesnych etapach ewolucji, co wynika z braku mechanizmów weryfikacji osobników populacji początkowej. Niemniej jednak, odpowiednio skonstruowana funkcja *fitness* uwzględniająca karę za niespełnienie ograniczeń pozwala ukierunkować poszukiwanie na obszar rozwiązań dopuszczalnych. W efekcie końcowe wyniki są porównywalnej lub wyższej jakości niż te wyznaczane za pomocą algorytmu harmonicznego i algorytmów ewolucyjnych.

Po zweryfikowaniu poprawności zaimplementowanej metody w zagadnieniach sformułowanych przez *De Jonga*, w kolejnych eksperymentach badano trudniejsze problemy testowe. Porównano rezultaty programowania genetycznego z wynikami metody PSO (ang. *Particle Swarm Optimization*) przedstawionymi przez *Clerca* i *Kennedy'ego* [53] oraz rezultatami algorytmu ewolucyjnego *Angeline'a* [16], które w pracy [53] przytoczono jako wyniki odniesienia. Zweryfikowano zbieżność programowania genetycznego w zagadnieniu minimalizacji uogólnionej funkcji *Rosenbrocka*:

$$\min \left[\sum_{i=1}^{N} 100 \left(x_{i+1} - x_i^2 \right)^2 + (1 - x_i)^2 \right], \tag{A.5}$$

przy ograniczeniach: $-10 \leqslant x_i \leqslant 10;\ x_i \in \mathbb{R},\ i = \overline{1,N}.$

Przyjęto wartość parametru $N = 30$, co odpowiada przeszukiwaniu 30-wymiarowej przestrzeni rozwiązań. Wyniki metody PSO uzyskano dla populacji o rozmiarze 20 osobników i 2000 iteracji, co przekłada się na 40000 mutacji. W programowaniu genetycznym wykorzystano populację o rozmiarze $G_{\text{size}} = 100$ osobników i limit liczby epok na poziomie $G_{\max} = 400$, co daje taką samą liczbę mutacji, jak w metodzie PSO. Drzewa populacji początkowej budowano zgodnie z metodą konstruowania pełnych drzew, a ich wysokość mieściła się w przedziale $[4, 10]$.

Zbiór operatorów zawierał cztery podstawowe operatory arytmetyczne $(+, -, *, /)$ oraz operator podnoszenia do kwadratu. Do zbioru terminali należały: wartości ograniczeń, stałe z funkcji celu, parametr N oraz liczby losowe. Najlepsze rozwiązanie cechowało się wartością funkcji celu 129,160 i zostało uzyskane dla osobnika nr 37661. Porównanie z rezultatami innych metod zaprezentowano w tabeli nr A.1. Programowanie genetyczne wyznaczyło gorszy wynik niż metoda PSO, jednak w porównaniu do podstawowego algorytmu ewolucyjnego *Angeline'a* [16] wartość funkcji celu została poprawiona o rząd wielkości. Zbieżność programowania genetycznego pokazano na rysunku A.3a.

W kolejnym eksperymencie wykorzystano funkcję *Rastrigina*, a zagadnienie optymalizacji sformułowano następująco:

$$\min \left[10 + \sum_{i=1}^{N} \left(x_i^2 - 10 \cos\left(2\pi x_i\right) \right) \right], \tag{A.6}$$

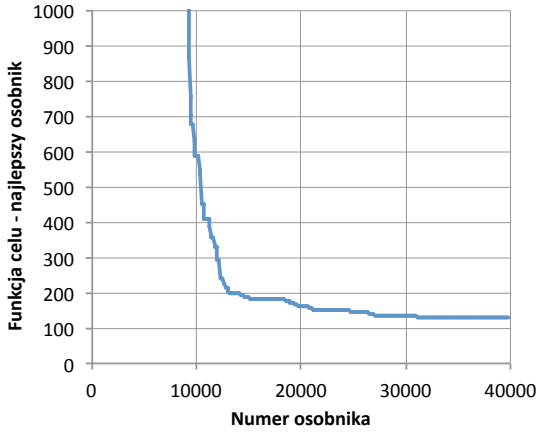
przy ograniczeniach: $-5{,}12 \leqslant x_i \leqslant 5{,}12;\ x_i \in \mathbb{R},\ i = \overline{1,N}.$

Zbiór terminali w programowaniu genetycznym rozszerzono o liczbę π, a zbiór procedur o funkcję trygonometryczną *cosinus*. Pozostałe ustawienia zachowano, jak w poprzednim eksperymencie. Za pomocą programowania genetycznego uzyskano rozwiązanie o ocenie 22,3816. W tabeli nr A.1 widoczne jest, że dla zagadnienia minimalizacji funkcji *Rastrigina* programowanie genetyczne uzyskuje najlepszy wynik spośród porównywanych metod.

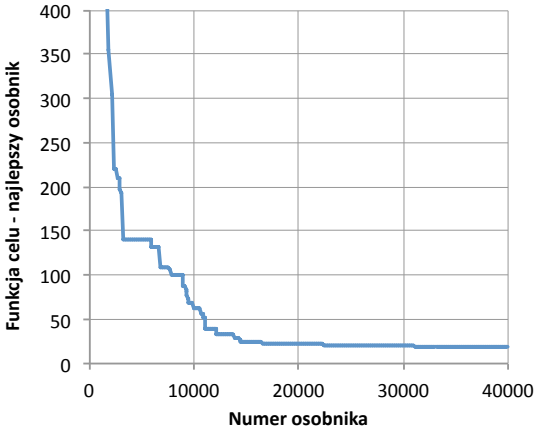
Tabela A.1: Porównanie programowania genetycznego z metodą PSO i algorytmem ewolucyjnym

Funkcja testowa $N = 30$	Wyznaczone wartości funkcji celu		
	Programowanie genetyczne	PSO [53]	Algorytm ewolucyjny [16]
Rosenbrocka	129,160	50,193877	1610,359
Rastrigina	22,3816	82,95618	46,4689
Griewanka	0,07843	0,003944	0,4033

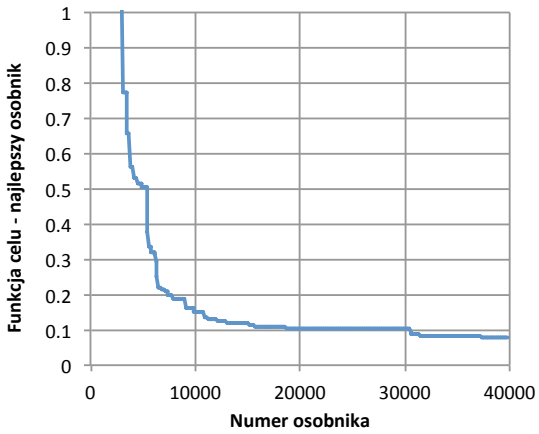
Źródło: opracowanie własne na podstawie [16, 53]



(a) Zagadnienie (A.5): uogólniona funkcja *Rosenbrocka*



(b) Zagadnienie (A.6): funkcja *Rastrigina*



(c) Zagadnienie (A.7): funkcja *Griewanka*

Rysunek A.3: Zbieżność programowania genetycznego dla problemów o 30-wymiarowej przestrzeni przeszukiwań
Źródło: opracowanie własne

Ostatnie z zagadnień testowych odnosi się do minimalizacji funkcji *Griewanka*, jak niżej:

$$\min \left[1 + \frac{1}{4000} \sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos\left(\frac{x_i}{\sqrt{i}} \right) \right], \tag{A.7}$$

przy ograniczeniach: $-300 \leqslant x_i \leqslant 300;\ x_i \in \mathbb{R},\ i = \overline{1,N}$.

Do zbioru procedur należały: cztery podstawowe operatory arytmetyczne, operator podnoszenia do kwadratu, operator pierwiastka kwadratowego oraz funkcja trygonometryczna *cosinus*. W zbiorze terminali uwzględniono wartości ograniczeń, stałe z funkcji celu oraz parametr N. Uzyskano ocenę 0,07059 dla osobnika nr 38942. Jest to rezultat gorszy od wyników uzyskanych za pomocą metody PSO, natomiast jest lepszy niż w przypadku algorytmu ewolucyjnego (tabela nr A.1). Przebieg wyszukiwania rozwiązania zobrazowano na rysunku A.3c.

We wszystkich przebiegach programowania genetycznego, które przedstawiono na rysunku A.3, widoczne są dwie fazy przeszukiwania przestrzeni rozwiązań. Pierwsza faza charakteryzuje się szybką poprawą wartości funkcji celu w początkowym etapie działania metaheurystyki. Następnie odbywa się dostrajanie rozwiązań. W oparciu o uzyskane wyniki wykazano poprawność i zbieżność opracowanego programowania genetycznego, co uprawnia do zastosowania go również w odniesieniu do trudniejszych problemów optymalizacji.

Dodatek B. Aplikacja Administratora Gridu AAG'16

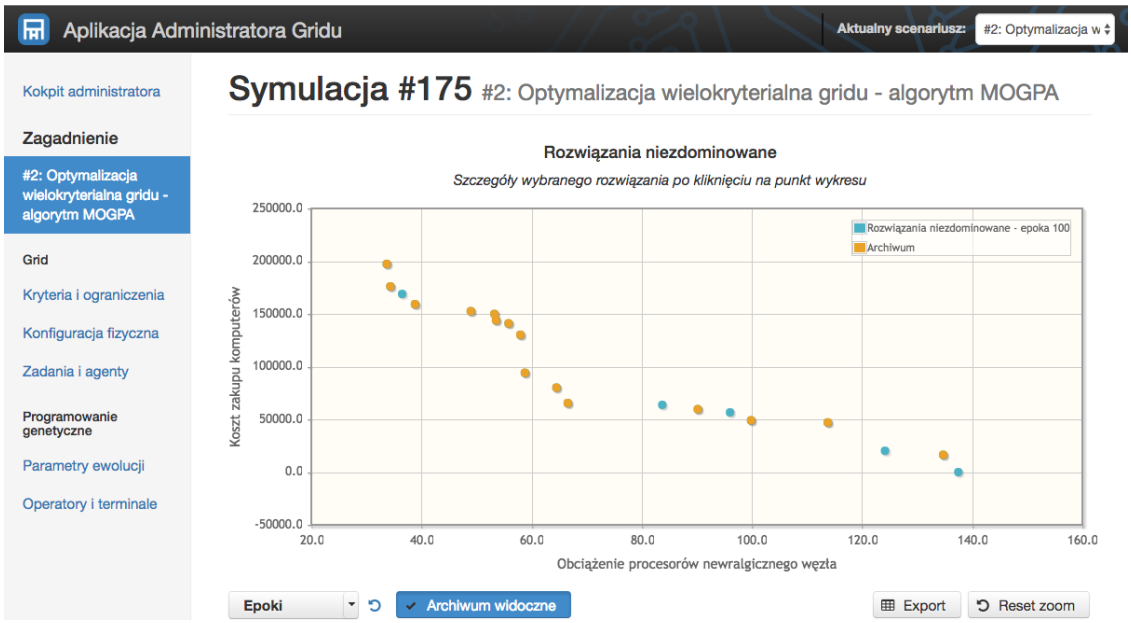
Opracowana *Aplikacja Administratora Gridu* AAG'16 jest dostępna do pobrania na stronie internetowej `http://genetic-programming.pl`. Wersja demonstracyjna, którą wdrożono na serwerach systemu *Comcute*, jest dostępna pod adresem: `http://aag.comcute.eti.pg.gda.pl/`. Dostęp do wersji demonstracyjnej jest możliwy przy użyciu loginu: *interesariusz* oraz hasła: *aag16*.

Posługując się aplikacją AAG'16 interesariusz wybiera zagadnienie optymalizacji, a następnie konfiguruje istotne parametry metaheurystyki do wyznaczenia rozwiązań. W szczególności następuje wybór kryteriów optymalizacji i ograniczeń (rysunek B.1). Konfiguracji podlega również specyfikacja zadań obliczeniowych, agentów w gridzie i dostępnych typów komputerów.

	n	Kryterium cząstkowe F_n	Oznaczenie	Optymalizacja
☑	1	Obciążenie procesorów newralgicznego węzła	$\hat{Z}_{\max}$	MIN
☐	2	Obciążenie komunikacyjne newralgicznego węzła	$\tilde{Z}_{\max}$	MIN
☐	3	Łączne obciążenie procesorów CPU w gridzie	$\hat{Z}_{\text{suma}}$	MIN
☐	4	Łączne obciążenie komunikacyjne w gridzie	$\tilde{Z}_{\text{suma}}$	MIN
☐	5	Łączna wydajność gridu	$\Theta(x)$	MAX
☑	6	Koszt zakupu komputerów	Ξ	MIN
☐	7	Koszt przetwarzania danych i komunikacji	$K(x)$	MAX
☐	8	Stopień rozproszenia modułów	$\eta(x)$	MAX
☐	9	Wielkość dostępnej pamięci RAM w newralgicznym hoście	$\kappa_{\min}^{\text{RAM}}(x)$	MAX
☐	10	Wielkość dostępnej pamięci HDD w newralgicznym hoście	$\kappa_{\min}^{\text{HDD}}(x)$	MAX
☐	11	Wielkość dostępnej pamięci SSD w newralgicznym hoście	$\kappa_{\min}^{\text{SSD}}(x)$	MAX
☐	12	Wielkość łącznej rezerwy pamięci RAM	$M^{\text{RAM}}(x)$	MAX
☐	13	Wielkość łącznej rezerwy pamięci HDD	$M^{\text{HDD}}(x)$	MAX
☐	14	Wielkość łącznej rezerwy pamięci SSD	$M^{\text{SSD}}(x)$	MAX

Rysunek B.1: Wybór kryteriów optymalizacji w aplikacji AAG'16
Źródło: opracowanie własne

Istnieje również możliwość dostrojenia parametrów programowania genetycznego, takich jak: rozmiar populacji, głębokość drzew, sposób budowania populacji początkowej, procedura nadawania rang oraz ustawienia archiwum. Po zakończeniu konfiguracji następuje uruchomienie *agenta-solwera*, który wyznacza rozwiązania sformułowanego problemu optymalizacji. Przebieg poszukiwania rozwiązań może być śledzony w trakcie działania algorytmu (rysunek B.2). Możliwa jest weryfikacja bieżącego zbioru rozwiązań niezdominowanych, ich ocen i wartości ograniczeń.



Rysunek B.2: Widok szczegółowy realizowanej symulacji dla zagadnienia dwukryterialnego
Źródło: opracowanie własne

Aplikacja AAG'16 wykorzystuje *MOEA Framework* [113] jako podstawę do implementacji programowania genetycznego. Podstawowy algorytm dostępny w bibliotece rozbudowano w celu uwzględnienia specyfiki metaheurystyki MOGPA. Modyfikacje obejmują również zrównoleglenie etapu oceny osobników w populacji na wiele wątków, co istotnie przyspiesza działanie programowania genetycznego. Dokonano implementacji nowych operatorów, a etap uruchamiania drzew programów rozszerzono o możliwość weryfikacji czasu wykonania.