POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

Imię i nazwisko autora rozprawy: mgr inż. Jacek Paluszak
Dyscyplina naukowa: Informatyka

ROZPRAWA DOKTORSKA

Tytuł rozprawy w języku polskim: Optymalizacja wykorzystania zasobów w systemach rozproszonych o architekturze typu grid

Tytuł rozprawy w języku angielskim: Optimizing the use of resources in distributed systems with grid architectures

Promotor *podpis*	Drugi promotor *podpis*
Dr hab. inż. Jerzy Balicki, prof. nadzw. PG	<Tytuł, stopień, imię i nazwisko>
Promotor pomocniczy *podpis*	Kopromotor *podpis*
<Stopień, imię i nazwisko>	<Tytuł, stopień, imię i nazwisko>

Gdańsk, rok 2015

Spis treści

Wykaz ważniejszych akronimów i oznaczeń

BOINC - ang. *Berkeley Open Infrastructure for Network Computing;*
GIMPS - ang. *Great Internet Mersenne Prime Search;*
GRAM - ang. *Globus/Grid Resource Allocation Management;*
GSI - ang. *Grid Security Infrastructure;*
GT - ang. *Globus Toolkit;*
MDS - ang. *Monitoring and Discovery Service;*
PL-Grid - *Polska Infrastruktura Gridowa*;
SOAP - ang. *Simple Object Access Protocol;*
$C=[c_{vr}]_{V\times 2}$ - macierz zapotrzebowania na pamięć przez moduły;
$\tilde{D}=[\tilde{d}_{jr}]_{J\times 2}$ - macierz wielkości dostępnych zasobów pamięci w komputerach;
E - łączne zużycie energii przez komputery gridu;
I - liczba węzłów;
$\boldsymbol{A}=\{\alpha_1,...,\alpha_v,...,\alpha_V\}$ - zbiór modułów programistycznych gridu;
S - moduł dystrybucyjny w systemie *Comcute PG*;
$\boldsymbol{T}=[t_{vj}]_{V\text{x}J}$ - macierz szacowanych czasów realizacji zadań na komputerach;
V - liczba modułów programistycznych;
W - moduł zarządzający dystrybucją pakietów danych w *Comcute PG*;
x - konfiguracja gridu;
$\boldsymbol{X}$ - zbiór możliwych konfiguracji;
$X^{\alpha}=[X_1^{\alpha},...,X_v^{\alpha},...,X_V^{\alpha}]^T$ - wektor przydziału modułów programistycznych;
$X^{\beta}=[X_1^{\beta},...,X_i^{\beta},...,X_I^{\beta}]^T$ - wektor przydziału komputerów do węzłów;
$Z_{\max}$ - obciążenie newralgicznego węzła;
Z_{suma} - łączne obciążenie wszystkich hostów;
$\hat{Z}_{max}$ - obciążenie newralgicznego hosta pod względem przetwarzania danych;
$\tilde{Z}_{max}$ - obciążenia newralgicznego hosta pod względem komunikacyjnym;
$\Delta_{\max}$ - ważone obciążenie newralgicznego komputera;
$\varepsilon=[\varepsilon_1,...,\varepsilon_j,...,\varepsilon_J]$ - wektor poboru mocy elektrycznej przez komputery;
$\varepsilon_{\max}$ - limit na łączne zużycie energii przez komputery gridu;
Θ - łączna moc obliczeniowa gridu;
ϑ_{min} - minimum wydajności w sensie wybranego benchmarku;
κ_{ir} - rezerwa *r*-tego rodzaju pamięci w komputerze usytuowanym w *i*-tym węźle;
μ_k - *k*-ty selektor wymagań jakościowych gridu, $k=\overline{1,5}$;
Ξ - koszt zakupu komputerów;
$\xi=[\xi_1,...,\xi_j,...,\xi_J]^T$ - wektor kosztów zakupu komputerów;
$\xi_{\max}$ - limit finansowy na zakup komputerów;
$\boldsymbol{B}=\{\beta_1,...,\beta_j,...,\beta_J\}$ - zbiór możliwych rodzajów komputerów;
$\boldsymbol{\tau}=[\tau_{vuil}]_{V\times V\times I\times I}$ - macierz szacowanych czasów komunikacji danych między parami modułów;
υ - stopień rozproszenia modułów programistycznych;
$\upsilon_{\min}$ - minimalny stopień rozproszenia modułów programistycznych;

Wprowadzenie

Współczesne aplikacje programistyczne cechują się wysokimi wymaganiami odnośnie mocy obliczeniowej komputerów, co powoduje, że prowadzone są intensywne badania nad rozwojem nie tylko superkomputerów, ale także wydajnych rozproszonych systemów komputerowych, w tym systemów o architekturze typu grid. Wydajność obliczeniowa najszybszego gridu *Folding@home* szacowana jest na 39,9 [PFLOPS] w oparciu o dane z grudnia 2014 roku [385].

Natomiast wydajność systemu BOINC wynosi 5,6 [PFLOPS] [379], przy czym moce obliczeniowe osiągnięte dla najważniejszych projektów realizowanych z wykorzystaniem tego rodzaju oprogramowania kształtują się następująco: *SETI@Home* - 681 [TFLOPS] [398], *Einstein@Home* - 492 [TFLOPS] [384] oraz *MilkyWay@Home* - 471 [TFLOPS] [390]. Natomiast moc obliczeniowa gridu GIMPS wynosi 173 [TFLOPS], co umożliwiło odkrycie 48-ej liczby pierwszej *Mersenne'a* w 2013 roku [388].

Dla porównania wydajność najszybszego superkomputera *Tianhe-2* to 33,86 [PFLOPS] osiągnięta w listopadzie 2014 roku [405]. W Polsce największym gridem jest ogólnokrajowy *Pl-Grid* [397]. Natomiast autor niniejszej dysertacji uczestniczył w projektowaniu eksperymentalnego systemu gridowego *Comcute PG*, w którym zrealizowano paradygmat wolontariatu obliczeniowego w oparciu o oprogramowanie wytworzone na Politechnice Gdańskiej [381].

Ważnym powodem do podjęcia badań w ramach powyższej problematyki są także bariery technologiczne związane z produkcją coraz to bardziej wydajnych mikroprocesorów. W tej sytuacji sposób zrównoleglania obliczeń w systemach gridowych jest alternatywą, która umożliwia efektywne zwiększanie wydajności obliczeń.

Jednakże architektura gridowa stawia nowe wyzwania dotyczące algorytmów w zakresie optymalizacji wykorzystania zasobów. W szczególności podczas pracy nad systemem utrzymania wielkiej mocy obliczeniowej w warunkach kryzysowych *Comcute PG* napotkano problem polegający na mało efektywnym równoważeniu obciążeń w systemach tej klasy w oparciu o dostępne oprogramowanie systemowe.

Badania nad technikami równoważenia obciążeń wykazały, że zasadne jest opracowanie kompleksowej optymalizacji równoważenia obciążeń komputerów

dystrybuujących zadania i dane, gdyż oprogramowanie na poziomie systemu nazw domenowych DNS oraz rozproszonych systemów operacyjnych nie zawiera zaawansowanych metod równoważenia obciążeń. Powyższe przesłanki pozwoliły na postawienie następującego celu pracy.

Celem naukowym jest *opracowanie algorytmów optymalizacji wykorzystania zasobów ze szczególnym uwzględnieniem równoważenia obciążeń w systemach gridowych przy uwzględnieniu wymagań związanych ze specyfiką przetwarzanych zadań.*

Konieczność równoważenia obciążeń w oparciu o różnorodne zagadnienia optymalizacji wymusza zastosowanie metaheurystyk opierających się na metodach sztucznej inteligencji, w tym szczególnie interesujących algorytmów przeszukiwania harmonicznego (ang. *harmony search*) [61]. Algorytmy harmoniczne od trzech lat stosowane są z powodzeniem do wyznaczania rozwiązań *Pareto*-optymalnych [295, 321], przy czym nie zastosowano ich jeszcze do równoważenia obciążeń w gridach. Dlatego też postawiono cel pomocniczy, jak niżej.

Celem pomocniczym jest *rozwój teorii i zastosowań algorytmów przeszukiwania harmonicznego do równoważenia obciążeń w systemach gridowych*.

Równoważenie obciążeń w gridzie może być realizowane za pomocą minimalizacji obciążenia newralgicznego komputera. Im mniejsze obciążenie newralgicznego komputera, tym większa równowaga w sensie obciążenia wszystkich węzłów. Jako kryteria rozpatruje się dwa rodzaje obciążeń: obciążenie procesorów i obciążenie komunikacyjne. Natomiast wykorzystanie pamięci RAM i pamięci dyskowej uwzględnia się w ograniczeniach rozważanych problemów optymalizacji. Koszt zakupu komputerów może być zarówno kryterium, jak również ograniczeniem. Podobnie, łączną wydajność gridu oraz inne wymagania projektowe można rozpatrywać jako kryterium optymalizacji lub ograniczenie. Z tego powodu mamy do czynienia z zagadnieniami polioptymalizacji, co wpływa na sformułowanie następującego problemu badawczego.

Problem badawczy polega na *opracowaniu modelu, sformułowaniu adekwatnego zagadnienia optymalizacji wielokryterialnej i skonstruowaniu metody wyznaczania konfiguracji kompromisowych w oparciu o algorytm przeszukiwania harmonicznego*.

Na podstawie tak określonego celu rozprawy i sformułowanego problemu badawczego dowodzi się prawdziwości następującej hipotezy. *Algorytm harmoniczny umożliwia wyznaczanie rozwiązań suboptymalnych w sensie Pareto w sformułowanym*

zagadnieniu polioptymalizacji w odniesieniu do równoważenia obciążeń w wybranych systemach klasy grid.

Wynikiem przeprowadzonych badań są modele, sformułowane problemy optymalizacji oraz algorytmy harmoniczne optymalizacji konfiguracji w systemach gridowych ze szczególnym uwzględnieniem przetwarzania realizowanego w oparciu o paradygmat wolontariatu obliczeniowego. Algorytmy zaimplementowane w języku *Java* udostępniono w pakiecie ADAIORG'14 (akronim od *Aplikacja Do Analizy I Optymalizacji Rekonfiguracji Gridu*).

W czterech rozdziałach zawarto problematykę pracy. W rozdziale pierwszym scharakteryzowano wybrane gridy. Omówiono oprogramowanie klasy *Globus Toolkit*, grid BOINC, grid GIMPS, *Polską Infrastrukturę Gridową PL-Grid*, a także grid i oprogramowanie *Comcute PG*. Po każdym rozdziale zamieszczono wnioski i uwagi.

Wybrane modele konfiguracji systemów gridowych opisano w rozdziale drugim. Przedstawiono podstawowe założenia modelu wyznaczania konfiguracji pod kątem równoważenia obciążeń. Zdefiniowano zbiór konfiguracji dopuszczalnych, a także kryteria oceny ich jakości. Wyznaczając konfigurację gridu, określa się rozdział kluczowych zasobów gridu: rodzaje komputerów, obciążenia procesorów, zajętość pamięci oraz lokalizację modułów warstwy pośredniczącej.

W rozdziale trzecim sformułowano wybrane problemy optymalizacji konfiguracji podstawowych zasobów gridu. Omówiono równoważenie obciążeń gridu, a także sformułowano zagadnienia minimalizacji ważonego obciążenia procesorów oraz obciążenia komunikacyjnego. Problemy wyznaczania konfiguracji optymalnych w sensie *Pareto* wraz z zadaniem wyznaczania rozwiązań kompromisowych stanowią oryginalne podejście do formułowania zagadnień równoważenia obciążeń gridu.

W rozdziale czwartym opisano algorytmy harmoniczne do poszukiwania konfiguracji gridu dla zagadnień z jednym kryterium, a także konfiguracji *Pareto*-optymalnych oraz kompromisowych. Rozdział zaczyna się od krótkiego omówienia metod optymalizacji, po którym scharakteryzowano algorytmy harmoniczne. Ponadto dokonano zwięzłego przeglądu metod optymalizacji wektorowej. Na tym tle zaprezentowano oryginalne wielokryterialne algorytmy harmoniczne w zastosowaniu do polioptymalizacji wybranych zasobów gridu.

Na zakończenie zamieszczono bibliografię, wykaz rysunków i tabel, a także dodatek, w którym opisano dodatkowe eksperymenty numeryczne. Opracowaną metodę wyznaczania konfiguracji zamierza się wdrożyć w systemie *Comcute PG*.

1. WYBRANE SYSTEMY ROZPROSZONE O ARCHITEKTURZE KLASY GRID

Nowoczesne systemy gridowe zbudowano w celu ułatwienia dostępu naukowcom do superkomputerów w oparciu o oprogramowanie klasy *Globus Toolkit* [387]. Ponadto w ostatnich latach dużą popularnością cieszyły się projekty realizowane za pomocą modelu obliczeń na zasadzie wolontariatu, który to model zaimplementowano w oprogramowaniu *BOINC* wywodzącym się z infrastruktury gridowej o tej samej nazwie [379]. W powyższym oprogramowaniu wykorzystuje się architekturę typu *klient-serwer* i jej rozszerzenia.

Zwłaszcza projekt *SETI@home* wzbudził szersze zainteresowanie internautów ze względu na to, iż odnosił się do poszukiwania cywilizacji pozaziemskich za pomocą analizy szumu radiowego z kosmosu [398]. Oprogramowanie *BOINC* z powodzeniem wykorzystywane jest w około siedemdziesięciu projektach takich jak: *Collatz Conjecture* czy *PrimeGrid* [379].

Gridem o najwyższej wydajności jest *Folding@home* nadzorowany przez *Stanford University* [385], w którym to projekcie symuluje się procesy zmian w strukturach białka w celu wynalezienia antidotum na wiele nieuleczalnych chorób. Grid działa w oparciu o oprogramowanie o tej samej nazwie.

Najnowsze doniesienia odnośnie wydajności gridów dotyczą wydajności *Bitcoin Network*, która to wydajność przekracza moc obliczeniową *Folding@home* [402]. Ze względu jednak na brak naukowej weryfikacji źródeł statystycznych w tej kwestii nie można tej klasy systemu jeszcze sklasyfikować. *Bitcoin* jest także nazwą waluty cyfrowej, za pomocą której można realizować płatności na całym świecie między internautami posiadającymi konto w gridzie. System opiera się na architekturze *P2P* (ang. *peer-to-peer*), gdyż nie zakłada się istnienia banku centralnego oraz kontroli nadzoru finansowego. Oprogramowanie *Bitcoina* rozwijane jest przez społeczność internautów [402].

Warto podkreślić, że grid jest rozproszonym systemem komputerowym, który implementuje wirtualne środowisko obliczeniowe dla pewnej organizacji zrzeszającej użytkowników potrzebujących usług informatycznych o odpowiedniej jakości [389]. Zarządzanie zasobami odbywa się w wielu domenach z wykorzystaniem uzgodnionego zestawu protokołów i powszechnie dostępnych interfejsów.

Natomiast zasoby gridowe, w tym komputery, specjalistyczne urządzenia, zbiory danych są heterogeniczne i cechują się lokalną autonomią. W gridach istotna jest optymalizacja wykorzystania zasobów oraz wykonania aplikacji. Ponadto organizacje mogą „przesuwać" obciążenia w inne obszary tego globalnego systemu [367].

1.1. Wprowadzenie do systemów rozproszonych klasy grid

Próbę usystematyzowania obszernej problematyki przetwarzania gridowego podjęto w [35]. Najbardziej znane platformy gridowe to *ARC*, *BOINC*, *DIET*, *Globus Toolkit* oraz *gLite* [393]. Warto także wspomnieć o *EMI*, *GridWay* czy *Oracle Grid Engine* [391].

Advanced Resource Connector ARC jest warstwą pośredniczącą oprogramowania opracowaną przez *NorduGrid*, która to warstwa dostarcza interfejs do zarządzania zadaniami przez użytkownika w celu ich wykonania w wybranych systemach rozproszonych [35]. *ARC* integruje systemy rozproszone w jeden grid. W szczególności *ARC* zastosowano jako warstwę nadrzędną w odniesieniu do *Globus Toolkit* w gridzie dedykowanym do wspierania obliczeń w ramach projektu *Wielki Zderzacz Hadronów LHC* (ang. *the Large Hadron Collider*) [391].

Wybrane komponenty pakietu *ARC* włączono do oprogramowania warstwy pośredniczącej *UMD* (ang. *the Unified Middleware Distribution*) w *Europejskiej Infrastrukturze Gridowej EGI* (ang. *the European Grid Infrastructure*). Ponadto oprogramowanie zastosowano w gridzie *Nordic e-Infrastructure Collaboration* [391].

Natomiast oprogramowanie *DIET* pełni rolę warstwy pośredniczącej w odniesieniu do francuskiego gridu uniwersyteckiego o nazwie *Décrypthon Grid*. W szczególności może zapewnić dostęp do chmur obliczeniowych w połączeniu z oprogramowaniem *Eucalyptus* (ang. *Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems*) oraz *Amazon Elastic Compute Cloud* [386].

Oprogramowanie warstwy pośredniczącej zastosowano przy budowie wielu gridów dla celów naukowych. *Europejska Infrastruktura Gridowa* zapewnia dostęp do zasobów komputerowych ponad 70 centrów obliczeniowych dla naukowców z 27 krajów europejskich [391]. Z kolei na niemieckim uniwersytecie w *Tübingen* skonstruowano grid *Cohesion Platform* o architekturze *peer-to-peer* do symulacji zmian w strukturach molekuł [394].

Badania nad mózgiem prowadzone są w ramach gridu *neuGRID*, a obliczenia neuronowe wspierane są przez komercyjny grid *ScottNet NCG* w Szkocji [35]. *ScottNet NCG* synchronizuje dane między bazami danych, superkomputerami oraz innymi repozytoriami danych o sztucznych sieciach neuronowych [91]. Realizowane zadania dotyczą: przetwarzania transakcji *e-commerce*, analizy zawartości dokumentów [145] oraz monitorowania sieci.

Do gridów wspierających procesy gospodarcze zaliczyć można *BREIN*, *AssessGrid* oraz *GridECON* [35]. W gridzie *BREIN* zastosowano system agentowy oraz elementy sieci semantycznej. Natomiast zagadnienia szacowania ryzyka rozwiązywane są za pomocą gridu *AssessGrid.*

Ponadto funkcjonuje szereg gridów, takich jak *A-Ware, GISELA, GridCOMP, Legion, NextGrid, OMII-Europe, Open Science Grid, OURGrid*, czy *TeraGrid* [386, 392, 393, 395, 401].

W tym miejscu warto podkreślić różnicę między klasycznymi obliczeniami gridowymi a obliczeniami gridowymi realizowanymi przez internautów-wolontariuszy. *Grid computing* w ujęciu tradycyjnym jest formą obliczeń rozproszonych, w których organizacja (zazwyczaj stowarzyszenie firm lub grupa uniwersytetów) wykorzystuje własne komputery (stacjonarne lub węzły w klastrach) do realizacji własnych i zazwyczaj długotrwałych zadań obliczeniowych [379].

Klasyczne obliczenia gridowe różnią się od wolontariatu obliczeniowego tym, że komputery zwracają wiarygodne wyniki. Zatem na ogół nie wymaga się replikacji danych wejściowych i weryfikacji wyników otrzymanych z superkomputerów lub stacji roboczych. Ponadto nie ma potrzeby synchronizowania obliczeń w zależności od stanu aktywności komputera wolontariusza, na którym obliczenia prowadzone mogą być zazwyczaj wraz z pojawieniem się wygaszacza ekranu. Klasyczne obliczenia gridowe są „niewidoczne" dla użytkowników komputera. Kolejną różnicą jest to, że instalacja klienta programistycznego jest zautomatyzowana [389].

Systemy gridowe zbudowane są z węzłów przetwarzających dane połączone za pomocą sieci komputerowej [19, 31]. Każdy węzeł składa się z jednostki centralnej, pamięci operacyjnej oraz systemu wejścia/wyjścia danych [40]. Zasobami mogą być także pamięci dyskowe, w tym macierze dyskowe oraz specyficzne urządzenia wejścia/wyjścia, w tym drukarki 3D.

Do głównych cech, które charakteryzują grid należą otwartość, współbieżność, skalowalność, współdzielenie zasobów, przezroczystość, a także odporność na błędy

i uszkodzenia [48, 101]. Rozszerzenie gridu o dodatkowe funkcjonalności jest możliwe za pomocą modyfikacji oprogramowania lub sprzętu [54, 73]. Ponadto grid powinien cechować się stabilną pracą mimo awarii węzłów [168]. Pod pojęciem *zasób* rozumiemy nie tylko procesor, pamięć operacyjną, system wejścia/wyjścia, pamięć dyskową, ale także dowolną usługę czy funkcjonalność udostępnioną przez grid [157].

Grid z punktu widzenia użytkownika można porównać do wirtualnego komputera składającego się z dużej liczby systemów współdzielących różnego rodzaju zasoby [260]. Oprogramowanie warstwy pośredniczącej wspiera komunikację za pomocą *serwerów webowych*, *serwerów aplikacji*, a także protokołu *SOAP* (ang. *Simple Object Access Protocol*) z wykorzystaniem *XML* (ang. *Extensible Markup Language*) [298, 394]. Oprogramowanie tej klasy zapewnia przenośność i pozwala na współdziałanie komputerów pracującymi pod różnymi systemami operacyjnymi [312]. W gridzie stosuje się wielozadaniowe protokoły uwierzytelniania i autoryzacji, a także dostępu do zasobów [316]. Grid zapewnia możliwość dostępu do zasobów z różnym poziomem jakości usług, np. gwarantowany dostęp do usługi z określonymi parametrami [329].

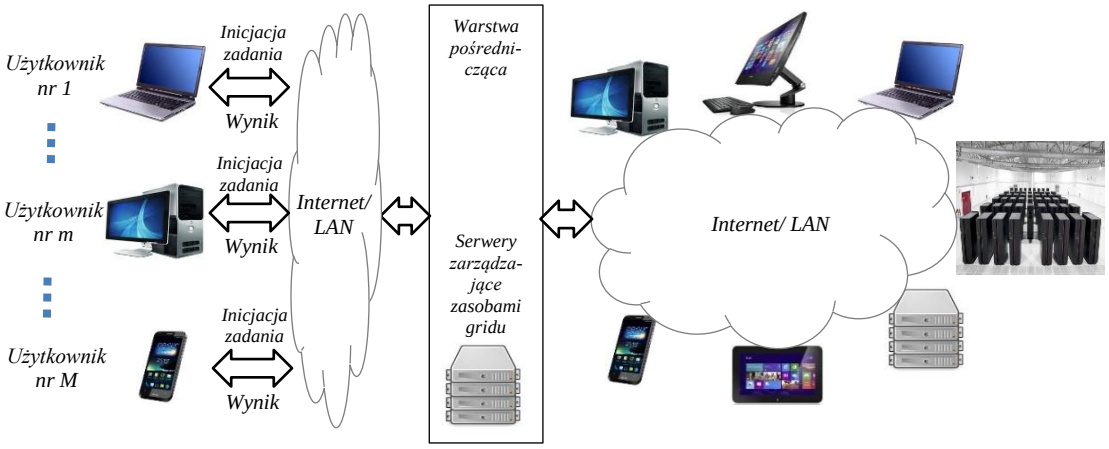
Systemy przetwarzania gridowego są kombinacją zdecentralizowanych architektur zarządzania zasobami systemu oraz wielowarstwowej architektury implementującej wybrane usługi [332, 340]. Moc obliczeniowa wszystkich komputerów może zostać rozdzielona między aplikacje użytkowników w najbardziej efektywny sposób [333, 337].

Należy zwrócić uwagę na to, że w systemach o klasycznej architekturze typu klient-serwer czas działania aplikacji zależy od ukompletowania danych z lokalnej bazy danych. Ze względu na czas replikacji danych oraz utrzymania ich spójności, pojawiają się dodatkowe opóźnienia. Kolejną wadą jest to, że niektóre zasoby zazwyczaj są niewykorzystane [350].

Rysunek 1 przedstawia infrastrukturę gridu, w którym warstwa pośrednicząca uruchomiona jest na dedykowanych serwerach zarządzających zasobami gridu. W szczególności warstwa ta rozdziela zadania między serwery obliczeniowe [367]. Warto podkreślić, że nie wszystkie aplikacje można „przenieść" z komputerów klasy PC do gridu bez większych modyfikacji. W niektórych z nich wymagane są obszerne zmiany [364].

W gridach istnieje system równoważenia obciążeń, który kolejkuje i rozdziela przychodzące zadania z komputerów użytkowników w celu najlepszego wykorzystania mocy obliczeniowej gridu [378]. Na rysunku 1 rolę maszyn obliczeniowych pełnią

superkomputery, stacje robocze, komputery klasy PC, tablety, a nawet smartfony. Dodatkowe zasoby mogą być dodane do gridu w ramach wirtualnej organizacji czy też wolontariatu – społeczności internetowej. Takie rozwiązanie ma dodatkowe zalety względem systemu standardowego, ponieważ zapewnia prawie bezpłatną moc obliczeniową z punktu widzenia użytkownika zlecającego zadania.

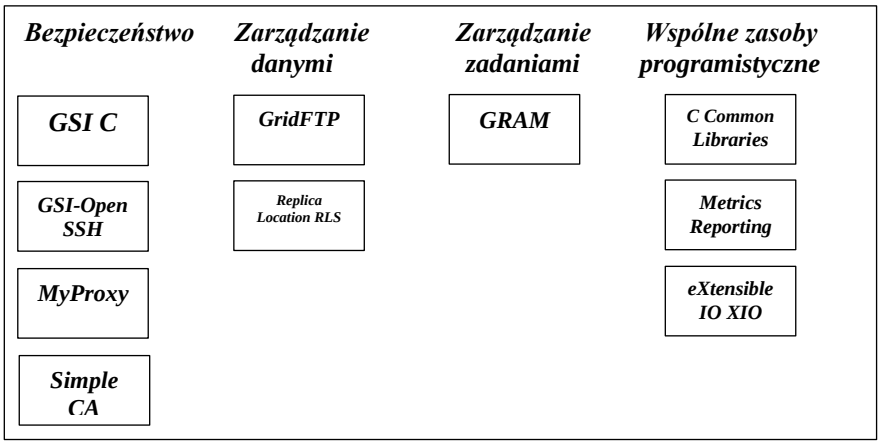


Rys. 1. Infrastruktura gridu, w którym obliczenia realizowane są przez superkomputery, stacje robocze, komputery klasy PC, tablety i smarfony
Źródło: opracowanie własne.

Warto zwrócić uwagę na fakt, że w korporacjach pracownicy eksploatują komputery jedynie przez kilka godzin na dobę [357]. Komputery te mogą być zatem podłączone do gridów obliczeniowych i służyć do rozwiązywania interesujących zagadnień naukowych lub też mogą być wykorzystane do wspomagania sytuacji kryzysowych, tak jak po awarii elektrowni atomowej w *Fukushimie* [35].

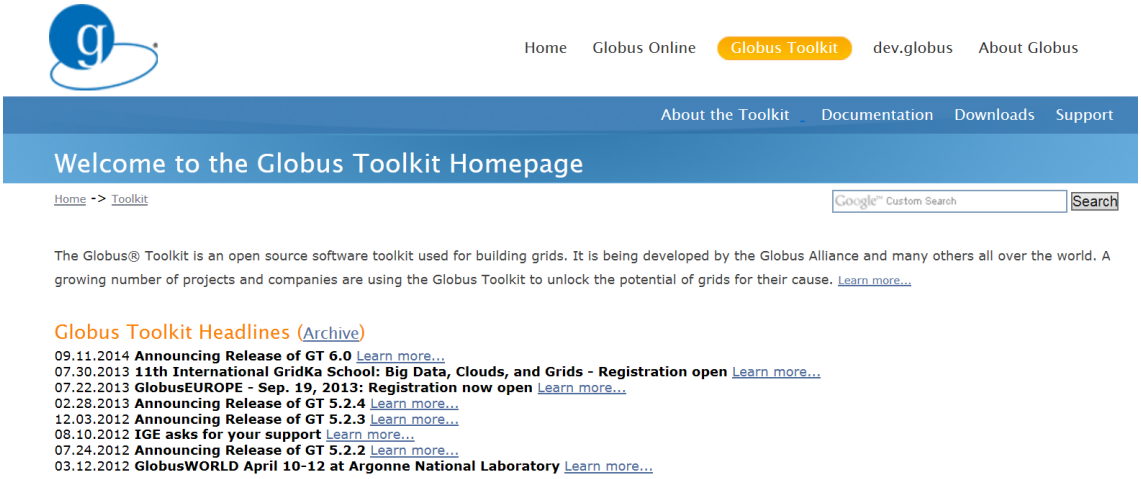
1.2. Oprogramowanie *Globus Toolkit*

Warstwa pośrednicząca gridu jest kluczowym oprogramowaniem, który umożliwia łączenie heterogenicznych zasobów i specyficzne aplikacje użytkowników w ramach wirtualnych organizacji. Oprogramowanie *Globus Toolkit* GT opracowane przez *Globus Alliance* jest najbardziej reprezentatywnym oprogramowaniem warstwy pośredniczącej gridu [387]. Pakiet wspomaga: zarządzanie zasobami infrastruktury, komunikację, bezpieczeństwo, wykrywanie błędów, a także zapewnia przenośność. Moduły *Globus Toolkit* mogą być stosowane w różnych zestawach (rys. 2).



Rys. 2. Moduły programistyczne *Globus Toolkit* w wersji 6 [387]

Współpracę między organizacjami w systemach gridowych utrudnia niekompatybilność zasobów takich jak: bazy danych, komputery czy sieci komputerowe. *Globus Toolkit* umożliwia znaczącą redukcję powyższych przeszkód (rys. 3). Ponadto usługi, interfejsy i protokoły warstwy pośredniczącej umożliwiają użytkownikom zdalny dostęp do zasobów, zachowując kontrolę nad korzystaniem z lokalnych zasobów [387].



Rys. 3. Strona główna projektu *Globus Toolkit* [387]

W połowie lat dziewięćdziesiątych ubiegłego wieku *Rick Stevens* z *Argonne National Laboratory* oraz *Tom Defanti* z *University of Illinois* w Chicago, zaproponowali połączenie jedenastu superkomputerów dedykowanych do celów badawczych, aby utworzyć sieć *I-WAY* w USA. W szczególności opracowano protokoły, które pozwoliły użytkownikom na uruchamianie aplikacji bezpośrednio na superkomputerach [387].

Udany eksperyment doprowadził do powstania pierwszej wersji *Globus Toolkit* sfinansowanej przez *Defense Advanced Research Projects Agency* DARPA w 1997 roku. Oprogramowanie wdrożono w 80 gridach na świecie. W szczególności skonstruowano system gridowy dla U.S. *Department of Energy* DOE. Natomiast *National Science Foundation* NSF sfinansowało zbudowanie *National Technology Grid* dla celów naukowych, a NASA - *Information Power Grid* [387].

Fizycy kwantowi projektujący *Wielki Zderzacz Hadronów* w CERN wykorzystują *European Data Grid*. W USA powstały: *Grid Physics Network* (GriPhyN) i *Particle Physics Data Grid*. Inne znaczące projekty e-nauki oparte na *Globus Toolkit* to: *Network for Earthquake Engineering and Simulation* (NEES), *FusionGrid* oraz *Earth System Grid* (ESG) [387].

Globus Toolkit zastosowano w gridach komercyjnych. Konsorcja takie jak: *Avaki, DataSynapse, Entropia, Fujitsu, Hewlett-Packard, IBM, NEC, Oracle, Platform, Sun* i *United Devices* prowadziły projekty w zakresie systemów gridowych oparte na GT [288].

Globus Toolkit opiera się na otwartych standardach usług gridowych. W 2002 r. projekt został uznany za najbardziej obiecującą technologię. Ponadto oprogramowanie zaliczono do najlepszych dziesięciu innowacji w 2003 r. przez magazyn *InfoWorld*. Prestiżowe *MIT Technology Review* zaliczyło *Globus Toolkit* do dziesięciu technologii, które zmienią świat. *Globus Toolkit* również zdobył nagrodę *Federal Laboratory Consortium* w dowód uznania za jego zastosowania dla celów przemysłowych [387].

O ile wersja 1 była prototypem badawczym, to już wersja 2 była powszechnie stosowana, mimo że nie opierała się na usługach sieciowych (ang. *web services*). Usługi te wykorzystano dopiero w wersji 3, przy czym nie zostały one powszechnie zaakceptowane ze względu na sposób ich implementacji i brak odporności na błędy aplikacji użytkowników. W wersji 4 poprawiono efektywność niektórych usług. Natomiast w wersji 5 zaniechano wykorzystania usług internetowych i powrócono do koncepcji z wersji 2 bez wykorzystania usług sieciowych [387].

W gridzie z oprogramowaniem GT dane przesyłane są między punktami końcowymi. *Punktem końcowym* jest adres logiczny w serwerze *GridFTP*, analogicznie do nazwy domeny dla serwera WWW. Ponadto klient *Globus Connect Personal* umożliwia komunikację aplikacji użytkowników z innymi serwerami *GridFTP*. Oprogramowanie *Globus Connect Personal Edition* na komputerze jest także punktem końcowym, który można wykorzystać do wymiany danych z komputerem. Natomiast

Globus Connect Server to pakiet linuksowy, który uruchamia serwer *GridFTP* do użytku z gridem klasy *Globus* dla wielu użytkowników. Użytkownicy z dostępem do serwera mogą wymieniać dane z tym serwerem [387].

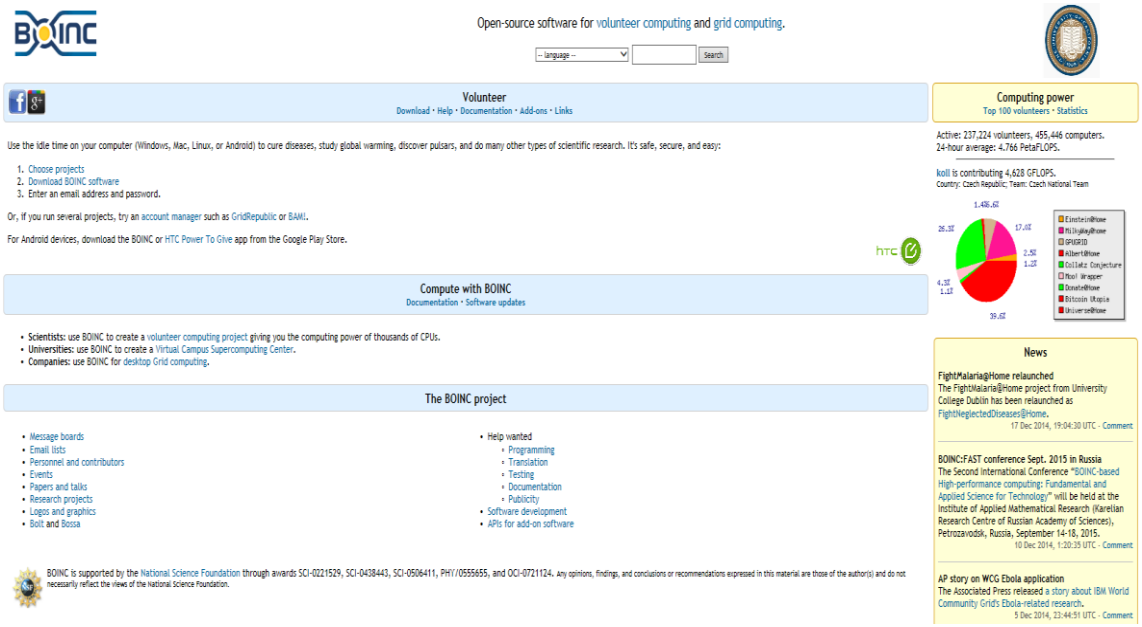
W celu podwyższenia poziomu bezpieczeństwa zastosowano *infrastrukturę bezpieczeństwa gridu* (ang. *Grid Security Infrastructure* GSI), która dostarcza bezpiecznego środowiska wokół zasobów sieciowych przy użyciu kryptografii z kluczem publicznym. *SimpleCA* implementuje centrum certyfikacji, które może wydawać certyfikaty *X.509* dla użytkowników i usług *Globus Toolkit* [387].

Natomiast do zarządzania zasobami wykorzystuje się oprogramowanie GRAM (ang. *Globus/Grid Resource Allocation Management*). GRAM przydziela zadania do zasobów, uruchamia je i monitoruje aż do ich zakończenia. GRAM współpracuje z lokalnym zarządcami zasobów (ang. *Local Resource Manager* LRM), takimi jak *Condor, Portable Batch System, Oracle GridEngine*. LRM umożliwia dostęp do lokalnych zasobów klastrów lub komputerów równoległych. GRAM współpracuje z LRM za pomocą modułu-adaptera napisanego w języku *Perl*.

Usługa GGS (ang. *the globus-gatekeeper service*) identyfikuje klienta użytkownika i uruchamia *menedżera zadań* (ang. *Job Manager* JM). JM obsługuje żądania zadań i koordynuje transfer plików. Każdemu użytkownikowi i każdemu LRM odpowiada instancja JM. Ponadto każdemu zadaniu przyporządkowani jest instancja JM. Procedura kolejkowania zdarzeń GSEG (ang. *the globus-scheduler-event-generator*) przekształca paczki lokalnych danych w format niezależny od specyficznych systemów. Opcjonalnie jednemu LRM może odpowiadać instancja GSEG [387].

1.3. Grid i oprogramowanie BOINC

Berkeley Open Infrastructure for Network Computing BOINC to infrastruktura gridowa realizująca paradygmat wolontariatu obliczeniowego, który powstał na potrzeby projektu naukowego *SETI@home* [379]. BOINC jest rozwijany na *Uniwersytecie Kalifornijskim w Berkeley* przez zespół *Davida Andersona*. Projekt wspierany jest finansowo przez amerykańską agencję rządową *National Science Foundation*. Ponadto oprogramowanie BOINC wykorzystywane jest w wielu projektach badawczych (rys. 4).



Rys. 4. Strona webowa projektu *BOINC* [379]

BOINC udostępnia niekomercyjne oprogramowanie pośredniczące na licencji *GNU LGPL* [379]. Warto podkreślić, że w Polsce działa *Drużyna BOINC@Poland*, która aktywnie wspiera projekt (rys. 5) [380].

Rys. 5. Strona grupy wolontariuszy *BOINC@Poland* [380]

Model wolontariatu obliczeniowego polega na wykorzystaniu wolnych mocy obliczeniowych gridu komputerów wolontariuszy lub gridu komputerów organizacji. Wykorzystuje się nieobciążone komputery w nocy, podczas przerw w pracy, a także podczas pracy z urządzeniami zewnętrznymi. Oprócz procesorów (ang. *CPU-scavenging*) istotnymi zasobami są: pamięć RAM, pamięć dyskową oraz przepustowość sieci [42].

Jeżeli na komputerze otrzymującym dane do przetwarzania nie została jeszcze zainstalowana aplikacja mająca je przetwarzać, jest ona również przesyłana do uczestnika projektu. W ramach jednego projektu może funkcjonować wiele aplikacji, a wysyłane dane mogą być przeznaczone dla wybranej z nich [379].

Jeśli na komputerze wolontariusza dostępne są zarówno dane do przetwarzania, jak i odpowiednia aplikacja, rozpoczyna się przetwarzanie danych. Czas przetwarzania paczki danych zależy od rodzaju projektu i waha się od kilkunastu sekund do kilkuset godzin. Dzięki systematycznemu zapisywaniu otrzymanych wyników na komputerze wolontariusza, obliczenia nie muszą odbywać się nieprzerwanie. Mogą być zawieszane, jeśli zachodzi potrzeba przeznaczenia mocy obliczeniowej na inne zadania lub wyłączenia komputera. Uwzględnia się także fakt, że komputery wolontariuszy mogą pracować *offline* [379].

Na komputerze wolontariusza mogą być jednocześnie zainstalowane dane i aplikacje z wielu projektów. W wybranym momencie na jednostce CPU (ang. *central processing unit*) może być przetwarzana tylko jedna porcja danych. Procesory wielordzeniowe z technologią HT (ang. *hyper-threading*) mogą przetwarzać jednocześnie dwie lub więcej porcji danych adekwatnie do liczby rdzeni lub wątków [35].

Wyjątkiem był projekt *DepSpid*, w którym nie wykorzystywano mocy CPU, lecz mierzono obciążenia połączeń internetowych. Jednostki tego projektu mogły być przetwarzane równolegle z jednostkami pozostałych projektów, ale ich liczba nie zależała od rodzaju czy liczby procesorów – przydzielano jedną jednostkę na jedno połączenie [379].

Jeżeli komputer wolontariusza umożliwia wykonywanie zadań dla kilku projektów, to czas procesora jest przydzielany aplikacjom w cyklu zgodnie z ustalonymi przez użytkownika przydziałami dla poszczególnych projektów [379]. Po przetworzeniu porcji danych wyniki obliczeń przesyłane są do serwera projektu.

W większości projektów kopie paczek danych są rozsyłane do kilku użytkowników, co umożliwia porównanie wyników w celu weryfikacji ewentualnych błędów.

Komputer użytkownika otrzymuje pewną liczbę *punktów kredytowych* (ang. *credits*) zależną od wykorzystanego czasu i mocy obliczeniowej procesora [379]. Punkty kredytowe umożliwiają nagradzanie wolontariuszy. Sposób wyliczenia liczby punktów kredytowych zależy od projektu, gdyż w niektórych projektach przydziela się różną liczbę punktów za takie same zużycie zasobów. Punkty kredytowe pozwalają na wzajemne współzawodnictwo uczestników projektów, którzy mogą organizować się w zespoły [379, 380].

System *BOINC*, oprócz platformy sprzętowej, składa się z programów typu: klient, serwer, a także z bazy danych. Klient *BOINC*-a (ang. *BOINC Daemon*) wspomaga komunikację i zarządzanie aplikacjami użytkownika. Klient ściąga i uruchamia na komputerze wolontariusza aplikację projektu, korzystając z cyfrowego podpisywania aplikacji, aby nie dopuścić do zainstalowania fałszywej aplikacji w miejsce oryginalnej. Program rezyduje w pamięci RAM komputera internauty, w przeciwieństwie do programu o nazwie *Menedżer*, który umożliwia konfigurowanie klienta. Aplikacja kliencka decyduje, jakie zadanie i przez jaki czas będzie liczone. Klient również komunikuje się z serwerami projektów, z których pochodzą zadania i dane.

Po stronie serwera wykonuje się *Scheduler* – serwer harmonogramowania realizacji zadań, który przydziela paczki danych do obliczeń między węzły obliczeniowe, a także odbiera uzyskane od nich wyniki. *Scheduler* zarządza zasobami komputerów internautów (moc *CPU*, pojemność pamięci operacyjnej) podczas dystrybucji zadań i danych. *Scheduler* uwzględnia średni czas w ciągu doby, jaki komputery te mogą przeznaczyć na pracę w gridzie. Dzięki temu mniej wydajny komputer nie jest zbyt intensywnie obciążony, a mocniejsza maszyna - lepiej wykorzystana.

Komputer internauty pobiera aplikację i dane wejściowe z *serwera danych* projektu. Aplikacje rezydujące na komputerach użytkowników wytwarzają pliki wyjściowe z wynikami, które są wysyłane na *serwer danych*. Komputer internauty po wysłaniu wyników do *Schedulera* otrzymuje od niego kolejne paczki danych.

W BOINC-u stosuje się redundancję, aby zapobiec błędom powstałym na komputerze użytkownika. Dane wejściowe dzielone są na pakiety. Dla nadmiarowych kopii pakietów danych, weryfikuje się zgodność otrzymanych wyników. W ten sposób

można wybrać wynik wzorcowy dla danego pakietu i dodać go do bazy danych za pomocą aplikacji *Asymilator*. Program ten może zażądać powtórzenia obliczeń dla paczki danych.

Istotną funkcjonalnością jest stosowanie *punktu przywracania stanu*. Moduł programistyczny *CheckPoint* wykorzystuje zapisywanie przetworzonego fragmentu danych wejściowych i wyników na dysku, dzięki czemu obliczenia mogą być kontynuowane po przerwie w obliczeniach spowodowanej awarią.

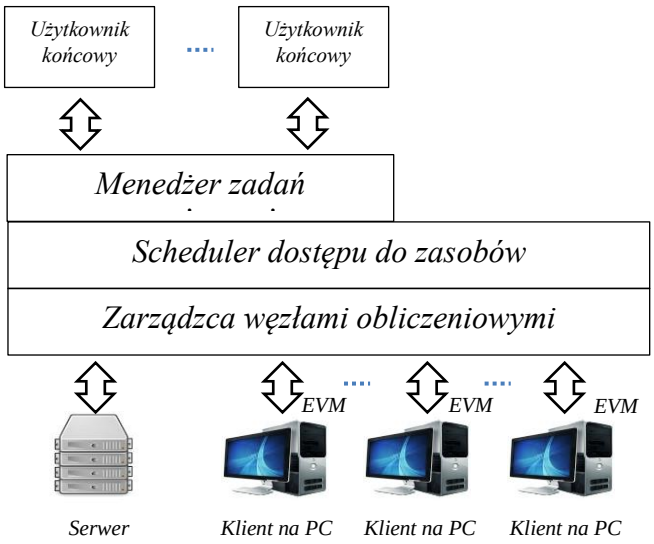
Program *Tranzycjoner* zajmuje się zmianami stanów zadań oraz pakietów danych. Śledzi pakiety będące w trakcie obliczeń, po czym zmienia ich statusy w odpowiednim momencie. Sprawdza, czy pakiet danych jest gotowy do wysłania, czy został wysłany, czy jest poprawny lub czy można go już usunąć. *Tranzycjoner* ma również możliwość generowania nowych pakietów, nadawania im statusu jako nieodwracalnie błędnej, bądź żądania weryfikacji lub akceptacji pakietu.

Poziom nadmiarowości obliczeń rośnie wraz z liczbą klientów. Ponadto ilość danych nadmiarowych zazwyczaj zwiększa się w ciągu „życia" projektu. Do przechowywania informacji o pakietach danych wejściowych, wynikach i użytkownikach wykorzystywana jest relacyjna baza danych. Serwer przechowujący dane rozprowadza je wśród użytkowników [379].

1.4. Grid GIMPS

Grid GIMPS (ang. *Great Internet Mersenne Prime Search*) to system obliczeń rozproszonych z udziałem wolontariuszy, za pomocą którego poszukuje się kolejnych liczb pierwszych *Mersenne'a* [388]. W styczniu 2013 roku za pomocą systemu wyznaczono czterdziestą ósmą liczbę pierwszą *Mersenne'a* o wartości $2^{57\,885\,161}$-1, którą można zapisać za pomocą 17 425 170 cyfr [388].

GIMPS umożliwia przyłączanie i odłączanie komputerów wolontariuszy. Program typu klient musi być zainstalowany przez użytkownika. W gridzie początkowo wykorzystano komercyjny serwer *Entropia* do konstruowania gridów obliczeniowych z komputerów klasy desktop. Architekturę systemu *Entropia* przedstawiono na rysunku 6 [58]. *Nadrzędny serwer zarządzający* sprawuje kontrolę nad węzłami obliczeniowymi, a także nadzoruje przydział zasobów do zadań.



Rys. 6. Architektura systemu obliczeń rozproszonych *Entropia* [58]

Użytkownik systemu zleca obliczenia, korzystając z *Menedżera zadań*, który dzieli obliczenia na wiele niezależnych podzadań i przekazuje je do *Schedulera dostępu do zasobów*. Informacje o dostępnych zasobach są okresowo wysyłane do *Menedżera zadań*, który przekazuje je do *Schedulera*. Natomiast *Scheduler* wysyła podzadania do klientów systemu na komputerach typu desktop w taki sposób, aby dopasować zasoby do wymagań podzadań. Wyniki wykonania zadań na klientach są odsyłane do *Menedżera zadań*, który łączy je i przekazuje do użytkownika końcowego.

Na komputerach-klientach zainstalowano komponent *Entropia Virtual Machine* EVM, który zarządza wykonaniem podzadań oraz dostępem do zasobów komputera, np. do systemu plików, pamięci operacyjnej lub CPU.

Pierwszość 48-ej liczby pierwszej *Mersenne'a* była testowana przez 39 dni na komputerze klasy PC. Natomiast obliczenia testu *Lucasa* na GPU klasy NVIDIA trwały 3,6 dni, a na CPU *Intel i*7 – 4,5 dni [388].

1.5. Polska Infrastruktura Gridowa PL-Grid

W projekcie PL-Grid skorzystano z doświadczeń nabytych przy projektowaniu *Krajowego Klastra Linuksowego* CLUSTERIX (ang. *national CLUSTER of lInuX systems*) zbudowanego w 2004 roku z przeznaczeniem do modelowania i symulacji różnorodnych procesów i zjawisk, w tym do symulacji procesów zachodzących w organizmach biologicznych.

Polska Infrastruktura Gridowa zbudowana w ramach projektu *PL-Grid* w latach 2009-2012, dostarcza polskiej społeczności naukowej infrastrukturę opartą na klastrach komputerowych (rys. 7). Infrastruktura wspiera badania naukowe poprzez integrację danych doświadczalnych i wyników zaawansowanych symulacji komputerowych prowadzonych przez geograficznie rozproszone zespoły. Infrastruktura *PL-Grid* umożliwia prowadzenie badań naukowych w oparciu o symulacje i obliczenia dużej mocy, a także zapewnia wygodny dostęp do rozproszonych zasobów komputerowych [397].

Rys. 7. Strona główna projektu *PL-Grid* [397]

Polska Infrastruktura Gridowa jest częścią infrastruktury budowanej w ramach *European Grid Initiative* EGI, której celem jest integracja narodowych infrastruktur gridowych w trwałą infrastrukturę [397]. Zasoby obliczeniowe *PL-Grid* cechują się łączną mocą obliczeniową 230 [TFLOPS], pamięcią dyskową o pojemności 3,6 [PB], pamięcią operacyjną o pojemności 112 [TB] oraz liczbą rdzeni obliczeniowych wynoszącą 41 160. Ponadto trzykrotne zwiększenie tych zasobów planowane jest w realizowanym projekcie *PLGrid Plus* [397].

Wdrożenie infrastruktury *PL-Grid* ułatwiło efektywne wykorzystanie zasobów poprzez dostarczenie użytkownikom innowacyjnych usług gridowych i narzędzi.

W gridzie udostępnia się specjalistyczne oprogramowanie naukowe, dostępne dotychczas tylko w niektórych polskich centrach obliczeniowych dużej mocy.

Do badań biologicznych udostępniono oprogramowanie: *AutoDock/AutoGrid*, BLAST, *Clustal*, CPMD, *Gromacs* i NAMD. Do badań w zakresie chemii kwantowej można wykorzystać następujące aplikacje: ADF, *Amber*, CFOUR, *Dalton*, GAMESS, *Gaussian*, *Molpro*, MOPAC, NW*Chem*, *Siesta* i *TURBOMOLE*. Natomiast oprogramowanie ANSYS *Fluent*, *Meep* i *Open*FOAM dedykowane jest do wspierania eksperymentów fizycznych. Z kolei obliczenia numeryczne i symulacje mogą być przeprowadzone za pomocą LAMMPS, MATLAB, *Nmag*, *R* oraz *Wolfram Mathematica*. Warto podkreślić, że dostępne są także aplikacje *Blender*, *Elegant*, *ParaView*, *POV-Ray*, *Visit* oraz X*CrySDen* [397].

Polska Infrastruktura Gridowa jest zarządzana przez konsorcjum *PL-Grid*, w skład którego wchodzą: ACK CYFRONET AGH – Akademickie Centrum Komputerowe CYFRONET AGH w Krakowie, ICM UW – Interdyscyplinarne Centrum Modelowania Matematycznego i Komputerowego w Warszawie, PCSS – Instytut Chemii Bioorganicznej PAN - Poznańskie Centrum Superkomputerowo Sieciowe w Poznaniu, CI TASK – Centrum Informatyczne Trójmiejskiej Akademickiej Sieci Komputerowej w Gdańsku oraz WCSS – Wrocławskie Centrum Sieciowo-Superkomputerowe we Wrocławiu [397].

1.6. Grid i oprogramowanie *Comcute PG*

Grid *Comcute PG* to eksperymentalny system do obliczeń rozproszonych z udziałem wolontariuszy, który to system opracowano na Politechnice Gdańskiej (rys. 8) [381]. System *Comcute* to środowisko wytwarzania i uruchamiania aplikacji rozproszonych. Projektując architekturę gridu, założono dekompozycję systemu na trzy warstwy: jądro, warstwę pośredniczącą oraz węzły obliczeniowe, w których środowiskami wykonawczymi są przeglądarki internetowe [32].

Celem projektu było zbudowanie demonstratora środowiska umożliwiającego prowadzenie pewnej klasy obliczeń rozproszonych związanych ze wspieraniem sytuacji kryzysowych, przy wykorzystaniu mocy obliczeniowych dostępnych w Internecie komputerów wolontariuszy [262]. Architektura infrastruktury gridowej *Comcute*, cechuje się niskim kosztem i zużyciem energii dla wielowarstwowego systemu realizacji obliczeń wielkiej skali [263].

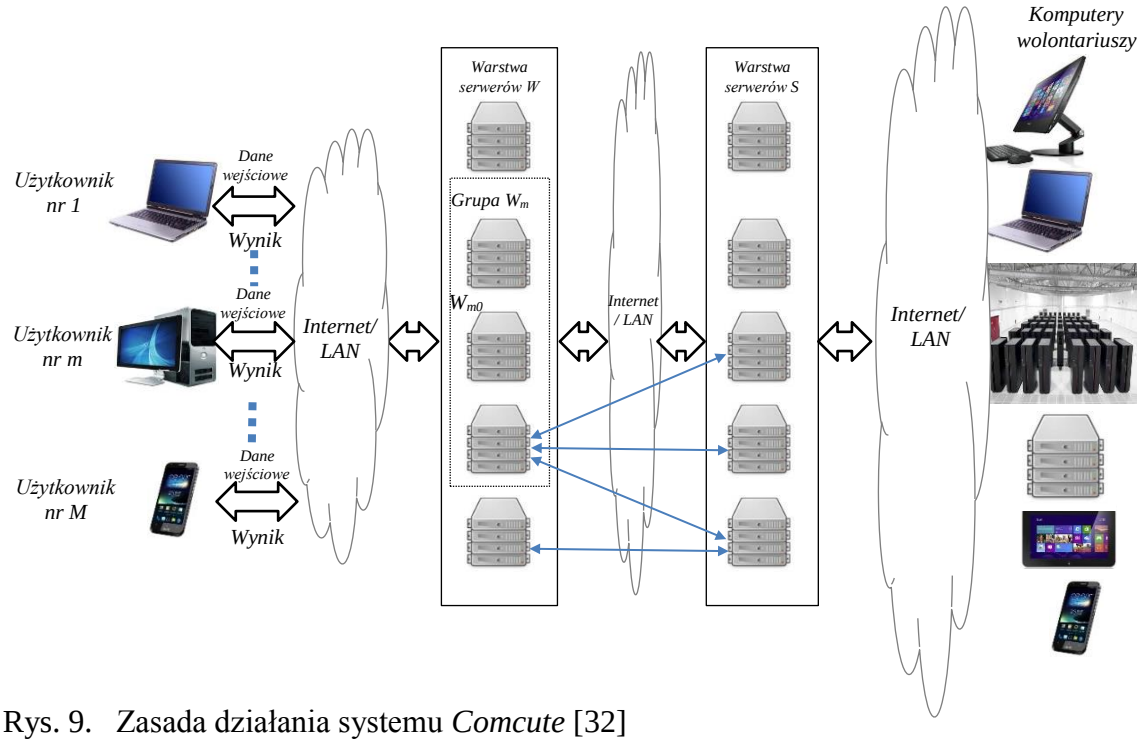
Rys. 8. Strona główna projektu *Comcute* [381]

Innowacją rozwiązania jest brak konieczności instalacji aplikacji klienckich, tak jak w wielu gridach typu *volunteer computing*. Tym samym znacząco upraszcza się sposób rekrutacji komputerów wolontariuszy [271]. Zakłada się, że w warunkach kryzysowych internauci, w tym również pracownicy administracji, powinni udostępnić moc obliczeniową komputerów do realizacji obliczeń wspierających działania antykryzysowe. Oprogramowanie serwera dystrybucyjnego odbiera sygnał o gotowości komputera internauty i przekazuje zwrotnie zadanie w formie najbardziej efektywnej do realizacji. Tablet otrzyma zatem skrypt w języku *JavaScript*, a komputer z wirtualną maszyną *Javy* – aplet *Javy* [273].

Przyjęte rozwiązania sprawiają, że użytkownik komputera osobistego nie odczuwa spadku mocy obliczeniowej w związku z obliczeniami prowadzonymi na jego maszynie, gdyż aplikacja z *Comcuta* zwiększa obciążenie procesora internauty do ok. 75% [265]. Otrzymane od nadrzędnej warstwy zadania i dane są przekazywane do komputerów, a po zrealizowaniu obliczeń zwracane do warstwy nadrzędnej (rys. 9).

Warto podkreślić, że oprogramowanie pośredniczące składa się z trzech warstw: inicjacji obliczeń *Z*, zarządzania zadaniami i danymi *W*, a także dystrybucji obliczeń *S*. Grid cechuje się skalowalnością w każdej z warstw. Skalowalność pionowa dotyczy

zmiany sprzętu, a pozioma – dodawania węzłów. System jest otwarty, przezroczysty, ma globalny zasięg i jest rozproszony geograficznie. Nietrudno jest także uzyskać interoperacyjność z innymi systemami wspomagającymi działania antykryzysowe [32].



Rys. 9. Zasada działania systemu *Comcute* [32]

Serwery typu *W zarządzają obliczeniami* i są zorganizowane w wirtualną grupę dla każdego zadania użytkownika. Natomiast każdy z serwerów *S dystrybuujących zadania i dane* jest logicznie powiązany przynajmniej z jednym *serwerem zarządzającym*. Oprogramowanie pośredniczące udostępnia zadaniu nr *m* wybrany serwer W_{m0}. Użytkownik, inicjując zadanie obliczeniowe znajdujące się w repozytorium zadań, specyfikuje dane wejściowe oraz parametry jego wykonania dotyczące wydajności i redundancji. Wykonanie zadania składa się z następujących faz: inicjacji, dzielenia danych, propagacji danych, wykonania obliczeń, zbierania wyników i scalania wyników.

Niech parametr wydajnościowy $PAR_1(m)$ specyfikujący liczebność grupy W_m wynosi *n*. Podczas inicjacji *m*-tego zadania moduł W_{m0} organizuje grupę *n* serwerów zarządzających. W_{m0} wysyła żądanie współpracy do serwerów klasy *W* i oczekuje na ich zgłoszenia, w których podawane są aktualne obciążenia. Jeśli W_{m0} otrzyma w zadanym czasie przynajmniej *n*-1 zgłoszeń, to rozsyła kod zadania obliczeniowego nr *m* do *n*-1 serwerów o najmniejszym obciążeniu. W przeciwnym wypadku, wysyła do użytkownika komunikat o niemożności wykonania zadania z parametrem PAR_1.

Użytkownik może podjąć decyzję o zaniechaniu wykonania zadania lub o wykonaniu zadania z innym parametrem PAR_1. Faza inicjacji może skończyć się utworzeniem grupy n serwerów klasy W zaangażowanych w wykonanie zadania.

Niech parametr $PAR_2(m)=r$ określa liczbę replikowanych danych m-tego zadania. W fazie dzielenia zbiór danych wejściowych D_m dla zadania nr m przekształcany jest na paczki danych $IN_{m1},\ldots,IN_{mk},\ldots,IN_{mK}$. Każda paczka danych zwielokrotniana jest r-krotnie, a następnie kopie paczek rozsyłane są do r serwerów klasy W.

W fazie propagacji danych serwery zarządzające z grupy W_m oczekują na zgłoszenia serwerów dystrybucyjnych S, które otrzymują informacje o gotowości komputerów wolontariuszy. Po otrzymaniu zgłoszenia od S, serwer z grupy W_m przesyła kod obliczeniowy zadania nr m oraz pierwszą paczkę danych.

Serwer S przechowuje paczki danych oraz kody zadań w kolejce, oczekując na zgłoszenia komputerów internautów. Internauci klikając na odpowiednie linki w serwisach *Comcuta*, zgłaszają gotowość swoich komputerów do serwerów dystrybucyjnych. W treści każdej z tego rodzaju witryn zawarta jest propozycja uczestniczenia w projekcie obliczeniowym. Internauta może skorzystać z tej opcji i udostępnić swój komputer do obliczeń.

W fazie wykonywania obliczeń w przeglądarce webowej na komputerze internauty uruchamiany jest kod programu *do pobierania paczek danych* oraz kod obliczeniowy m-tego zadania. Następnie wynik obliczeń OUT_{mk} odsyłany jest do serwera S, z którego pobrano paczkę danych wejściowych IN_{mk}.

W fazie zbierania wyników serwery dystrybucyjne przesyłają otrzymane wyniki OUT_{mk} do serwerów zarządzających z grupy W_m, z których pobrały paczki danych wejściowych IN_{mk}. Po otrzymaniu wyniku OUT_{mk} serwer W we współdziałaniu z pozostałymi serwerami z grupy W_m sprawdza, czy wyniki dotyczące tej samej paczki danych IN_{mk} są identyczne z wynikami otrzymanymi przez pozostałe serwery w grupie. Jeśli wyniki nie są identyczne, to za wiarygodne uznawane są wyniki uzyskane przez większą liczbę serwerów. Domyślnym warunkiem zakończenia obliczeń jest skompletowanie wszystkich wyników. Po zakończeniu obliczeń następuje faza *konsolidacji*. Wyniki końcowe użytkownik może pobrać z dowolnego serwera zarządzającego.

W wersji laboratoryjnej gridu *Comcute* skonfigurowano 9 węzłów obliczeniowych, macierz dyskową oraz lokalną sieć komputerową *Gigabit Ethernet*. Równoważenie na poziomie DNS to pierwszy z poziomów równoważenia obciążeń

w środowisku gridowym *Comcute*. DNS zapewnia zamianę adresów komputerów użytkowników na adresy IP urządzeń w sieci. Jeśli generowany jest sygnał o gotowości komputera wolontariusza do obliczeń, to wyznacza się adres IP serwera klasy *S* w oparciu o przyjętą strategię planowania typu *round-robin* [32]. Zachodzi przekierowywanie ruchu do właściwych serwerów na podstawie zawartości pakietów danych (wartości nagłówka, nazwy domeny lub adresu URL) [20].

Drugi sposób równoważenia obciążeń realizowany może być na poziomie systemu operacyjnego *CentOS*, który zarządza lokalnie węzłami obliczeniowymi *Comcuta* [20]. W tym celu może być wykorzystany *Linux Virtual Server* (akronim LVS), za pomocą którego dostępne jest oprogramowanie IPVS dedykowane do równoważenia obciążeń. IPVS wspomaga warstwę transportową protokołu i kieruje żądania o usługi TCP i UDP do serwerów. Z poziomu systemu *CentOS* wykorzystywany może być także moduł KTCPVS, w którym równoważenie obciążeń realizowane jest na poziomie aplikacji [20].

Trzeci sposób równoważenia obciążeń może odbywać się na poziomie serwera aplikacji *GlassFish* [20], który wspiera wykonywanie modułu klasy *S* lub *W* poprzez zarządzanie sesją i zarządzanie transakcjami danych. Serwer aplikacji wspiera współdzielenie zasobów oraz skalowalność aplikacji. Wykorzystanie serwera aplikacji niesie ze sobą wiele zalet. Architektura *cienkiego* klienta nie tylko redukuje w aplikacji klienta obciążenie związane z przetwarzaniem, ale również powoduje, iż łatwiej jest zarządzać logiką biznesową, która może być scentralizowana na jednym serwerze lub małej ich liczbie [20].

Model klient-serwer z serwerem aplikacji opiera się na architekturze wielowarstwowej, w którym warstwa pośrednicząca przyjmuje żądania cienkiego klienta. Warstwa ta zajmuje się mało wymagającymi zadaniami, przetwarza je, komunikuje się z serwerem bazy danych i zwraca klientowi wyniki w odpowiedniej formie. Rozwiązanie to zapewnia bezpieczeństwo, elastyczność, a także umożliwia równoważenie obciążenia [20].

Serwer aplikacji *GlassFish* cechuje się rozszerzonymi funkcjonalnościami w odniesieniu do serwerów WWW właśnie dzięki takim usługom jak równoważenie obciążenia. *GlassFish* rozdziela też warstwę kliencką od warstwy bazodanowej, implementując mechanizmy autoryzacji i uwierzytelniania. Zwiększenie wydajności za pomocą serwera *GlassFish* polega na redukcji ruchu sieciowego.

Skalowalność i zapewnienie zintegrowanego zarządzania to kolejna zaleta serwera *GlassFish*. Warto nadmienić, że klienty webowe komunikują się bezpośrednio za pomocą protokołu HTTP lub HTTPS, natomiast klienty EJB porozumiewają się pośrednio przy pomocy ORB (ang. *Object Request Broker*) poprzez protokoły IIOP i IIOP/SSL. Istotną zaletą serwera aplikacji jest także zapewnienie środowiska do wytworzenia, testowania, wdrażania i zarządzania aplikacjami, a także dostarczenie biblioteki klas [20].

1.7. Wnioski i uwagi

Nowoczesne systemy gridowe zbudowano w celu ułatwienia dostępu naukowcom do superkomputerów w oparciu o oprogramowanie warstwy pośredniczącej ARC oraz *Globus Toolkit*. Alternatywą są gridy realizowane za pomocą modelu obliczeń na zasadzie wolontariatu, który zaimplementowano w oprogramowaniu BOINC.

Warstwa pośrednicząca dostarcza narzędzi do zarządzania zadaniami przez użytkownika w celu ich wykonania w wybranych systemach rozproszonych. Umożliwia łączenie heterogenicznych zasobów i specyficzne aplikacje użytkowników w ramach wirtualnych organizacji. Warstwa wspomaga: zarządzanie zasobami infrastruktury, komunikację, bezpieczeństwo, wykrywanie błędów, a także zapewnia przenośność.

W wielu gridach wykorzystuje się architekturę typu *klient-serwer* i jej rozszerzenia. Konstruowane są także gridy o architekturze typu *peer-to-pe*er.

Grid computing to forma obliczeń rozproszonych, w których organizacja wykorzystuje własne komputery do realizacji długotrwałych zadań obliczeniowych. Klasyczne obliczenia gridowe różnią się od obliczeń wolontariackich tym, że komputery zwracają wiarygodne wyniki. Nie wymaga się replikacji danych wejściowych i weryfikacji otrzymanych wyników. Ponadto nie ma potrzeby synchronizowania obliczeń w zależności od stanu aktywności komputera wolontariusza. Klasyczne obliczenia gridowe są „niewidoczne” dla użytkowników komputera.

W oprogramowaniu *Globus Toolkit* do zarządzania zasobami wykorzystuje się oprogramowanie GRAM, które zarządza wykonaniem zadań. W szczególności przydziela zadania do zasobów, uruchamia je i monitoruje aż do ich zakończenia. GRAM współpracuje z lokalnym zarządcami zasobów LRM, za pomocą których możliwy jest dostęp do zasobów klastrów lub komputerów równoległych. GRAM współpracuje z LRM za pomocą modułu-adaptera. Usługa GGS identyfikuje klienta

użytkownika i uruchamia *menedżera zadań* JM, który obsługuje żądania zadań i koordynuje transfer plików. Ponadto, procedura kolejkowania zdarzeń GSEG przekształca paczki lokalnych danych w format niezależny od specyficznych systemów.

Czas przetwarzania paczki danych w gridach implementujących paradygmat wolontariatu obliczeniowego zależy od rodzaju projektu i waha się od kilkunastu sekund do kilkuset godzin. Pierwszość 48-ej liczby pierwszej *Mersenne'a* była testowana przez 39 dni na komputerze klasy PC. Natomiast test *Lucasa* na GPU klasy NVIDIA potrzebował 3,6 dni, a na CPU Intel i7 – 4,5 dni. Dzięki systematycznemu zapisywaniu otrzymanych wyników na komputerze wolontariusza, obliczenia nie muszą odbywać się nieprzerwanie. Uwzględnia się także fakt, że komputery wolontariuszy mogą pracować *offline*.

Polska Infrastruktura Gridowa dostarcza polskiej społeczności naukowej infrastrukturę opartą na klastrach komputerowych. Infrastruktura wspiera badania naukowe poprzez integrację danych doświadczalnych i wyników zaawansowanych symulacji komputerowych prowadzonych przez geograficznie rozproszone zespoły. W szczególności udostępniono oprogramowanie w ramach badań biologicznych, chemicznych, fizycznych oraz obliczeń numerycznych.

Grid *Comcute PG* to eksperymentalny system do obliczeń rozproszonych z udziałem wolontariuszy, w którym wykonywane są takie zadania, jak: symulacja rozprzestrzeniania się pożaru [289], analiza skutków awarii elektrowni atomowej, weryfikacja hipotezy *Collatza* czy wyznaczanie liczb pierwszych *Mersenne'a*. Ponadto prowadzone są prace nad wykorzystaniem inteligentnych agentów w warstwie pośredniczącej tego gridu [167, 301].

2. MODELE WYKORZYSTANIA ZASOBÓW W SYSTEMACH GRIDOWYCH

W systemach gridowych optymalizacja wykorzystania zasobów może być realizowana zgodnie z różnorodnymi kryteriami, w tym:

- minimalizacja obciążenia newralgicznego węzła;
- minimalizacja kosztów zakupu serwerów;
- maksymalizacja łącznej wydajności obliczeniowej gridu;
- maksymalizacja dostępności systemu;
- maksymalizacja prawdopodobieństwa realizacji zadań w terminach.

Istotne jest zatem opracowanie modelu, sformułowanie adekwatnego zagadnienia optymalizacji wielokryterialnej i skonstruowanie metody wyznaczania rozwiązań kompromisowych w odniesieniu do wykorzystania zasobów w systemach gridowych.

W literaturze przedmiotu wraz z rozwojem gridów omawiane są modele wykorzystania zasobów [364, 372]. Na ogół rozważane modele konstruowane są pod kątem ich optymalizacji ze względu na wybrane kryterium. Warto podkreślić, że niektóre modele powstały pod koniec lat siedemdziesiątych ubiegłego wieku, jeszcze przed skonstruowaniem pierwszych gridów [326]. Wprawdzie opisywały one szeroko rozumiane systemy rozproszone, ale wiele założeń, ograniczeń i kryteriów z tych modeli pozostawiono lub tylko nieco zmodyfikowano w modelach współczesnych gridów. Oczywiście, pojawiły się nowe założenia, wymagania i kryteria oceny nowoczesnych gridów, które powodują, że modele wykorzystania zasobów w gridach są zasadniczo inne niż wcześniejsze modele dla systemów rozproszonych.

Ze względu na dużą różnorodność zasobów i wielkość gridów liczba możliwych zmiennych decyzyjnych może być tak duża, że w praktyce problemu optymalizacji wykorzystania zasobów nie można rozwiązać w zadanym czasie [326]. Z tego powodu stosuje się różne podejścia do modelowania wykorzystania zasobów. Jednym z nich jest zawężenie rozważań do krytycznych zasobów i zadań. Ponieważ kluczową rolę odgrywa warstwa pośrednicząca gridu, to optymalizacji podlega jedynie wykorzystanie zasobów przez moduły programistyczne tej warstwy dla najważniejszych scenariuszy.

Ponadto redukuje się liczbę zasobów do takich, jak: procesory, pamięć RAM, pamięć dyskowa, przepustowość układów we/wy, czy specjalistyczne urządzenia zewnętrzne. W rozprawie rozpatruje się konfiguracje gridu, którymi są zestawy

komputerów wraz z przydzielonymi do nich zadaniami. W ten sposób wykorzystanie zasobów jest dopasowane do zapotrzebowania na nie ze strony zadań.

2.1. Przegląd wybranych modeli wykorzystania zasobów w gridach

Oprogramowanie w gridzie jest zbiorem powiązanych ze sobą modułów programistycznych, co odnosi się nie tylko do oprogramowania warstwy pośredniczącej, ale również do większości aplikacji użytkowników [175]. Podział oprogramowania gridowego na moduły umożliwia skrócenie czasu wykonania zadań poprzez zrównoleglenie ich wykonania, a ponadto redukuje koszt modyfikacji oprogramowania. Warto zauważyć, że wprowadzanie zmian deweloperskich odnosi się wówczas do podzbioru modułów, a nie do całego systemu [1, 194].

Moduł programistyczny zawiera wyodrębniony zestaw instrukcji języka programowania oznaczony za pomocą identyfikatora i ograniczników, co umożliwia odwołanie się do niego [33]. Moduł zawiera także dane lokalne i implementuje zazwyczaj zadaną funkcjonalność [45]. Modułem mogą być powiązane tematycznie klasy [261]. Podział oprogramowania na moduły powinien być taki, aby powiązania między modułami były jak najmniejsze oraz aby jak najmniej elementów jednego modułu miało wpływ na budowę innego. Programiści powinni koncentrować się na interfejsie i specyfikacji modułu, a nie na sposobie implementacji modułu [269].

Moduł jako zawartość pliku przechowywanego na dysku jest obiektem pasywnym w odróżnieniu od zadania, które jest jego wykonaniem [264]. Czas wykonania zadania można oszacować na podstawie kodu źródłowego modułu lub zmierzyć dla wybranych komputerów. Przyjmuje się, że czasy te znane są *a priori*.

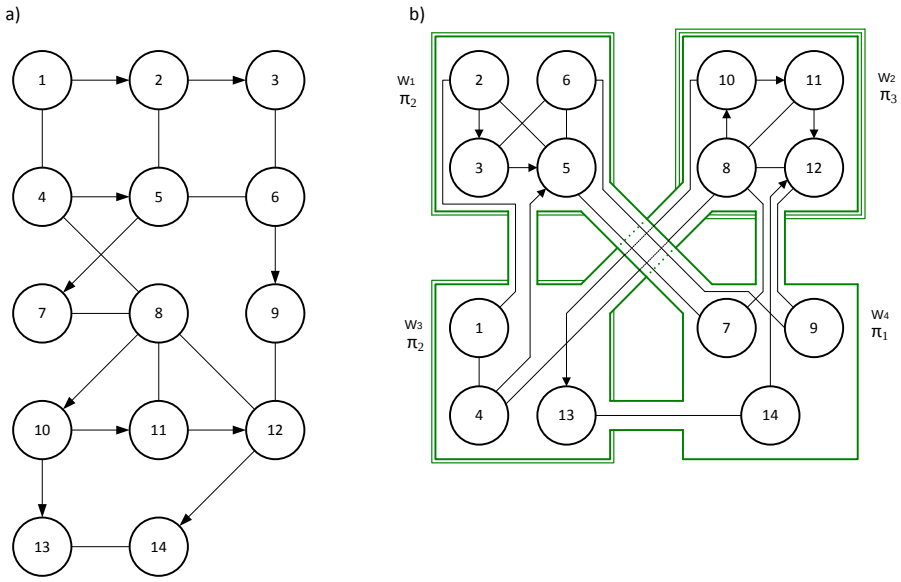
Dzielenie programu na moduły ułatwia zarządzanie projektem realizowanym przez zespół programistów, co skutkuje podniesieniem jakości produktu [266]. W ten sposób można poprawić przejrzystość pisanych aplikacji, a także obniżyć ryzyko popełnienia błędu. Jeśli programista-tester dokładnie przetestuje moduł, to można wielokrotnie i efektywnie wykorzystać raz napisany kod [267]. Taka sytuacja występuje przy rozszerzaniu funkcjonalności w warstwie pośredniczącej gridu lub też dodawaniu nowych aplikacji użytkowników [268].

W gridzie można wyznaczyć przydział zadań do komputerów, co nazywamy *konfiguracją gridu* [171]. Jeśli każde zadanie jest wykonywane na tym samym komputerze, co stowarzyszony z nim moduł, to przydział modułów można utożsamiać

z przydziałem zadań [270]. Problemy wyznaczania konfiguracji gridów należą do fundamentalnych aspektów zarządzania zasobami, ponieważ umożliwiają pełniejsze dopasowanie właściwości komputerów do cech modułów. Instrukcje graficzne lub instrukcje w języku CUDA C++ zapisane w module będą szybciej wykonywane na karcie graficznej NVIDIA niż na procesorze serwera bazodanowego, do którego należałoby raczej przydzielić moduł z instrukcjami w języku SQL.

Warto podkreślić, że moduły warstwy pośredniczącej w gridzie *Comcute* napisano w języku *Java*, a komunikacja między nimi odbywa się zgodnie z modelem *klient-serwer* z wykorzystaniem serwera aplikacji *GlassFish* [400]. Natomiast interfejsy webowe do inicjacji zadań dostępne są w dowolnym momencie za pomocą przeglądarki internetowej z dowolnego miejsca [272].

Przykładową konfigurację gridu do wspomagania zdalnego nauczania *Moodle* przedstawiono na rysunku 10 [366].



Rys. 10. Moduły systemu *Moodle*:
a) diagram powiązań między modułami
b) przykładowa konfiguracja modułów w gridzie [366]

Łuk (1,2) reprezentuje przesyłanie danych z modułu nr 1 do nr 2 lub fakt, że moduł nr 2 wykona się po zakończeniu realizacji modułu nr 1. Natomiast krawędź (1,4) odnosi się do przesyłania danych między parą modułów (1,4) w obie strony co najmniej raz podczas wykonywania programu. Kwadraty reprezentują komputery połączone kanałami komunikacyjnymi. Komputery usytuowano w czterech węzłach w_1, w_2, w_3 i w_4. Czternaście modułów przydzielono do komputerów, których rodzaje oznaczono jako π_1, π_2 oraz π_3. Komputery klasy π_2 usytuowano w węzłach w_1 oraz w_3. Rodzaj

komputera cechuje się unikatowym zestawem modelowanych zasobów, np. wielkości pamięci, wydajności procesora czy GPU.

Można zminimalizować koszt realizacji zadań z warstwy pośredniczącej lub gridowej aplikacji użytkownika za pomocą przydzielenia modułów do komputerów w odpowiedni sposób [45]. Alternatywnie minimalizacji może podlegać łączny czas przetwarzania danych. Podstawowe podejście do optymalizacji wykorzystania zasobów gridowych polega na sformułowaniu adekwatnego zagadnienia optymalizacji i jego rozwiązaniu [46].

Istotnym kryterium optymalizacji wykorzystania zasobów jest łączny czas komunikacji między modułami. Aby zredukować czas komunikacji między komputerami, współpracujące moduły powinny rezydować na tym samym komputerze. Z drugiej strony, aby skrócić czas obliczeń, moduły należy przydzielić tam, gdzie wykonują się najszybciej.

W systemach heterogenicznych istotną przyczyną rozproszenia modułów jest dążenie do wykorzystania specyficznych cech różnych typów komputerów. Jeżeli zatem para modułów cechuje się najkrótszymi czasami realizacji na dwóch różnych komputerach, to korzyści czasowe, wynikające z przydziału minimalizującego łączny czas wykonania modułów, mogą być zniwelowane jedynie przez opóźnienia komunikacyjne. Występuje zatem konflikt między minimalizacją czasu komunikacji a minimalizacją czasu obliczeń na komputerach w gridzie [262].

Niektóre metody wyznaczania konfiguracji gridu opierają się na algorytmach z teorii grafów [326]. Metody te są wrażliwe na wystąpienie „nietypowych" interakcji między modułami. Specyficzne interakcje mogą zakłócić „regularność" grafu reprezentującego komunikację danych między modułami. Jest to zasadnicza wada podejść wywodzących się z teorii grafów.

Pionierski eksperyment z optymalizacją konfiguracji systemu rozproszonego przeprowadzono na systemie graficznym w *Brown University*, w którym moduły programu podzielono między komputer IBM 360 a mikrokomputer graficzny [103]. Zastosowanie grafowej metody optymalizacji rozdziału modułów, którą zaproponował *Stone* [326], znacząco zmniejszyło koszt obliczeń. Modelem był graf z odpowiednią funkcją kosztu określoną na łukach. Metoda rozwiązania problemu polegała na skonstruowaniu grafu przydziału, a następnie na wyznaczeniu maksymalnego przepływu. Złożoność zaproponowanej metody to $O(V^3)$, gdzie V oznacza liczbę modułów [326].

Model *Stone'a* nie uwzględniał istotnych ograniczeń wynikających z wielkości pamięci operacyjnej [326]. *Rao, Stone* i *Hu* udowodnili, że zadanie minimalizacji kosztu aplikacji rozproszonej w systemie dwukomputerowym z ograniczoną wielkością pamięci operacyjnej należy do klasy problemów NP-trudnych [292]. Natomiast *Bokhari* [46] i *Sinclair* [314] podali, że problem optymalizacji przydziału modułów polegający na minimalizacji kosztów wykonania aplikacji rozproszonej o dowolnej strukturze interakcji między modułami, dla co najmniej czterech komputerów, należy do klasy problemów NP-trudnych, nawet przy braku ograniczeń wynikających z limitowanych wielkości zasobów.

Bokhari opracował efektywną metodę optymalizacji konfiguracji w odniesieniu do programu hierarchicznego, który cechuje się tym, że moduł nadrzędny wywołuje moduły poziomu niższego, a te z kolei wywołać mogą jednostki należące do jeszcze niższego poziomu hierarchii [46]. Nie dopuszcza się do uaktywnienia modułu przez co najmniej dwa segmenty z poziomu nadrzędnego. Moduł może wchodzić w interakcje jedynie z jednostkami niższego poziomu oraz z co najwyżej jednym modułem stopnia nadrzędnego.

W metodzie wykorzystuje się procedurę wyznaczania najkrótszej ścieżki w odpowiednio skonstruowanym grafie przydziału, a jej cechą charakterystyczną jest złożoność obliczeniowa rzędu $O(n^3)$, gdzie $n = \max\{V, k\}$, przy czym k reprezentuje liczbę poziomów hierarchii [46]. Metody *Bokhariego* nie można wykorzystać w wielu gridach, w tym w systemie *Comcute*, ponieważ moduły z poziomu *W* tego gridu komunikują się ze sobą.

Efektywna metoda optymalizacji konfiguracji systemu rozproszonego została opracowana przez *Towsleya* dla oprogramowania o strukturze sekwencyjno-równoległej [339]. W metodzie stosuje się procedurę wyznaczania najkrótszej ścieżki w konstruowanym grafie przydziału. Złożoność obliczeniowa metody jest rzędu $O(n^4)$, gdzie $n=\max\{V, k\}$, przy czym k reprezentuje liczbę poziomów hierarchii modułów w programie sekwencyjno-równoległym [339].

Przetwarzanie potokowe o strukturze łańcucha analizowano w [46]. W tym modelu minimalizowanym kryterium jest czas trwania ramki zdefiniowany jako czas trwania cyklu wykonywania modułów podczas przetwarzania potokowego na najbardziej obciążonym komputerze [46]. Efektywna metoda minimalizacji czasu trwania ramki cechuje się złożonością obliczeniowa $O(V^4)$. Natomiast do optymalizacji przydziału modułów w odniesieniu do zbioru rozproszonych programów potokowych

o strukturze gwiazdy opracowano metodę minimalizacji czasu trwania ramki o złożoności obliczeniowej $O(V^3 \log V)$ [46].

Interesujący nurt badań ukierunkowano na opracowanie metod optymalizacji przydziału (ang. *mapping*) grafu reprezentującego powiązane moduły programistyczne w graf reprezentujący macierz procesorów w komputerze równoległym [54, 157, 173, 226]. W metodzie zaproponowanej przez *Liu* i *Soffa* graf przydziału dzielony jest na podgrafy, w odniesieniu do których równolegle stosowane są algorytmy optymalizacji [226].

Chaudhary i *Aggarwal* zaproponowali heurystyczną strategię *mappingu*, która bazuje na teorii grafów i programowaniu matematycznym [54]. Metoda polega na skonstruowaniu grafu problemu, który to graf jest reprezentacją algorytmu równoległego. Ponadto konstruowany jest rozszerzony graf gridu, w którym uwzględnia się przydział modułów do komputerów. Minimalizacji podlega suma opóźnień komunikacyjnych, a jeśli istnieje kilka rozwiązań o identycznej sumie opóźnień, to wybiera się alternatywę o najmniejszym łącznym czasie obliczeń i komunikacji [54].

Efektywną metodę dystrybucji danych w macierzy komputerów przedstawili *Kalns* i *Ni* [173]. Pozwala ona osiągnąć wyniki lepsze o około 40% w porównaniu z wcześniej opracowanymi metodami. Za pomocą metody *Kalnsa-Ni* minimalizuje się liczbę interakcji komunikacyjnych ze szczególnym uwzględnieniem komputerów klasy IBM SP [173].

Hsu et al. opracowali metodę dystrybucji danych w macierzy procesorów w celu redukcji liczby zdalnych odwołań aplikacji do pamięci dyskowej [157]. Poszukiwano kompromisu między wydajnością dekompozycji danych podczas wykonywania aplikacji a kosztem redystrybucji danych wśród procesorów. Opracowano funkcję *mappingu* do wyznaczania rangi procesora, na podstawie której konstruuje się kolejkę procesorów komputera równoległego IBM SP2 w celu minimalizacji liczby zdalnych odwołań do pamięci dyskowej [157].

Natomiast w metodzie *Legranda* celem jest minimalizacja czasu wykonania zadań realizowanych w heterogenicznych gridach [215]. Dane dzielone są między komputery umieszczone w tzw. wirtualnym pierścieniu. Podano zasady skonstruowania wirtualnego pierścienia w sieci. W każdej iteracji, obliczenia są wykonywane równolegle, a komunikacja odbywa się między sąsiednimi komputerami w pierścieniu. Opracowano metodę heurystyczną dla sformułowanego NP-trudnego problemu, którą zastosowano do 90 węzłów łączących klastry we Francji (*Strasbourg, Lille, Grenoble,*

Orsay), USA (*Knoxville, San Diego, Argonne*) oraz Japonii (*Nagoya, Tokio*). Metodę udoskonalono w [231].

Pierwszą metodą do minimalizacji obciążenia newralgicznego komputera była metoda *Chu* i *Lan*, za pomocą której zaprojektowano przydział 23 modułów do 3 komputerów w systemie obrony powietrznej USA [60]. Uwzględniono ograniczenia wydajnościowe, minimalizując czas komunikacji między modułami oraz sumaryczny czas wykonania modułu w newralgicznym komputerze. Ponadto, pokazano istotną rolę adekwatnego podziału oprogramowania na moduły w kontekście minimalizacji czasu odpowiedzi systemu [60].

Interesujący nurt badań zapoczątkowała praca *Price'a* i *Poocha*, w której autorzy zaproponowali nieliniową funkcję kosztów do formułowania problemów optymalizacji przydziału modułów programów w systemach rozproszonych [284]. Sposób rozwiązania problemu bazował na heurystycznej metodzie lokalnego przeszukiwania. Następnie *Iqbal* opracował metodę przeszukiwania charakteryzującą się złożonością obliczeniową $O(n^2 \log n)$ [162]. Powyższe metody nie uwzględniały ograniczeń wynikających z wielkości zasobów systemu oraz awarii węzłów.

Konakia i *Tobagi* opracowali wariant metody podziału i ograniczeń do optymalizacji przydziału modułów programów w systemie z limitowanymi zasobami lokalnymi [191]. Optymalizację przydziału modułów w przypadku ograniczonej zarówno pamięci lokalnej, jak i wydajności obliczeniowej przedstawiono w [89].

W zagadnieniach wyznaczania optymalnego wykorzystania zasobów gridu, obok kosztu wykonania aplikacji, stosuje się inne funkcje celu, takie jak łączne obciążenie systemu [47], obciążenie najbardziej eksploatowanego komputera [49], równomierność obciążenia komputerów [102] czy dostępność gridu [156]. Rozpatruje się także prawdopodobieństwo ukończenia zadań w terminach [220-223]. W aplikacjach potokowych często stosowanym kryterium jest czas ekspozycji ramki [45].

Lo zaproponowała karanie przydziału wszystkich modułów do jednego komputera w celu przeciwdziałania nadmiernemu scentralizowaniu przetwarzania, co wiąże się z niewykorzystaniem pozostałych zasobów. Ponadto zmodyfikowała heurystyczny algorytm optymalizacji *A**, który porównała z algorytmem przydziału zadań w rozproszonym systemie operacyjnym *Micros* [227].

Zagadnienia dotyczące maksymalizacji niezawodności systemów rozproszonych przedstawiono w [5, 6, 175, 308, 311]. *Kartik* i *Murthy* opracowali wariant metody podziału i ograniczeń do maksymalizacji prawdopodobieństwa realizacji wszystkich

zadań w terminach [175]. Ponadto dokonali przeglądu stosowanych metod w odniesieniu do sformułowanego NP-trudnego zadania optymalizacji [175].

Ajith i *Murthy* uwzględnili ograniczenia związane z równoważeniem obciążeń, komunikacją i zrównoleglaniem obliczeń, zapewniając tolerowanie uszkodzeń przez grid [5, 6]. Zaproponowali metodę opartą na symulowanym wyżarzaniu, którą porównano z metodą losową oraz metodą cyklicznej alokacji typu *round-robin* [5].

He et al. rozpatrywali zagadnienie minimalizacji średniego czasu odpowiedzi dla zadań klasy *non-real-time jobs*, na które nie nałożono terminów ich ukończenia [151]. Ponadto minimalizowali średnie tempo przekraczania wymaganych terminów zakończenia zadań w odniesieniu do zadań klasy *soft real-time jobs*, dla których preferowane jest ich ukończenie w terminach, przy czym dopuszcza się przekroczenie terminu ich wykonania. W obu wypadkach ważne jest ukończenie zadań. Ponadto wyróżnia się terminy zakończenia zadań jako *umiarkowanie twarde* (ang. *weakly-hard*), jeśli dopuszcza się przekroczenia ich w przewidywalny sposób i przy założeniu, że liczba naruszeń terminów jest ograniczona [308]. Natomiast termin zakończenia zadania nazywany jest *twardym*, jeśli jego przekroczenie może prowadzić do nieprawidłowego działania systemu [311].

W modelu *He* zakłada się, że zadania pojawiają się w gridzie w sposób losowy z zadaną średnią intensywnością. Jeśli tempo to będzie zbyt wysokie w odniesieniu do zadanego progu, to następuje ponowny rozdział obciążeń w klastrach gridu. Eksperymentalnie pokazano, że zaproponowana metoda heurystyczna poprawia oba kryteria w odniesieniu do metody *round-robin* oraz metody losowej dla instancji z 24 węzłami rozmieszczonymi w 4 klastrach [151].

Problemom konstrukcji gridów tolerujących uszkodzenia za pomocą adekwatnej konfiguracji modułów programistycznych poświęcono liczne prace [55, 159, 255, 299]. *Chen* i *Cherkassky* maksymalizowali efektywne wykorzystanie zasobów gridowych pod kątem redukcji czasu realizacji zadań [55]. Minimalizowano koszt komunikacji międzykomputerowej przy ograniczeniach na zasoby wynikające z wymagań systemowych oraz projektowych. Ograniczenia odnoszą się do czasów wykonania zadań, liczby komputerów i ich wydajności, a także wielkości pamięci [55]. Wraz ze wzrostem liczby komputerów rośnie prawdopodobieństwo awarii któregoś z nich, i dlatego rozważa się prawdopodobieństwo zrealizowania wszystkich zadań.

Przydział zadań implementowany jest dwuetapowo. W etapie pierwszym zadania przydzielane są bez uwzględnienia awarii komputerów. Natomiast w etapie drugim

zadania przydzielane są do dodatkowych komputerów w celu zwiększenia niezawodności gridu w wypadku awarii hosta podstawowego [55]. Jeśli komputer ulegnie awarii podczas wykonywania zadań, to zadania z tego komputera są przesuwane na pozostałe sprawne komputery. Powyższe wprawdzie wydłuża czas wykonania zadań, ale umożliwia ich zakończenie. Pokazano, że to podejście jest bardziej praktyczne od dynamicznej rekonfiguracji zadań oraz stosowania technologii przywracania stanu gridu do stanu sprzed określonego przedziału czasowego. W metodzie nie stosuje się równoległego wykonywania kopii zadań na różnych komputerach i procedury rozstrzygania o wiarygodności uzyskanych wyników [55].

Huang i *Tripathi* zaproponowali maksymalizowanie odporności na uszkodzenia (ang. *fault-tolerance*) w odniesieniu do węzłów gridu [159]. Każdy węzeł może być skojarzony z kilkoma węzłami zapasowymi. W alternatywnym wariancie gridu nie ma dedykowanych węzłów zapasowych, gdyż każdy węzeł używa innych jako zapasowe. Zatem każdy węzeł może być podstawowy i jednocześnie zapasowy dla innych węzłów. Jeśli węzeł ulega uszkodzeniu, to wszystkie zapytania przydzielone do tego węzła są przekierowywane do zapasowego węzła. Metoda przydziału zasobów uwzględnia tempo awarii/naprawy węzłów oraz funkcję kosztów związaną ze zwiększaniem odporności na błędy [159].

Xie i *Qin* [357] opracowali metodę przydziału modułów w klastrach, w której oprócz miary skutecznej realizacji zadań w terminach, maksymalizuje się również poziom bezpieczeństwa. Przegląd algorytmów, w których cyklicznie wykonywane moduły mogą zostać wykonane w terminach, zaprezentowano w [153]. Przyjęto, że moduły nie są poddane ograniczeniom kolejnościowym i mogą zostać wywłaszczone w dowolnym momencie.

Na tym tle niezmiernie interesujące jest rozważenie innego rodzaju decyzji podejmowanych na etapie projektowania gridów, które to decyzje mają wpływ na skrócenie czasu przetwarzania danych. Kluczową decyzją jest dobór odpowiednich typów komputerów, gdyż koszt systemu zależy w dużym stopniu od cen maszyn, a wydajność systemu jest związana z wydajnością obliczeniową komputerów wchodzących w jego skład [362].

Dobierając zestaw komputerów oraz przydzielając do nich moduły, można uzyskać rozwiązania minimalizujące koszt wykonania programu oraz maksymalizujące wydajność systemu. Wymagane jest wówczas wprowadzenie dodatkowych założeń do modeli znanych z literatury przedmiotu [20].

W wypadku przetwarzania współbieżnego minimalizacja kosztu przetwarzania jest związana z doborem tańszych komputerów o mniejszej mocy obliczeniowej. W rezultacie może to prowadzić do wydłużenia czasu wykonania programu.

Modelowanie zaistniałych sytuacji konfliktowych może być oparte na sformułowaniu odpowiednich zagadnień optymalizacji wielokryterialnej z uwzględnieniem wybranych preferencji decyzyjnych. Dlatego w rozprawie położono nacisk na formułowanie zagadnień polioptymalizacji, w których poszukuje się reprezentacji rozwiązań optymalnych w sensie *Pareto* [18, 201-203].

W powyższym kontekście warto zwrócić uwagę na metody sztucznej inteligencji, które można wykorzystać do rozwiązania zagadnień polioptymalizacji. Za pomocą algorytmów ewolucyjnych wyznaczono najlepsze rozwiązania, jakie uzyskano do tej pory dla szeregu instancji zagadnień optymalizacji [125, 174, 230].

W pracy [59] opracowano model podziału i alokacji obiektów do odwzorowania aplikacji zorientowanych obiektowo na środowisko heterogeniczne. Wykorzystano algorytm genetyczny NPGA (ang. *Niched-Pareto Genetic Algorithm*). Wielokryterialne algorytmy ewolucyjne wykorzystuje się do wyznaczania reprezentacji efektywnych przydziałów modułów [88, 148]. Warto podkreślić, że algorytmów harmonicznych nie zastosowano do tej pory w tego rodzaju zagadnieniach optymalizacji.

2.2. Zbiór konfiguracji dopuszczalnych

Adekwatne skonfigurowanie gridu umożliwia efektywne wykorzystanie jego zasobów pod kątem założonego celu. Przykładowo, administrator gridu *Comcute* może dokonać jego rekonfiguracji za pomocą przydzielenia modułów klasy *W* lub *S* do komputerów ulokowanych w węzłach, które to moduły należą do zbioru $\boldsymbol{A}=\{\alpha_1,...,\alpha_v,...,\alpha_V\}$. W szczególności administrator może dodać moduł w wybranym węźle, usunąć moduł lub przesunąć moduł do innego węzła. Z punktu widzenia projektanta, możliwe jest także dodanie węzła, jego redukcja lub wymiana komputerów w istniejących węzłach. Działanie to powinno wynikać z interpretacji rozwiązania odpowiedniego zagadnienia optymalizacji zgodnie z wybranym kryterium wykorzystania zasobów.

Rolę administratora lub projektanta w tym zakresie może pełnić aplikacja wyznaczająca automatycznie pożądaną rekonfigurację systemu. W ten sposób grid będzie się dynamicznie i samodzielnie „przystosowywał” do zmieniającego się

obciążenia, a także preferencji decydentów. Możliwe są zatem rekonfiguracje gridu zachodzące w wyniku wystąpienia pewnego zdarzenia, np. przekroczenia progu dotyczącego liczby uruchomionych zadań przez użytkowników, liczby zgłoszonych komputerów wolontariuszy lub w wyniku planowanego zmniejszenia dostaw energii elektrycznej. Ponadto rekonfiguracja może być wykonywana cyklicznie w zależności od pory dnia, dnia tygodnia lub miesiąca, na podstawie zarejestrowanych wcześniej statystyk obciążeń. W szczególności uzasadnione jest zwiększenie wydajności i zużycia energii w godzinach nocnych [74, 75].

Zakłada się, że moduł α_v może być wykonany na komputerach, których rodzaje należą do zadanego zbioru $\boldsymbol{B} = \{\beta_1,...,\beta_j,...,\beta_J\}$. Komputery te mogą różnić się między sobą liczbą i jakością procesorów, wielkością pamięci RAM, wielkością pamięci zewnętrznych lub dostępnością do urządzeń zewnętrznych, np. macierzy dyskowej lub drukarki 3D.

W wersji laboratoryjnej gridu *Comcute* w grudniu 2014 roku dostępnych było dziewięć węzłów obliczeniowych, w których można instalować moduły *W* lub *S*. W gridzie istnieje możliwość bezpośredniej rekonfiguracji, a po zakupie kolejnych hostów i zainstalowaniu odpowiedniego oprogramowania (system operacyjny, serwer aplikacji) można rozszerzyć grid o dodatkowe węzły. Natomiast w sytuacji zmniejszenia zapotrzebowania na moc obliczeniową gridu, administrator może wyłączyć wyznaczoną liczbę węzłów lub zredukować liczbę pracujących modułów w warstwie pośredniczącej.

Węzły obliczeniowe mogą być zlokalizowane w centrach obliczeniowych konsorcjów zrzeszających: uczelnie, szkoły, instytucje wojskowe, ośrodki straży pożarnej, a także jednostki administracji publicznej. Przyjmuje się założenie, że ze względów bezpieczeństwa komputery realizujące zadania warstwy pośredniczącej mogą być usytuowane tylko w ustalonych węzłach należących do zbioru $\boldsymbol{W} = \{w_1,...,w_i,...,w_I\}$.

Ponadto zakłada się, że w węźle usytuowany jest dokładnie jeden komputer wybrany ze zbioru $\boldsymbol{B}$. Niech zadana jest macierz skumulowanych czasów realizacji zadań na komputerach $\boldsymbol{T}=[t_{vj}]_{V\text{x}J}$, gdzie t_{vj} oznacza skumulowany czas wykonania zadania α_v na komputerze typu β_j [152]. Czasem cząstkowym realizacji zadania nazywamy czas jego wykonywania od momentu rozpoczęcia lub wznowienia do

momentu zakończenia lub wywołania innego zadania. Skumulowany czas wykonania zadania jest sumą czasów cząstkowych podczas jego wykonywania.

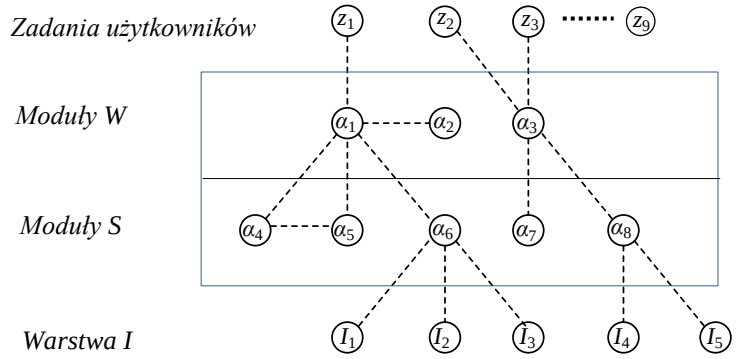
W modelu przyjęto, że zadana jest macierz skumulowanych czasów komunikacji danych między parami zadań $\boldsymbol{\tau} = [\tau_{vu}]_{V \times V}$, gdzie τ_{vu} oznacza skumulowany czas transmisji danych do zadania α_u z zadania α_v. W niektórych modelach przyjmuje się, że czas komunikacji między parą zadań jest znany *a priori*, stały i zależy od komunikującej się pary [183]. Wprowadzenie tego założenia wynika z faktu, że czas komunikacji między parą zadań zależy od objętości przesyłanych danych oraz od przepustowości połączenia wirtualnego między nimi.

Jeśli przepustowość połączenia wirtualnego między zadaniami jest jednakowa, to czas komunikacji między procesami nie zależy ani od komputerów, ani od tego, do jakich węzłów zostały one przydzielone. Jeśli zadania są realizowane w tym samym węźle, to przyjmuje się, że nie występują opóźnienia komunikacyjne między zadaniami. Zakłada się, że $\tau_{vv} = 0$ dla $v = \overline{1,V}$ [242].

Stosowanych jest kilka sposobów komunikacji między modułami w gridzie. Pierwszy sposób polega na reagowaniu na zdarzenia, co jest związane z asynchronicznym przesyłaniem komunikatów. Moduł źródłowy wysyła komunikat do kolejki zdarzeń modułu docelowego [257]. Kolejny sposób komunikacji między modułami polega na tym, że niektóre moduły bezpośrednio używają lokalnych danych innego modułu. Jest to forma silnego powiązania komunikacyjnego modułów [285].

Interesująca strategia, będąca rozszerzeniem poprzedniej, polega na użyciu API (ang. *Application Programming Interface*) innego modułu [288]. Rozwijana jest także technika komunikacji między modułami polegająca na zastosowaniu dedykowanego pliku bibliotek. Pozwala to na implementację wybranych funkcji do interakcji między modułami [296].

Przyjmuje się, że podstawowy model gridu jest opisany za pomocą uporządkowanej piątki (***A***, ***B***, ***W***, ***T***, $\boldsymbol{\tau}$) [366]. W rozprawie pod pojęciem *konfiguracji modułów gridu* rozumie się zbiór relacji między modułami programistycznymi. Na rysunku 11 zaznaczono interakcje polegające na przesyłaniu danych między modułami. Jeżeli między parą modułów zachodzą interakcje, to taka sytuacja jest oznaczana krawędzią. W gridzie *Comcute* zadanie użytkownika rozpoczyna obliczenia na serwerze W_0, który organizuje grupę serwerów *W'* do zrównoleglania obliczeń. Na rysunku 11 moduły α_1 oraz α_2 tworzą grupę *W'*, przy czym rolę W_0 pełni α_1.



Rys. 11. Wybrana konfiguracja modułów w systemie *Comcute*
Źródło: opracowanie własne.

Poprzez serwery *S* zadanie dostarczane jest do węzłów reprezentujących komputery internautów. Ponadto serwery z grupy *W'* przesyłają strumienie paczek danych przez serwery *S* do zadań użytkownika, które przemieszczono do komputerów internautów *I*.

Istotną decyzją jest przydzielenie modułu W_0 do zadania użytkownika. Mamy zatem taką sytuację, że zadania $z_1, \ldots z_m, \ldots z_M$ należy przydzielić do serwerów *W*, których jest V_W. Moduły klasy *W* oraz *S* indeksuje się w ten sposób, że moduły klasy *W* otrzymują numery od 1 do V_W, a moduły klasy *S* – od V_W+1 do *V*. Na rysunku 11 zaznaczono osiem modułów w warstwie pośredniczącej, przy czym trzy pierwsze są modułami *W*, a kolejnych pięć – klasy *S*. Przydział zadań użytkowników do modułów zarządzających definiuje się następująco:

$$x^z = [x^z_{11},\ldots,x^z_{1v},\ldots,x^z_{1V},\ldots,x^z_{mv},\ldots,x^z_{MV}]^T, \tag{2.2.1}$$

gdzie:

$$x^z_{mv} = \begin{cases} 1 \text{ gdy } z_m \text{ rozpoczyna się w } W_v, \\ 0 \text{ w przeciwnym razie,} \end{cases} \text{ dla } m = \overline{1,M},\ v = \overline{1,V_W}.$$

Ponieważ proces dystrybucji każdej aplikacji zleceniodawcy musi rozpoczynać się w dokładnie jednym module klasy *W*, to wprowadza się poniższe ograniczenie:

$$\sum_{v=1}^{V_W} x^z_{mv} = 1 \text{ dla } m = \overline{1,M}. \tag{2.2.2}$$

Aby spełnić ograniczenie (2.2.2), można zastosować specyfikację przydziału zadań użytkowników do modułów klasy *W* za pomocą wektora $X^z = [X^z_1,\ldots,X^z_m,\ldots,X^z_M]^T$, gdzie X^z_m oznacza numer modułu klasy *W*, od którego rozpoczyna się dystrybucja kodu

i danych zadania z_m. Jeśli $x_{mv}^z = 1$, to $X_m^z = v$, przy czym $1 \le X_m^z \le V_W$. Przykładowy przydział zlecanych zadań użytkowników do modułów zarządzających klasy *W* z rysunku 11 można zapisać za pomocą wektora $X^z = [1, 3, 3]^T$.

Znając przydział zadań użytkownika do modułów zarządzających, można oszacować obciążenie modułów klasy *W*, a to z kolei wpływa na obciążenie modułów dystrybucyjnych klasy *S*. Natomiast przydział modułów z warstwy pośredniczącej do węzłów można określić za pomocą binarnego wektora w następujący sposób:

$$x^\alpha = [x_{11}^\alpha, ..., x_{1i}^\alpha, ..., x_{1I}^\alpha, ..., x_{vi}^\alpha, ..., x_{VI}^\alpha]^T, \tag{2.2.3}$$

gdzie $x_{vi}^\alpha = \begin{cases} 1 \text{ gdy } \alpha_v \text{ usytuowanow } w_i, \\ 0 \text{ w przeciwnymrazie,} \end{cases}$ dla $v = \overline{1,V}$, $i = \overline{1,I}$.

Zakładamy, że w jednym węźle może być realizowanych kilka modułów *W* oraz *S*. Przydział modułów do węzłów można scharakteryzować także za pomocą binarnej macierzy $\overline{X}^\alpha = [x_{vi}^\alpha]_{V \times I}$. Ponieważ każdy moduł powinien zostać przydzielony tylko do jednego węzła, to powyższy postulat formalny można zapisać następująco:

$$\sum_{i=1}^{I} x_{vi}^\alpha = 1 \text{ dla } v = \overline{1,V}. \tag{2.2.4}$$

Aby spełnić ograniczenie (2.2.4), można zastosować specyfikację przydziału modułów do węzłów za pomocą całkowitoliczbowego wektora przydziału $X^\alpha = (X_1^\alpha, ..., X_v^\alpha, ..., X_V^\alpha)^T$, gdzie X_v^α oznacza numer węzła, do którego przydzielono moduł α_v. Jeżeli $x_{vi}^\alpha = 1$, to $X_v^\alpha = i$, przy czym $1 \le X_v^\alpha \le I$.

Binarny wektor przydziału x^α umożliwia formułowanie istotnych ograniczeń, funkcji kryteriów i w konsekwencji zagadnień optymalizacji [353]. Natomiast całkowitoliczbowy wektor X^α stosowany jest w metodach optymalizacji [34].

Sposób kodowania przydziałów ma znaczący wpływ na zredukowanie przestrzeni przeszukiwań podczas optymalizacji parametrów gridu. Przykładowo, liczba możliwych (dopuszczalnych lub niedopuszczalnych) rozwiązań zapisanych za pomocą binarnego wektora przydziału x^α jest równa 2^{VI} i rośnie szybciej niż liczba możliwych wariantów całkowitoliczbowego wektora X^α, która wynosi I^V.

Najszybszym komputerem na świecie w listopadzie 2014 roku był *Tianhe-2* z *China's National University of Defense Technology*, który to superkomputer cechował się wydajnością 33,86 PFLOPS (kwadryliony operacji na sekundę) podczas wykonywania testu *the Linpack benchmark* [405, 406]. Niech *Tianhe-2* w ciągu

sekundy porówna $33{,}86\times10^{15}$ wariantów konfiguracji modułów. Zgodnie z tabelą 1 w wypadku wektora binarnego x^{α} należy ocenić $1{,}268 \times 10^{30}$ wariantów dla 10 modułów i 10 węzłów, a zatem czas porównania tych wariantów na *Tianhe-2* wyniósłby aż 11 875 lat. Warto podkreślić, że w wypadku zastosowania wektora X^{α} powyższa metoda wyznacza optymalne rozwiązanie wg założonego kryterium zaledwie w ciągu 2,95 mikrosekundy. Wnioskujemy stąd, że odpowiednie kodowanie rozwiązania znacząco redukuje przestrzeń przeszukiwań i skraca czas wyznaczenia konfiguracji optymalnych. Przy ustalonej liczbie węzłów oraz liczbie modułów kodowanie całkowitoliczbowe znacząco redukuje liczbę wariantów (tab. 1).

Tabela 1. Liczba możliwych wariantów konfiguracji systemu gridowego w zależności od liczby modułów oraz liczby węzłów [366]

V	Liczba węzłów *I*							
	2		5		10		100	
	Bin	*Int*	*Bin*	*Int*	*Bin*	*Int*	*Bin*	*Int*
2	16	4	1024	25	$1{,}049 \times 10^{6}$	10^{2}	$1{,}607 \times 10^{60}$	10^{4}
10	$1{,}049 \times 10^{6}$	1024	$1{,}126 \times 10^{15}$	$9{,}766 \times 10^{6}$	$1{,}268 \times 10^{30}$	10^{10}	$1{,}072 \times 10^{301}$	10^{20}
10^{2}	$1{,}607 \times 10^{60}$	$1{,}268 \times 10^{30}$	$3{,}273 \times 10^{150}$	$7{,}889 \times 10^{69}$	$1{,}072 \times 10^{301}$	10^{100}	$1{,}436 \times 10^{3004}$	10^{200}
10^{3}	$1{,}148 \times 10^{602}$	$1{,}072 \times 10^{301}$	$1{,}486 \times 10^{1502}$	$4{,}223 \times 10^{684}$	$1{,}436 \times 10^{3004}$	10^{1000}	$1{,}325 \times 10^{30005}$	10^{2000}

Bin - kodowanie binarne wektora x^{α},
Int - kodowanie całkowitoliczbowe wektora X^{α}.

Mimo tak znaczącej redukcji przestrzeni przeszukiwań za pomocą kodowania całkowitoliczbowego, metoda przeglądu nie może być zastosowana do instancji o znacząco większym rozmiarze, nawet przy wykorzystaniu najbardziej wydajnych superkomputerów. Przy zwiększeniu *V* z 10 do 100 potrzeba aż $9{,}36\times10^{73}$ tysięcy lat obliczeń za pomocą tej metody z kodowaniem X^{α}. Stąd istotne jest opracowanie metody optymalizacji, która cechowałaby się znacznie krótszym czasem obliczeń.

Przy poczynionych założeniach, w ciągu godziny obliczeń na *Tianhe-2* może być porównane $1{,}22\times10^{19}$ wariantów, co umożliwia wybór najlepszego rozwiązania dla 8 modułów i 8 węzłów, przy reprezentacji binarnej x^{α}, a przy specyfikacji całkowitoliczbowej X^{α} - dla 16 segmentów programów i 16 węzłów,

Wzrost 1000-krotny mocy najszybszych superkomputerów - co zajmuje niespełna 15 lat przy założeniu, że moc obliczeniowa podwaja się co 1,5 roku - spowoduje, że będą mogły być porównane $1{,}22\times10^{22}$ warianty. Wówczas w wypadku wektora x^{α} będzie możliwe rozważenie 9 modułów programu i 8 węzłów, a dla wektora X^{α} – 18 modułów i 17 węzłów. Niestety, nawet istotny przyrost mocy obliczeniowej komputera

nie gwarantuje zatem wyznaczenia rozwiązania optymalnego za pomocą metody pełnego przeglądu dla obu sposobów kodowania konfiguracji.

W wypadku rozbudowy gridu zbiór możliwych rodzajów komputerów scharakteryzowany jest za pomocą wektora kosztów ich zakupu $\xi = [\xi_1,...,\xi_j,...,\xi_J]^T$, gdzie ξ_j jest kosztem zakupu komputera *j*-tego rodzaju, który udostępnia określony zestaw zasobów. Do sformułowania zagadnień optymalizacji wygodna jest reprezentacja przydziału typów komputerów do węzłów w postaci binarnego wektora przydziału typów komputerów do węzłów, który zapisuje się, jak niżej:

$$x^{\beta} = [x_{11}^{\beta},...,x_{1j}^{\beta},...,x_{1J}^{\beta},...,x_{ij}^{\beta},...,x_{IJ}^{\beta}]^T, \tag{2.2.5}$$

gdzie

$$x_{ij}^{\beta} = \begin{cases} 1 \text{ gdy do } w_i \text{ przydzielono komputer typu } \beta_j, \\ 0 \text{ w przeciwnym razie,} \end{cases} \quad \text{dla } i = \overline{1,I},\ j = \overline{1,J}.$$

Możliwych jest 2^{IJ} przydziałów komputerów do węzłów, które są reprezentowane za pomocą wektora x^{β}. W podstawowym modelu przydziału modułów przyjmuje się, że w węźle usytuowany jest dokładnie jeden komputer, co implikuje następujące ograniczenie:

$$\sum_{j=1}^{J} x_{ij}^{\beta} = 1 \text{ dla } i = \overline{1,I}. \tag{2.2.6}$$

Ograniczenie (2.2.6) jest spełnione, jeżeli przydział rodzajów komputerów do węzłów jest przedstawiony w postaci całkowitoliczbowego wektora przydziału komputerów do węzłów, jak niżej:

$$X^{\beta} = [X_1^{\beta},...,X_i^{\beta},...,X_I^{\beta}]^T, \tag{2.2.7}$$

gdzie X_i^{β} reprezentuje numer rodzaju komputera przydzielonego do węzła w_i.

Warto zwrócić uwagę, że ograniczenia $\sum_{i=1}^{I} x_{vi}^{\alpha} = 1, v = \overline{1,V}$ oraz $\sum_{j=1}^{J} x_{ij}^{\beta} = 1, i = \overline{1,I}$ stanowią układ $V+I$ równań. Jeżeli $x_{ij}^{\beta} = 1$, to $X_i^{\beta} = j$. Ponadto $1 \le X_i^{\beta} \le J$ dla $i = \overline{1,I}$. Możliwych jest J^I całkowitoliczbowych przydziałów komputerów do węzłów.

Przydział modułów do węzłów oraz przydział komputerów do węzłów nazywamy *konfiguracją gridu,* co specyfikuje się następująco:

$$x = [x_{11}^{\alpha},...,x_{1i}^{\alpha},...,x_{1I}^{\alpha},...,x_{vi}^{\alpha},...,x_{VI}^{\alpha},x_{11}^{\beta},...,x_{1j}^{\beta},...,x_{1J}^{\beta},...,x_{ij}^{\beta},...,x_{IJ}^{\beta}]^T. \tag{2.2.8}$$

Przyjmuje się, że konfiguracja dopuszczalna powinna spełniać postulaty alokacji każdego modułu do jednego z węzłów oraz warunki przydziału tylko jednego komputera do węzła. Binarna przestrzeń przeszukiwań zawiera $2^{I(V+J)}$ elementów, a całkowitoliczbowa przestrzeń przeszukiwań - $I^V J^I$ elementów.

Z definicji (2.2.8) dotyczącej konfiguracji x wynika, że binarna przestrzeń przeszukiwań $\boldsymbol{A}_{\text{bin}} = \boldsymbol{B}_{\text{bin}}{}^{I(V+J)}$, gdzie $\boldsymbol{B}_{\text{bin}} = \{0,1\}$. Binarna przestrzeń przeszukiwań zawiera $2^{I(V+J)}$ elementów. Liczebność binarnej przestrzeni przeszukiwań rośnie znacząco szybciej niż 2^{IV}, które to wartości wyznaczono w tabeli nr 1.

Niech w zbiorze możliwych konfiguracji $\boldsymbol{X}$ uwzględnia się ograniczenia formalne (2.2.4) i (2.2.6), jak niżej:

$$\boldsymbol{X} = \{x \in \boldsymbol{B}_{\text{bin}}{}^{I(V+J)} \mid \sum_{i=1}^{I} x_{vi}^{\alpha} = 1 \text{ dla } v = \overline{1,V};\ \sum_{j=1}^{J} x_{ij}^{\beta} = 1 \text{ dla } i = \overline{1,I}\ \}. \qquad (2.2.9)$$

Zbiór $\boldsymbol{X}$ cechuje się następującą własnością.

<u>*Twierdzenie 2.1*</u> [33]

Jeżeli $V \geq 2$ oraz $J \geq 1$, to zbiór rozwiązań dopuszczalnych $\boldsymbol{X}$ określony za pomocą zależności (2.2.9), nie jest zbiorem pustym i zawiera $I^V J^I$ przydziałów modułów do komputerów.

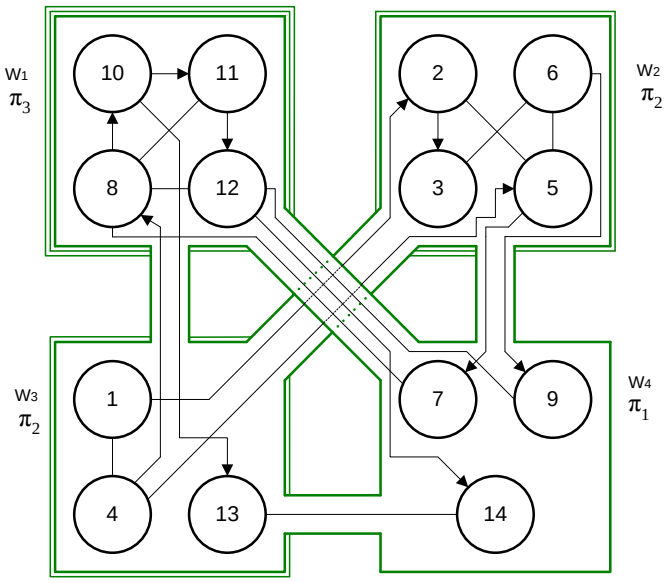
Rozwiązanie binarne x można zakodować za pomocą dwóch wektorów całkowitoliczbowych $X^{\alpha} = [X_1^{\alpha},...,X_v^{\alpha},...,X_V^{\alpha}]^T$ i $X^{\beta} = [X_1^{\beta},...,X_i^{\beta},...,X_I^{\beta}]^T$, przy czym $1 \leq X_v^{\alpha} \leq I, v = \overline{1,V}$; $1 \leq X_i^{\beta} \leq J, i = \overline{1,I}$. Kodowanie całkowitoliczbowe umożliwia znaczącą redukcję liczebności binarnej przestrzeni przeszukiwań $\boldsymbol{A}_{\text{bin}}$ do liczebności podstawowego zbioru rozwiązań dopuszczalnych $\boldsymbol{X}$. Stopień powyższej redukcji dla J=1 przedstawiono w tabeli 1, a dla J>1 stopień redukcji będzie się znacząco zwiększał.

W *Aplikacji Do Analizy I Optymalizacji Rekonfiguracji Gridu* ADAIORG'14 wykorzystano całkowitoliczbowy wektor przydziału modułów do węzłów X^{α}. Ponadto wykorzystano całkowitoliczbowy wektor przydziału rodzajów komputerów X^{β}.

W literaturze przedmiotu rozpatruje się tzw. konfiguracje *przesunięte* [366]. W konfiguracji *przesuniętej do konfiguracji referencyjnej* klaster modułów wraz z komputerem z węzła w_i przydziela się do w_k, a klaster modułów z komputerem z w_k do w_1 [33]. Zachodzi zatem zamiana miejscami dwóch hostów wraz z zainstalowanym na nich oprogramowaniem. Taką sytuację można rozpatrywać na etapie projektowania lub rekonfiguracji gridu.

Na rysunku 12 przedstawiono przykładowe rozwiązanie *przesunięte* do przydziału modułów z rysunku 10 [366]. Ponieważ każdy z *I* hostów można wymienić z serwerem spośród (*I*-1) komputerów, to w wypadku rozwiązania dopuszczalnego *x* istnieje *I*(*I*–1) rozwiązań przesuniętych. Dla dwóch węzłów rozwiązania przesunięte nazywane są rozwiązaniami *symetrycznymi* [33].

Oprócz ograniczeń formalnych związanych z przydziałem każdego modułu do jednego z węzłów, a także z umieszczeniem komputera w węźle, w literaturze przedmiotu uwzględnia się ograniczenia wydajnościowe, limity kosztowe, minimalny stopień decentralizacji oraz wymagania na wielkości zasobów [352].



Rys. 12. Konfiguracja przesunięta do konfiguracji z rysunku 10 [366]

Ważnym wskaźnikiem oceny możliwości obliczeniowych gridu jest jego *łączna moc obliczeniowa* dla zadanego benchmarku wydajności. Stosowanych może być wiele testów wydajności. Przykładowo, testy wydajnościowe aplikacji webowych można zaprojektować za pomocą *Visual Studio* [412]. Dla superkomputerów podstawowymi testami są *Linpack* i *SPEC* [147, 405]. Natomiast wydajność pojedynczych komputerów określonych klas może być wyznaczona za pomocą testów wydajności typu *indeks wydajności systemu Windows* [412], *3DMark* [191] czy *Whetstone* [408]. Smartfony i tablety testowane mogą być za pomocą: *Linpack*, *3DMark*, *AnTuTu*, *Quadrant*, *Epic Citadel* czy *Vellamo* [407].

Wydajność gridu powinna być wyznaczona dla reprezentatywnego testu w odniesieniu do realizowanych aplikacji. Warto wspomnieć o benchmarku, który

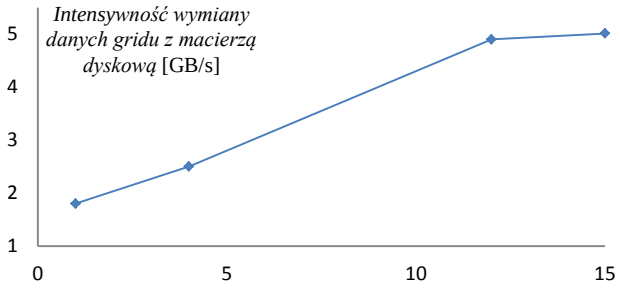
zaprojektowano dla *SAS Grid* do oceny wydajności komunikacji między siecią serwerów *HP ProLiant BL460c G7 Blades* a dyskową pamięcią masową *HP 3PAR T800 Storage System* o maksymalnej przepustowości 5,3 [GB/s] [378]. Przyjęto, że wykonywanych może być nie więcej niż 4 320 identycznych zadań. Za pomocą czterogodzinnego testu eksperymentalnie wyznaczono największą intensywność korzystania z dyskowej pamięci masowej. Wartość 5,01 [GB/s] stanowi 94,5% maksimum, przy czym wyznaczono ją dla 15 hostów oraz 2 160 zadań (144 zadań/host) [378]. Zwraca uwagę fakt, że mniejszą intensywność 4,57 [GB/s] uzyskano przy maksymalnej liczbie zadań.

Zadanie w rozpatrywanym teście jest pewnego rodzaju mieszanką opierającą się na najczęściej wykorzystywanych przez użytkowników modułach systemów informatycznych wykonywanych w gridzie. Zadanie realizuje wybrane algorytmy sortowania, procedury logistyczne, algorytmy wyznaczania ryzyka, procedury obliczania średniej, operacje na macierzach i procedury generowania zapytań w języku SQL. Symulowana jest także wymiana danych między zadaniami oraz korzystanie z baz danych [378], 50% obciążenia zadania stanowią obliczenia na CPU, a pozostałe 50% dotyczy komunikacji z pamięcią dyskową [378].

Rozmiar danych wejściowych (rekordy z bazy danych, pliki) nie przekracza 50 [GB] na zadanie, które wymaga do 1 [GB] RAM. Zadanie komunikuje się z macierzą dyskową z intensywnością ok. 25 [MB/s]. Przyjęto, że czas wykonywania zadań może być relatywnie krótki rzędu 1-2 sekund lub długi - rzędu kilku godzin [378].

Benchmark jest zarządzany przez *Grid Manager*, który dynamicznie równoważy obciążenie, przydzielając zadania do węzłów obliczeniowych. Ponadto nadaje priorytety zadaniom i zarządza ich wykonaniem w środowisku *Red Hat Enterprise Linux* z wykorzystaniem klastrowego systemu plików *Quantum StorNext File System*. Każdy host zawiera dwa sześciordzeniowe procesory *Intel Xeon* 2.8 GHz oraz 48 [GB] pamięci RAM [378]. Natomiast pojemność macierzy dyskowej to 43,2 [TB] na 144 dyskach, przy czym 30 [TB] przeznaczono na system plików.

Na rysunku 13 zobrazowano intensywność wymiany danych między serwerami a macierzą dyskową w zależności od liczby hostów dla benchmarku *SAS Grid* [378]. Wybrano największą intensywność w zależności od liczby zadań. Wadą benchmarku *SAS Grid* jest jego ukierunkowanie jedynie na pomiary intensywności wymiany danych z macierzą dyskową.



Rys. 13. Intensywność wymiany danych z macierzą dyskową w zależności od liczby hostów wyznaczona na podstawie benchmarku *SAS Grid*.
Źródło: opracowanie własne na podstawie danych z [378]

Bardziej ogólne podejście polega na wykorzystaniu *łącznej mocy obliczeniowej* Θ, którą dla zadanej konfiguracji gridu wyznaczamy za pomocą następującej zależności [33]:

$$\Theta(x) = \sum_{i=1}^{I}\sum_{j=1}^{J} \vartheta_j x_{ij}^{\beta}, \tag{2.2.10}$$

gdzie ϑ_j - wydajność hosta typu β_j, np. dla testu *Linpack* wyrażona w [FLOPS].

W literaturze przedmiotu rozpatruje się wymaganie interesariuszy, aby wydajność gridu nie była mniejsza niż założone minimum wydajności ϑ_{min} w sensie zadanego benchmarku, co zapisujemy jako następującą nierówność [338]:

$$\Theta(x) \geq \vartheta_{\min}. \tag{2.2.11}$$

Kolejnym istotnym wymaganiem projektowym jest fakt, aby łączny *koszt zakupu komputerów* $\Xi(x)$ nie przekroczył założonego w projekcie limitu finansowego $\xi_{\max}$. W tym wypadku należy wcześniej oszacować koszty zakupu komputerów $\xi = [\xi_1,...,\xi_j,...,\xi_J]$ oraz limit $\xi_{\max}$. Koszt zakupu komputerów wyznaczamy następująco:

$$\Xi(x) \leq \xi_{\max}, \tag{2.2.12}$$

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

W niektórych modelach zakłada się, że powinna istnieć co najmniej jedna para modułów przydzielonych do różnych węzłów, co umożliwia pominięcie rozwiązań, w których wszystkie moduły przydzielono do tego samego węzła i nie są wykorzystane pozostałe komputery [336].

Powyższy warunek odpowiada nierówności:

$$\upsilon(x) \geq 1, \tag{2.2.13}$$

gdzie $\upsilon(x) = \sum_{v=1}^{V} \sum_{\substack{u=1 \\ u>v}}^{V} \sum_{i_1=1}^{I} \sum_{\substack{i_2=1 \\ i_1 \neq i_2}}^{I} x_{vi_1}^{\alpha} x_{ui_2}^{\alpha}$.

Wymaganie w zakresie decentralizacji przetwarzania można rozszerzyć na żądanie, aby co najmniej $\upsilon_{\min}$ par modułów przydzielono do różnych węzłów, przy czym $1 \leq \upsilon_{\min} \leq V(V\text{-}1)$. Powyższe można zapisać w postaci następującej nierówności:

$$\upsilon(x) \geq \upsilon_{\min}. \tag{2.2.14}$$

Ograniczenia mogą dotyczyć również wielkości pamięci operacyjnej, wielkości pamięci dyskowej, zagwarantowania minimalnego czasu pracy procesora, żądania odpowiedniej przepustowości kanałów komunikacyjnych, czy zapewnienia dostępu do niezbędnych urządzeń sieciowych lub zewnętrznych. Niech dostępne będą zasoby lokalne $z_1,...,z_r,...,z_R$, a $\tilde{d}_{jr}$ oznacza wielkość zasobu z_r w komputerze *j*-tego typu [JZ_r – jednostka miary r - tego zasobu]. Przyjmuje się, że zadanie m_v rezerwuje c_{vr} jednostek zasobu z_r [JZ_r], $c_{vr} \geq 0$.

Zakłada się także, że zapotrzebowanie na zasób jest stałe podczas realizacji zadań. Wymaga się, aby w hoście była dostępna wystarczająca ilość zasobów *addytywnych*, czyli takich, dla których łączne zapotrzebowanie przez pewien podzbiór modułów jest sumą zapotrzebowań przez poszczególne moduły. Powyższe zapisujemy następująco [334]:

$$\sum_{v=1}^{V} c_{vr} x_{vi}^{\alpha} \leq \sum_{j=1}^{J} \tilde{d}_{jr} x_{ij}^{\beta}, \quad i = \overline{1,I}, \ r = \overline{1,R}. \tag{2.2.15}$$

Zazwyczaj zwraca się uwagę na wystarczającą wielkość pamięci RAM oraz pamięci dyskowej. Eksperymenty, które przeprowadzono z serwerami aplikacji *GlassFish* i *JBoss* wykazały, że w wypadku przekroczenia limitu pamięci RAM, którą zarezerwowano dla aplikacji testowej, następuje zawieszenie aplikacji (*GlassFish*) lub obsługa wyjątku i żądanie zwiększenia pamięci (*JBoss*) [20]. Jeśli w funkcji celu uwzględnione są: obciążenie procesorów oraz obciążenie komunikacyjne, to szczególnie istotnymi zasobami pozostają: pamięć operacyjna (r=1) oraz pamięć dyskowa (r=2).

Niech zadane są następujące macierze:

$\tilde{D} = \left[\tilde{d}_{jr}\right]_{J\times 2}$ - macierz wielkości dostępnych zasobów pamięci w komputerach;

$C = \left[c_{vr}\right]_{V\times 2}$ - macierz zapotrzebowania na pamięć przez moduły.

Jeśli $\kappa_{ir}(x)$ oznacza rezerwę *r*-tego rodzaju pamięci w komputerze usytuowanym w *i*-tym węźle, to żądanie przydzielenia pamięci o odpowiedniej wielkości może być zapisane za pomocą 2*I* nierówności, jak niżej:

$$\kappa_{ir}(x) \geq 0, \quad i = \overline{1,I}, \ r = \overline{1,2}, \tag{2.2.16}$$

gdzie $\kappa_{ir}(x) = \sum_{j=1}^{J} \tilde{d}_{jr} x_{ij}^{\beta} - \sum_{v=1}^{V} c_{vr} x_{vi}^{\alpha}, \quad i = \overline{1,I}, \ r = \overline{1,2}.$

Ważnym ograniczeniem jest nieprzekroczenie limitu związanego ze zużyciem energii elektrycznej przez komputery gridu. Najbardziej oszczędne rozwiązania wśród najszybszych superkomputerów opierają się na heterogenicznych architekturach wykorzystujących GPU NVIDIA *Kepler K20x* oraz CPU *Intel Xeon E5-2620v2 6C 2,100 GHz* [404]. W *Tokyo Institute of Technology* skonstruowano najbardziej energooszczędny w 2014 roku superkomputer *TSUBAME-KFC* o wydajności obliczeniowej 152 [TFLOPS], który cechował się efektywnością energetyczną 4,390 [GFLOPS/W] [404]. Komputer cechuje się poborem mocy 34,580 [kW]. Zazwyczaj im mniej wydajne superkomputery, tym mniejsza ich efektywność energetyczna [404].

Zakłada się, że *łączne zużycie energii elektrycznej przez komputery gridu* $E(x)$ [W] nie powinno przekroczyć zadanego limitu poboru mocy $\varepsilon_{\max}$. W tym wypadku zadany jest wektor zużycia energii przez komputery $\varepsilon = [\varepsilon_1, ..., \varepsilon_j, ..., \varepsilon_J]$. Ograniczenie na moc poboru energii przez komputery gridu formułuje się, jak niżej:

$$E(x) \leq \varepsilon_{\max}, \tag{2.2.17}$$

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

Kolejne wymagania mogą odnosić się do ograniczeń nakładanych na kryteria oceny wykorzystania zasobów gridowych. Często przyjmuje się, że obciążenie newralgicznego komputera nie może przekroczyć zadanej maksymalnej wartości lub że koszt zakupu komputerów nie powinien być większy od założonego limitu. Rozważane powyżej ograniczenia mogą znacząco zredukować zbiór konfiguracji dopuszczalnych lub nawet wpłynąć na to, że zbiór ten będzie zbiorem pustym.

2.3. Kryteria do oceny jakości konfiguracji gridów

Jeżeli przydział modułów programistycznych do hostów minimalizuje łączne obciążenie gridu, to może wystąpić sytuacja asymetrycznego skupienia modułów w wybranym węźle obliczeniowym. Ten newralgiczny węzeł w odróżnieniu od pozostałych będzie zatem najbardziej obciążony. Dlatego też można minimalizować $Z_{\max}$ - obciążenie newralgicznego węzła. W konsekwencji równoważone jest obciążenie procesorów w gridzie, gdyż zadania przesuwane z hosta newralgicznego są przydzielane do mniej obciążonych komputerów.

Z drugiej strony minimalizacja $Z_{\max}$ może spowodować, że wzrośnie Z_{suma} - łączne obciążenie wszystkich hostów. Występuje zatem konflikt między zmniejszaniem obciążenia newralgicznego komputera w systemie a redukcją łącznego obciążenia gridu. Administrator gridu, dysponując odpowiednim narzędziem, mógłby wybierać między minimalizacją jednego lub drugiego kryterium. Możliwe byłoby także wyznaczenie konfiguracji *Pareto*-optymalnej, leksykograficznej lub kompromisowej. Taką rolę spełnia wytworzona na potrzeby rozprawy aplikacja ADAIORG'14.

Niech $\hat{Z}_i(x)$ oznacza obciążenie procesorów hosta przydzielonego do *i*-tego węzła dla konfiguracji *x*. Wartość $\hat{Z}_i(x)$ wyznacza się w następujący sposób:

$$\hat{Z}_i(x) = \sum_{j=1}^{J} \sum_{v=1}^{V} t_{vj}\, x_{vi}^{\alpha} x_{ij}^{\beta}, \qquad i = \overline{1,I} \ . \tag{2.3.1}$$

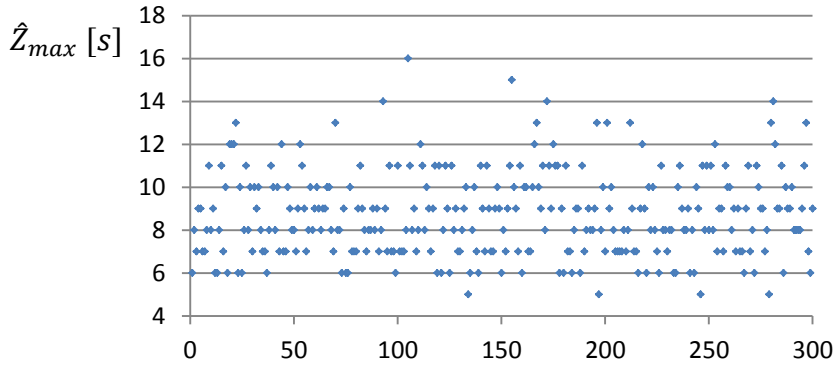
Procesory hostów usytuowane w różnych węzłach mogą być obciążone w różnym stopniu pod względem $\hat{Z}_i(x)$. Obciążenie newralgicznego hosta pod względem obciążenia procesorów wyznaczamy za pomocą następującej zależności:

$$\hat{Z}_{max}(x) = \max_{i=\overline{1,I}} \hat{Z}_i(x). \tag{2.3.2}$$

Na rysunku 14 przedstawiono zobrazowanie wartości obciążenia procesorów newralgicznego hosta dla wybranych konfiguracji gridu. Wartości $\hat{Z}_{\max}$ wyznaczono na danych modelowych w systemie sześciowęzłowym (I=6) dla 300 losowo wygenerowanych przydziałów modułów do komputerów. Rozpatruje się dwanaście modułów (V=12), a macierz skumulowanych czasów przetwarzania modułów dla trzech rodzajów komputerów (J=3) ma postać, jak niżej:

$$\boldsymbol{T} = \begin{bmatrix} 1 & 3 & 2 & 4 & 3 & 1 & 2 & 1 & 2 & 3 & 1 & 3 \\ 2 & 2 & 2 & 3 & 2 & 2 & 3 & 2 & 1 & 4 & 2 & 2 \\ 3 & 1 & 3 & 3 & 3 & 1 & 2 & 1 & 2 & 2 & 2 & 1 \end{bmatrix}^{\mathrm{T}} \text{ [s]}. \tag{2.3.3}$$

Najmniejszą wartość $\hat{Z}_{max}$=5 [s] wyznaczono dla czterech konfiguracji, w tym rozwiązania nr 278 (X^{α}=[6,5,5,3,1,2,3,4,1,2,4,6]^T, X^{β}=[1,3,3,1,1,1]^T). Natomiast Z_{suma} dla tej konfiguracji wynosi 982 [s]. Największa wartość obciążenia procesorów newralgicznego hosta to 16 [s].



Rys. 14. Wartości ważonego obciążenia Δ_{max} dla trzystu losowo wygenerowanych konfiguracji, przy czym V=12, I=6, J=3, $\rho_1=1$ $\rho_2=0$.
Źródło: opracowanie własne.

Istotnym obciążeniem *i*-tego węzła jest wysyłanie i odbiór danych. Skumulowany czas $\tilde{Z}_i(x)$ nadawania i odbioru danych w *i*-tym węźle wyznacza się następująco:

$$\tilde{Z}_i(x) = \sum_{v=1}^{V} \sum_{\substack{u=1 \\ u \neq v}}^{V} \sum_{i=1}^{I} \sum_{\substack{k=1 \\ i \neq k}}^{I} \tau_{vuik}\, x_{vi}^{\alpha} x_{uk}^{\beta}, \tag{2.3.4}$$

gdzie τ_{vuik} – skumulowany czas wysyłania i odbierania danych między modułem nr *v* usytuowanym w węźle nr *i* a modułem nr *u* usytuowanym w węźle *k* [s].

W niektórych modelach w literaturze przedmiotu przyjmuje się, że czas komunikacji między modułami nie zależy od miejsca ich alokacji [315]. Takie założenie jest słuszne w systemach zamkniętych, w których przepustowości wirtualnych połączeń komunikacyjnych między parami węzłów są identyczne.

W systemach otwartych, które cechują się zazwyczaj dużą liczbą węzłów, czas komunikacji między modułami zależy od miejsca alokacji modułów. Wraz ze wzrostem liczby węzłów tranzytowych rośnie opóźnienie przesyłanego komunikatu między modułami. Ponadto, istotny wpływ na wielkość opóźnienia wywierają przepustowości kanałów komunikacyjnych. W konsekwencji przy tej samej wielkości przesyłanych danych, czas transmisji komunikatu zależy od pary węzłów, do których przydzielono komunikujące się moduły.

Macierz komunikacji między dwunastoma modułami, które mogą być skonfigurowane w sześciu węzłach obliczeniowych jest macierzą czterowymiarową

$\tau = [\tau_{vuik}]_{12x12x6x6}$. Jednakże na potrzeby zobrazowania wartości tej macierzy przyjmuje się, że czas komunikacji nie zależy od numerów węzłów. Wówczas można rozważać macierz kwadratową $\tau = [\tau_{vu}]_{12x12}$ o przykładowych wartościach, jak niżej:

$$\tau = \begin{bmatrix} 0 & 1 & 2 & 1 & 4 & 3 & 2 & 3 & 2 & 4 & 1 & 2 \\ 3 & 0 & 1 & 2 & 3 & 4 & 0 & 4 & 4 & 4 & 1 & 5 \\ 1 & 4 & 0 & 1 & 1 & 0 & 1 & 2 & 2 & 1 & 5 & 5 \\ 7 & 2 & 8 & 0 & 7 & 1 & 7 & 4 & 7 & 7 & 2 & 5 \\ 8 & 0 & 8 & 8 & 0 & 8 & 1 & 5 & 2 & 5 & 6 & 8 \\ 4 & 8 & 8 & 2 & 8 & 0 & 3 & 2 & 7 & 4 & 8 & 2 \\ 8 & 4 & 6 & 0 & 5 & 5 & 0 & 7 & 2 & 7 & 8 & 1 \\ 7 & 1 & 2 & 1 & 4 & 7 & 7 & 0 & 9 & 9 & 4 & 1 \\ 2 & 1 & 0 & 5 & 1 & 1 & 4 & 9 & 0 & 9 & 1 & 4 \\ 7 & 0 & 1 & 9 & 0 & 2 & 9 & 1 & 7 & 0 & 9 & 1 \\ 4 & 7 & 4 & 1 & 5 & 9 & 0 & 9 & 0 & 9 & 0 & 1 \\ 8 & 1 & 7 & 0 & 1 & 0 & 9 & 1 & 9 & 4 & 1 & 0 \end{bmatrix} \text{ [s].} \quad (2.3.5)$$

Warto podkreślić, że macierze $\boldsymbol{T}$ i $\boldsymbol{\tau}$, które określono za pomocą zależności (2.3.3) i (2.3.5), wchodzą w skład benchmarku danych nr 1, który umożliwia analizę instancji zagadnień optymalizacji sformułowanych w rozprawie. Na rysunku 15 przedstawiono wartości obciążenia komunikacyjnego newralgicznego hosta, które to obciążenie wyznaczono za pomocą zależności:

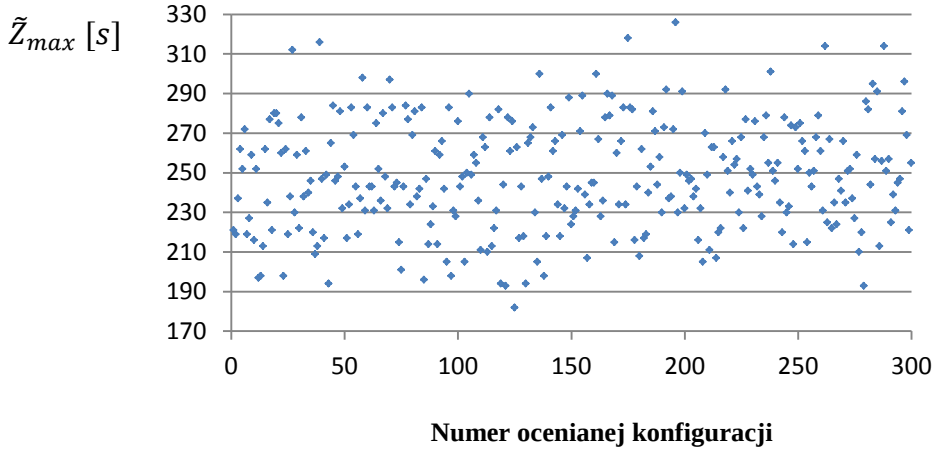
$$\tilde{Z}_{max}(x) = \max_{i=\overline{1,I}} \tilde{Z}_i(x). \quad (2.3.6)$$

Minimalna wartość $\tilde{Z}_{\max}$ wynosi 182 [s] dla konfiguracji nr 124 (X^{α}=[3,6,3,5,1,1,6,4,2,4,2,3]^T; X^{β} =[1,2,1,2,3,2]^T). Ponadto $\hat{Z}_{max} = 6\ [s]$ oraz Z_{suma}=961 [s]. Natomiast wartość maksymalna $\tilde{Z}_{\max}$ to 326 [s]. Warto zauważyć, że wybór niewłaściwej konfiguracji może spowodować znaczący wzrost obciążenia komunikacyjnego.

Im większa liczba węzłów obliczeniowych, modułów programistycznych oraz rodzajów możliwych hostów, tym większych różnic między wartością minimalną a wartością maksymalną obciążeń komunikacyjnych można oczekiwać. Podobne spostrzeżenie dotyczy obciążenia procesorów. Powyższe obrazuje jak ważna jest optymalizacja konfiguracji gridu umożliwiająca efektywne zarządzanie jego zasobami.

Porównując oceny najlepszych konfiguracji z benchmarku nr 1 pod względem obu obciążeń, warto zauważyć, że wariant nr 278 może być preferowany ze względu na $\hat{Z}_{max}$, gdyż obciążenie to jest mniejsze o 1 [s] od obciążenia $\hat{Z}_{max}$ dla wariantu nr 124. Natomiast alternatywa nr 124 powinna być preferowana ze względu na $\tilde{Z}_{max}$, które jest mniejsze o 11 [s]. Niestety, nie można jednoznacznie przesądzić, który wariant

powinien być preferowany ze względu na oba kryteria jednocześnie. Sumowanie obu obciążeń obarczone jest błędem, gdyż może dotyczyć obciążenia procesorów oraz obciążenia komunikacyjnego dla różnych newralgicznych hostów.



Rys. 15. Wartości obciążeń $\tilde{Z}_{\max}$ dla benchmarku nr 1
Źródło: opracowanie własne.

Jednym z podejść, które można zastosować do rozstrzygnięcia powyższego dylematu jest zastosowanie dodatkowego kryterium, np.: łącznego obciążenia Z_{suma}, kosztu zakupu komputerów lub dostępności gridu, za pomocą którego to kryterium można wybrać jeden wariant do rekonfiguracji gridu.

Jeszcze innym podejściem może być zastosowanie optymalizacji wielokryterialnej i wybór wariantu *Pareto*-optymalnego, kompromisowego lub też leksykograficznego [330]. Na rysunku 16 przedstawiono oceny obciążeń jako uporządkowane dwójki $(\hat{Z}_{max}, \tilde{Z}_{max})$ w odniesieniu do konfiguracji z benchmarku nr 1. Ocenę obciążeń wariantu nr 278 oznaczono jako P_1, a ocenę konfiguracji nr 124 - jako P_2. Warto podkreślić, że kilka konfiguracji może cechować się identyczną oceną.

W literaturze przedmiotu rozpatruje się także łączne obciążeniem *i*-tego węzła $Z_i^+(x)$ dla konfiguracji *x*, które to obciążenie wyznacza się w następujący sposób [33]:

$$Z_i^+(x) = \hat{Z}_i(x) + \tilde{Z}_i(x). \tag{2.3.7}$$

Maksymalne obciążenie węzła dla konfiguracji *x* wyraża się następującą formułą:

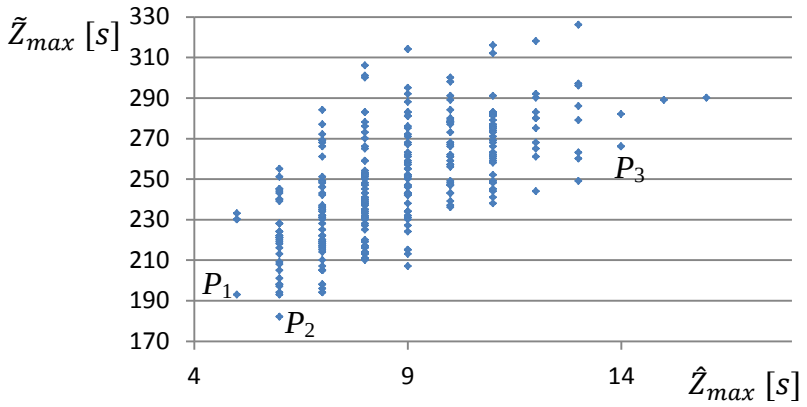
$$Z_{max}(x) = \max_{i\in\overline{1,I}}\{Z_i^+(x)\}. \tag{2.3.8}$$

Warto zauważyć, że nie zawsze $Z_{max}(x) = \hat{Z}_{max}(x) + \tilde{Z}_{max}(x)$. Powyższa systuacja wystąpi, jeśli ten sam węzeł będzie najbardziej obciążony pod względem przetwarzania danych oraz pod względem obciążenia komunikacyjnego. Można natomiast pokazać, że $Z_{max}(x) \leq \hat{Z}_{max}(x) + \tilde{Z}_{max}(x)$ dla $x \in \boldsymbol{X}$. Ponadto istotnym jest fakt, że postulat równoważenia obciążeń może być uwzględniony nie tylko za

pomocą minimalizacji funkcji celu Z_{max}, ale także za pomocą wprowadzenia następującego ograniczenia:

$$Z_{max} \leq Z_{gr}, \tag{2.3.9}$$

gdzie Z_{gr} reprezentuje zadane największe dopuszczalne obciążenie newralgicznego hosta.

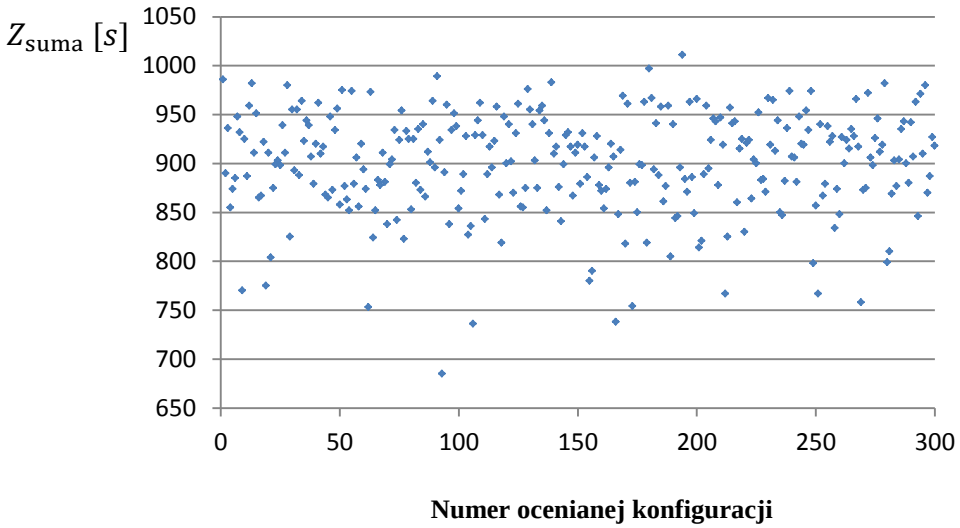


Rys. 16. Oceny obciążeń jako uporządkowane dwójki ($\hat{Z}_{max}$, $\tilde{Z}_{max}$) w odniesieniu do sześciokomputerowego gridu dla trzystu konfiguracji
Źródło: opracowanie własne.

Analogiczne ograniczenia można nałożyć na wartości $\hat{Z}_{max}$ czy $\tilde{Z}_{max}$. Reasumując ten fragment dyskusji, administrator gridu dysponuje czteroma kryteriami do oceny obciążenia systemu: $Z_{\max}, \hat{Z}_{\max}, \tilde{Z}_{\max}$, a także Z_{suma} - łączne obciążenie wszystkich węzłów, które można wyznaczyć, jak niżej:

$$Z_{suma}(x) = \sum_{i=1}^{I} Z_i^+(x). \tag{2.3.10}$$

Na rysunku 17 przedstawiono wartości Z_{suma} w odniesieniu do sześcio-komputerowego gridu dla trzystu losowo wybranych konfiguracji. Najmniejsza wartością 685 [s] cechuje się konfiguracja nr 92 (X^{α}=[5,5,5,2,6,6,5,6,1,5,5,5]^T; X^{β}=[1,1,3,2,3,1]^T). Warto zwrócić uwagę, że największe obciążenie procesorów dla tej konfiguracji wynosi aż 14 [s], a obciążenie komunikacyjne – 266 [s]. Ocenę konfiguracji nr 92 reprezentuje punkt P_3 na rysunku 16. Można zatem ocenić tę konfigurację jako mało interesującą pod względem obciążenia procesorów newralgicznego hosta i obciążenia komunikacyjnego newralgicznego komputera. Z kolei konfiguracja nr 191 cechuje się największym sumarycznym obciążeniem o wartości 1 011 [s].



Rys. 17. Wartości łącznego obciążenia Z_{suma} w systemie sześciokomputerowym dla trzystu losowo wybranych konfiguracji
Źródło: opracowanie własne.

Często stosowanym sposobem porównania ocen P_1 i P_2 z rysunku 16 jest wprowadzenie wag do obu miar obciążenia [300]. Z tego powodu wprowadza się kryterium $\Delta_{\max}$ - ważone obciążenie newralgicznego komputera, które to obciążenie dla zadanej konfiguracji *x* wyznacza się następująco:

$$\Delta_{\max}(x) = \rho_1 \hat{Z}_{\max}(x) + \rho_2 \tilde{Z}_{\max}(x), \tag{2.3.11}$$

gdzie:

ρ_1 – zadana waga dla obciążenia procesorów, $0 \le \rho_1 \le 1$;

ρ_2 – zadana waga dla obciążenia komunikacyjnego, $0 \le \rho_2 \le 1$, $\rho_1 + \rho_2 = 1$.

W niektórych modelach gridów przyjmuje się, że koszt transmisji jest proporcjonalny do czasu transmisji. Wówczas dostaje się następujące wyrażenie na koszt wykonania programu rozproszonego [313]:

$$K(x) = c_1 \sum_{i=1}^{I} \hat{Z}_i(x) + c_2 \sum_{i=1}^{I} \tilde{Z}_i(x), \tag{2.3.12}$$

gdzie:

c_1 – koszt przetwarzania danych w węźle w jednostce czasu [JM/s, przy czym JM – jednostka monetarna];

c_2 – koszt przesyłania danych między węzłami w jednostce czasu [JM/s].

Kolejnym kryterium oceny gridu jest *czas wykonania aplikacji*, który można wyznaczyć za pomocą symulacji lub zmierzyć dla wybranych konfiguracji [297].

Należy zauważyć, że optymalna konfiguracja pozwala na znacząco krótsze wykonanie aplikacji niż dla przypadku z „najgorszym" przydziałem [33].

Ponadto stosowane mogą być jeszcze alternatywne kryteria, takie jak dostępność gridu czy prawdopodobieństwo ukończenia zadań w terminach [366]. Jeśli w wyniku braku zasilania energetycznego nastąpi wyłączenie hosta, to może być konieczna migracja modułów z wyłączonego komputera na inne hosty. W konsekwencji może się zwiększyć obciążenie niektórych z nich.

Niech niezależne zdarzenia powodujące niedostępność hosta *j*-tego rodzaju zachodzą z intensywnością γ_j^{-1} [s] zgodnie z rozkładem wykładniczym [366]. Wówczas dostępność hosta maleje w sposób wykładniczy w czasie, a dostępność Γ całego gridu o konfiguracji *x* wyznacza się w następujący sposób [366]:

$$\Gamma(\gamma, T, x) = \prod_{v=1}^{V} \prod_{i=1}^{I} \prod_{j=1}^{J} \exp(-\gamma_j t_{vj} x_{vi}^{\alpha} x_{ij}^{\beta}), \tag{2.3.13}$$

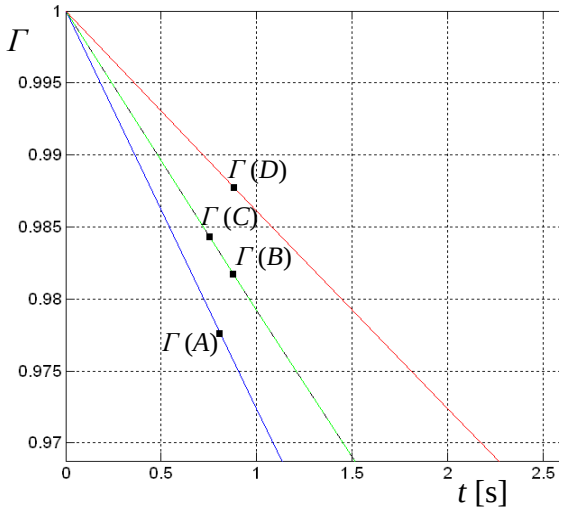
gdzie $\gamma = [\gamma_1, ..., \gamma_j, ..., \gamma_J]$ - wektor parametrów rozkładów wykładniczych dostępności hostów poszczególnych klas.

Z zależności (2.3.13) wynika, że dostępność gridu jest liczbą rzeczywistą z zakresu [0; 1]. Można także pokazać, że dla zadanej konfiguracji *x* dostępność gridu maleje szybciej w czasie niż dostępność każdego z hostów [366]. Niech zadane wartości parametrów γ_j są niezmienne podczas eksploatacji gridu. Dostępność gridu zależy od elementów macierzy $\boldsymbol{T}$ dla zadanych parametrów γ oraz zadanej konfiguracji *x*.

Na rysunku 18 przedstawiono wartości dostępności Γ dla czterech wybranych konfiguracji *A, B, C* i *D* z czternastoma modułami. Im niższe wartości parametrów γ oraz krótsze czasy realizacji zadań, tym wartości dostępności są wyższe [366].

Istotną grupę kryteriów stanowią rezerwy wielkości zasobów krytycznych w gridzie. Jeśli konfiguracja spełnia żądanie przydzielenia zasobów o odpowiedniej wielkości, to można maksymalizować rezerwy krytycznych zasobów w gridzie po to, aby wykorzystać je podczas realizacji innych zadań. Warto podkreślić, że sumaryczna rezerwa pamięci RAM (*r*=1) lub pamięci dyskowej (*r*=2) w gridzie jest stała i nie zależy od rozmieszczenia modułów, ale od przydziału rodzajów komputerów. Łączną rezerwę *r*-tego zasobu wyznacza się za pomocą następującej zależności:

$$M_r(x) = \sum_{i=1}^{I} \kappa_{ir}(x), \quad \text{dla } x \in \boldsymbol{X},\ r = \overline{1,2}. \tag{2.3.14}$$



Rys. 18. Wartości dostępności Γ dla wybranych konfiguracji [366]

Natomiast od rozmieszczenia modułów i rodzaju przydzielonych komputerów zależy rezerwa wielkości pamięci w newralgicznym hoście, którą wyznaczamy następująco:

$$M_r^{\min}(x) = \min_{i=\overline{1,I}} \kappa_{\mathrm{ir}}(x), \quad \text{dla } x \in \boldsymbol{X},\ r = \overline{1,2}. \tag{2.3.15}$$

W pewnych sytuacjach decyzyjnych minimalizuje się $E(x)$ - łączną moc poboru energii elektrycznej przez komputery gridu, która to moc może także być uwzględniona w ograniczeniu (2.2.17). Pozostałe kryteria mogą być także rozpatrywane w ograniczeniach, co znacząco zwiększa liczbę możliwych sytuacji decyzyjnych.

2.4. Wnioski i uwagi

Do oceny konfiguracji gridów w rozprawie zaproponowano wykorzystanie kryteriów związanych z obciążeniem newralgicznego komputera, takich jak: $\hat{Z}_{\max}$ - obciążenie procesorów w newralgicznym hoście oraz $\tilde{Z}_{\max}$ - obciążenie komunikacyjne w newralgicznym serwerze. Aby uniknąć sumowania obciążeń $\hat{Z}_{\max}$ oraz $\tilde{Z}_{\max}$, wprowadzono $\Delta_{\max}$ - ważone obciążenie newralgicznego hosta. Minimalizacja powyższych kryteriów prowadzi do równoważenia obciążeń w węzłach. Warto także wspomnieć o alternatywnych kryteriach: $Z_{\max}$ - łącznym obciążeniu newralgicznego hosta, Z_{suma} - łącznym obciążeniu gridu, a także K - koszcie realizacji modułów w gridzie.

Oprócz kryteriów związanych z obciążeniem w systemie gridowym, optymalizacja konfiguracji może być realizowana zgodnie z następującymi kryteriami:

- minimalizacja kosztów zakupu hostów;
- maksymalizacja wydajności systemu gridowego;
- maksymalizacja stopnia decentralizacji przetwarzania;
- maksymalizacja dostępności gridu;
- maksymalizacja prawdopodobieństwa realizacji zadań w terminach;
- maksymalizacja rezerwy wielkości pamięci w newralgicznym hoście;
- minimalizacja zużycia energii elektrycznej.

Podstawowe wymagania ilościowe mogą być modelowane jako ograniczenia nierównościowe nałożone ww. kryteria. Ponadto mogą być stosowane ograniczenia związane z łączną rezerwą pamięci RAM i pamięci dyskowej. W modelu uwzględniono ograniczenia formalne dotyczące: usytuowania dowolnego hosta w każdym węźle oraz przydzielenia każdego modułu do jednego z węzłów. Ograniczenia formalne są spełnione za pomocą kodowania całkowitoliczbowego rozwiązań, co znacząco redukuje przestrzeń przeszukiwań.

Ograniczenie wydajnościowe, limity finansowe, wysoki stopień decentralizacji, niedobór zasobów gridowych czy zbyt duże zużycie energii mogą spowodować, że zbiór konfiguracji dopuszczalnych będzie zbiorem pustym. Natomiast jeśli $V \geq 2$ oraz $J \geq 1$, to podstawowy zbiór rozwiązań dopuszczalnych określony za pomocą zależności (2.2.9), nie jest zbiorem pustym i zawiera $I^V J^I$ przydziałów modułów do komputerów.

3. SFORMUŁOWANIE PROBLEMÓW OPTYMALIZACJI WYKORZYSTANIA ZASOBÓW GRIDOWYCH

Konfiguracje gridów komputerowych są niezwykle złożone, a ponadto ich ocena może być dokonana za pomocą wielu kryteriów. Warto podkreślić, że konfiguracje dopuszczalne powinny spełniać różnorodne ograniczenia. W tej sytuacji zdecydowano się na sformułowanie trzech grup oryginalnych zagadnień optymalizacji wykorzystania zasobów w systemach gridowych.

Pierwszą grupę rozpatrywanych problemów stanowią zadania optymalizacji jednokryterialnej. Druga grupa zagadnień cechuje się poszukiwaniem konfiguracji optymalnych w sensie *Pareto*. Natomiast trzecia grupa dylematów odnosi się do wyznaczania rozwiązań kompromisowych.

Równoważenie obciążeń procesorów oraz obciążeń komunikacyjnych jest istotnym sposobem umożliwiającym zwiększanie wydajności gridu. Ponadto zapewnia efektywne wykorzystanie jego zasobów, takich jak: czas pracy procesorów, przepustowość układów we/wy, wielkość pamięci RAM czy pojemność pamięci dyskowych. Nie można także pominąć aspektów finansowych ze względu na wysokie koszty eksploatacji czy koszty inwestycyjne związane z zakupem serwerów. Z powyższych powodów istotne jest sformułowanie adekwatnych zagadnień optymalizacji i opracowanie metod optymalnego wykorzystania zasobów gridowych.

3.1. Równoważenie obciążeń w gridach

Równoważenie obciążeń w gridach może być osiągnięte za pomocą minimalizacji kryterium minimalizującego obciążenie przetwarzania lub komunikacji danych. W literaturze przedmiotu sformułowano zagadnienia optymalizacji systemów rozproszonych w oparciu o minimalizację łącznego obciążenia gridu Z_{suma} [5], a także K - kosztu realizacji modułów w systemie [326]. Rozpatruje się także minimalizację $Z_{\max}$ - łącznego obciążenia newralgicznego komputera [33].

Do oceny konfiguracji gridów w rozprawie zaproponowano wykorzystanie nowych kryteriów związanych z obciążeniem krytycznej infrastruktury, takich jak: $\hat{Z}_{\max}$ - obciążenie CPU w newralgicznym komputerze, a także $\tilde{Z}_{\max}$ - obciążenie komunikacyjne najbardziej obciążonego hosta. Agregacja obu obciążeń $\hat{Z}_{\max}$ oraz $\tilde{Z}_{max}$

zachodzi w $\Delta_{\max}$ - ważonym obciążeniu newralgicznego hosta. Poniżej sformułowano podstawowe zagadnienie minimalizacji $\Delta_{\max}$.

Dla danych wejściowych: ρ_1, ρ_2, V, I, J, $T = \left[t_{vj}\right]_{V\times J}$, $\tau = [\tau_{vuik}]_{V\times V\times I\times I}$

należy wyznaczyć x^*, takie że:

$$\Delta_{max}(x^*) = \min_{x\in \boldsymbol{X}} \Delta_{max}(x), \tag{3.1.1}$$

gdzie:

$\Delta_{\max}(x) = \rho_1 \hat{Z}_{\max}(x) + \rho_2 \tilde{Z}_{\max}(x)$;

$x = \left[x_{11}^{\alpha}, \ldots, x_{vi}^{\alpha}, \ldots, x_{VI}^{\alpha}, x_{11}^{\beta}, \ldots, x_{ij}^{\beta}, \ldots, x_{IJ}^{\beta}\right]$;

$$\boldsymbol{X} = \left\{ x \in \boldsymbol{B_{bin}}^{I(V+J)} \middle| \sum_{i=1}^{I} x_{vi}^{\alpha} = 1 \text{ dla } v = \overline{1,V};\ \sum_{j=1}^{J} x_{ij}^{\beta} = 1 \text{ dla } i = \overline{1,I} \right\}$$

Zagadnienie (3.1.1) jest jednokryterialnym zagadnieniem optymalizacji zero-jedynkowej, dla którego liczebność przestrzeni możliwych wariantów wynosi $2^{I(V+J)}$. Można pokazać, że jeśli $V \geq 1$, $J \geq 1$ oraz $I \geq 1$, to istnieje co najmniej jedna konfiguracja dopuszczalna. Co więcej, stosując kodowanie całkowitoliczbowe rozwiązań, uzyskuje się konfigurację dopuszczalną. Zbiór rozwiązań dopuszczalnych $\boldsymbol{X}$ zawiera $I^V J^I$ konfiguracji, co jest znaczącą redukcją w odniesieniu do wartości $2^{I(V+J)}$.

Ponieważ problem minimalizacji liniowej funkcji kosztów wykonania aplikacji rozproszonej o dowolnej strukturze interakcji między modułami, dla co najmniej czterech komputerów, należy do klasy problemów NP-trudnych, nawet przy braku ograniczeń wynikających z limitowanych wielkości zasobów [46, 314], to zagadnienie (3.1.1) należy także do klasy zagadnień NP-trudnych, przy czym funkcja celu w tym wypadku jest nieliniowa.

W szczególności można pokazać, że problem decyzyjny wywodzący się z zagadnienia (3.1.1) jest równoważny NP-zupełnemu problemowi plecakowemu, a algorytm transformacji problemów jest wielomianowy [292]. W zagadnieniu decyzyjnym bada się, czy istnieje konfiguracja dopuszczalna x^*, taka że $\Delta_{\max} \leq \Delta_{gr}$?

Niewątpliwie problem należy do klasy zagadnień NP, gdyż nie są znane efektywne algorytmy jego rozwiązania. W szczególności nie można zastosować efektywnego algorytmy wyznaczania maksymalnego przepływu w grafie, który *Stone* opracował do optymalizacji przydziałów w dwóch komputerach [326].

Niech $\Delta_i(x) = \rho_1 \hat{Z}_i(x) + \rho_2 \tilde{Z}_i(x)$ dla $i = \overline{1,I}$ reprezentuje ważone obciążenie w węźle nr *i*. Dla konfiguracji X^{α}=[2,5,1,5,6,1,4,4,3,6,2,2]^T, X^{β}=[3,1,3,3,2,3]^T, macierzy $\boldsymbol{T}$ w postaci (2.3.3), macierzy $\boldsymbol{\tau}$ w postaci (2.3.5) oraz dla $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$, wektor obciążeń w sześciu węzłach jest następujący Δ=[77; 113,5; 45; 101; 72; 85]^T [s]. Zatem dla tej konfiguracji $\Delta_{\max}(x) = 113{,}5$ [s].

Bez straty ogólności rozważań przyjmuje się, że przydział komputerów do węzłów jest zadany, np. X^{β}=[3,1,3,3,2,3]^T. Wówczas poszukuje się jedynie przydziału zadań do węzłów, których jest V^I. Ponadto przyjmuje się, że czasy komunikacji w macierzy $\boldsymbol{\tau}$ wynoszą zero.

Niech skumulowane czasy wykonania modułów w komputerach t_{vj} odpowiadają wagom w_v $v = \overline{1,V}$ przedmiotów w decyzyjnym problemie plecakowym. Każdy przedmiot cechuje się wagą w_v dla $v = \overline{1,V}$ oraz ceną c_v=$T_{\max}$-w_v dla $v = \overline{1,V}$, przy czym $T_{\max}$ jest maksymalną wagą w_v dla $v = \overline{1,V}$.

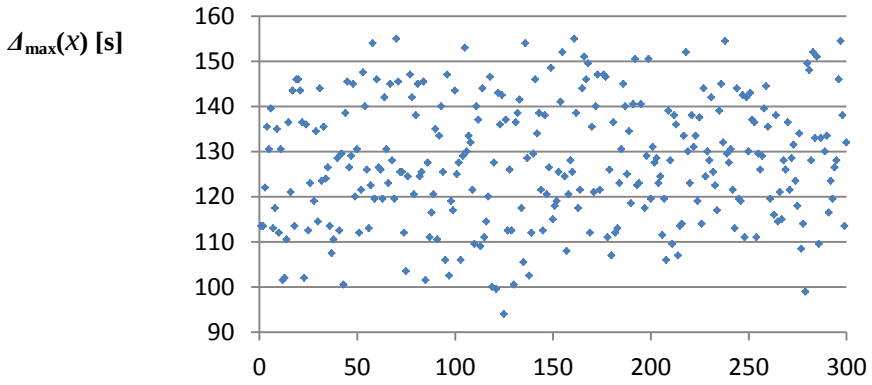
W wypadku newralgicznego węzła problem polega na spakowaniu plecaku, w którym waga przedmiotów nie przekroczy wartości Δ_{gr}. W problemie maksymalizowany jest koszt przedmiotów w plecaku. Jednakże ze względu na formułę c_v=$T_{\max}$-w_v dla $v = \overline{1,V}$ w istocie minimalizuje się wagę przedmiotów w plecaku.

Zatem dopuszczalna konfiguracja odpowiada podzbiorowi przedmiotów, których łączna waga nie przekracza założonej wielkości plecaka. Natomiast przydział modułów w newralgicznym komputerze odpowiada rozwiązaniu o największej wartości w problemie plecakowym. Ponieważ decyzyjny problem plecakowy jest NP-zupełny [43], to także decyzyjne zagadnienie wywodzące się z problemu (3.1.1) jest NP-zupełne. Zatem problem optymalizacji (3.1.1) jest NP-trudny [43].

Zazwyczaj zachodzi sytuacja, w której newralgiczny host pod względem $\hat{Z}_{\max}$ nie będzie najbardziej obciążony pod względem $\tilde{Z}_{\max}$ i odwrotnie. Wagi ρ_1, ρ_2 odzwierciedlają preferencje decydentów (właściciela, projektanta lub administratora gridu) w odniesieniu do priorytetów związanych z obciążeniami $\hat{Z}_{\max}$ oraz $\tilde{Z}_{\max}$. Mogą być także narzędziem skalowania wartości obu obciążeń.

Dla wag $\rho_1 = 0{,}5;\ \rho_2 = 0{,}5$ żadne z obciążeń nie jest preferowane. Natomiast, jeśli $\rho_1 = 1,\ \rho_2 = 0$, to preferowane jest kryterium $\hat{Z}_{max}$ i zachodzi minimalizacja zgodnie z tym kryterium. W wypadku $\rho_1 = 0,\ \rho_2 = 1$ mamy do czynienia z minimalizacją $\tilde{Z}_{max}$.

Na rysunku 19 przedstawiono wartości ważonego obciążenia Δ_{max} dla trzystu losowo wygenerowanych konfiguracji spośród $1{,}59*10^{12}$ rozwiązań dopuszczalnych dla instancji, w której V=12, I=6, J=3. Natomiast binarna przestrzeń przeszukiwań składa się z $1{,}24*10^{27}$ rozwiązań dopuszczalnych i niedopuszczalnych. Najlepszym rozwiązaniem pod względem $\Delta_{\max}$ (dla $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$) jest konfiguracja nr 124 (dla X^{α}=[3,6,3,5,1,1,6,4,2,4,2,3]^T. X^{β}=[1,2,1,2,3,2]^T) cechująca się $\Delta_{\max}$=94 [s].

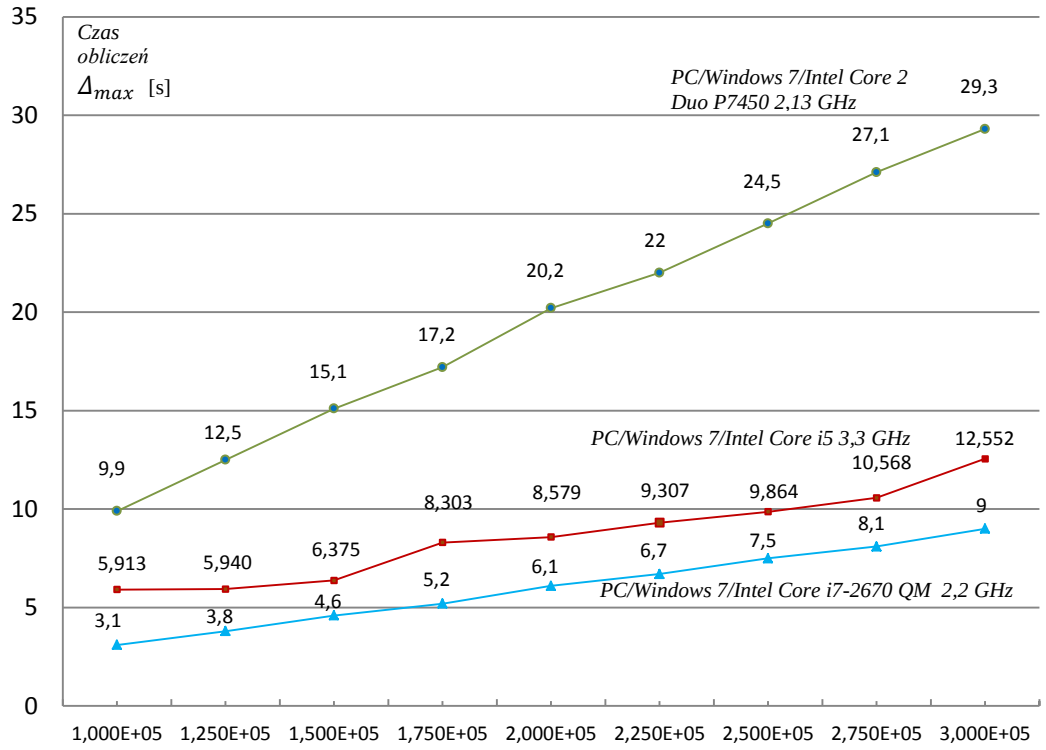


Rys. 19. Wartości ważonego obciążenia Δ_{max} dla trzystu losowo wygenerowanych konfiguracji, przy czym V=12, I=6, J=3, $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$.
Źródło: opracowanie własne.

Konfiguracja nr 124 okazała się także najlepsza pod względem $\hat{Z}_{max}$ dla wygenerowanego zbioru rozwiązań. Warto podkreślić, że najmniejsza $\Delta_{max} = 94$ [s] znacząco jest większa od minimum globalnego wynoszącego 14 [s].

Dla dużych instancji problemu (3.1.1) metody pełnego przeglądu nie można stosować ze względu na bardzo intensywny wzrost liczebności $I^V J^I$ wraz ze wzrostem I, V oraz J. Wygenerowanie konfiguracji, wyznaczenie wartości obciążeń oraz wyświetlenie wyników dla 10^5 konfiguracji za pomocą opracowanej aplikacji w języku *Java* zajmuje 3,1 [s] na komputerze klasy *PC/Windows 7/Intel Core i7-2670 QM 2,2 GHz*. Na rysunku 20 przedstawiono czasy obliczeń w zależności od liczby konfiguracji dla trzech rodzajów komputerów klasy PC.

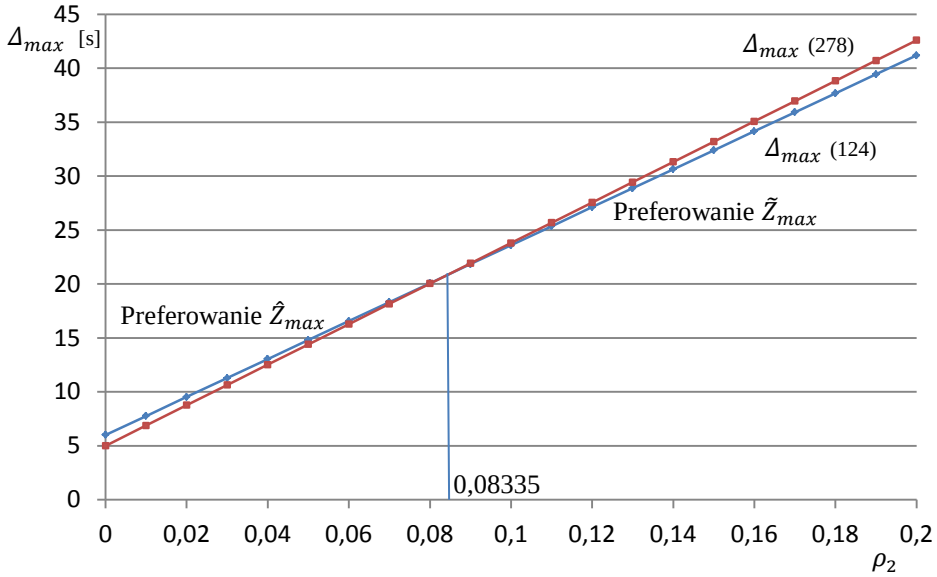
Metoda pełnego przeglądu w praktyce może być zastosowana dla przestrzeni przeszukiwań o liczebności co najwyżej 10^9 elementów (ok. 9 godz. obliczeń). W konsekwencji dla instancji cechujących się parametrami V=6, J=5, I=6 możliwe jest zastosowanie metody pełnego przeglądu. Natomiast większe wartości powyższych parametrów skutkują znaczącym zwiększeniem liczebności przestrzeni przeszukiwań w stosunku do 10^9.



Rys. 20. Czas wygenerowania, obliczenia obciążeń i wyznaczenia minimum dla zadanej liczby konfiguracji gridu
Źródło: opracowanie własne.

Metoda losowego generowania zadanej liczby konfiguracji może być zastosowana do wyznaczania rozwiązania startowego, które jest wykorzystywane przez metaheurystyki. Generując losowo 3×10^5 konfiguracji w ciągu niespełna 10 [s] na PC, można wyznaczyć rozwiązania relatywnie dobrej jakości dla instancji o niewielkich rozmiarach. Jakość tak wyznaczonych konfiguracji znacząco pogarsza się dla instancji o bardzo dużych wartościach *V*, *J*, *I*. Dla tych instancji na podstawie analizy literatury przedmiotu oraz własnych eksperymentów numerycznych rekomenduje się wykorzystanie algorytmu harmonicznego, w którym za pomocą algorytmu losowego wyznacza się początkową pamięć harmoniczną [294].

Na podstawie obciążeń $\hat{Z}_{max}$ i $\tilde{Z}_{max}$ obliczonych dla benchmarku nr 1, stwierdzono, że dla $\rho_1 = 1,\ \rho_2 = 0$ konfiguracja nr 278 cechuje się najmniejszą $\Delta_{\max}=\hat{Z}_{max}$ (rys. 21). Zwiększenie wagi obciążenia komunikacyjnego powyżej wartości 0,08335, np. do wartości $\rho_2 = 0{,}2$, powoduje zmianę najlepszej konfiguracji w sensie $\Delta_{\max}=0{,}5\hat{Z}_{max} + 0{,}5\tilde{Z}_{max}$ na konfigurację nr 124, która jest także najbardziej preferowana pod względem $\tilde{Z}_{max}$.



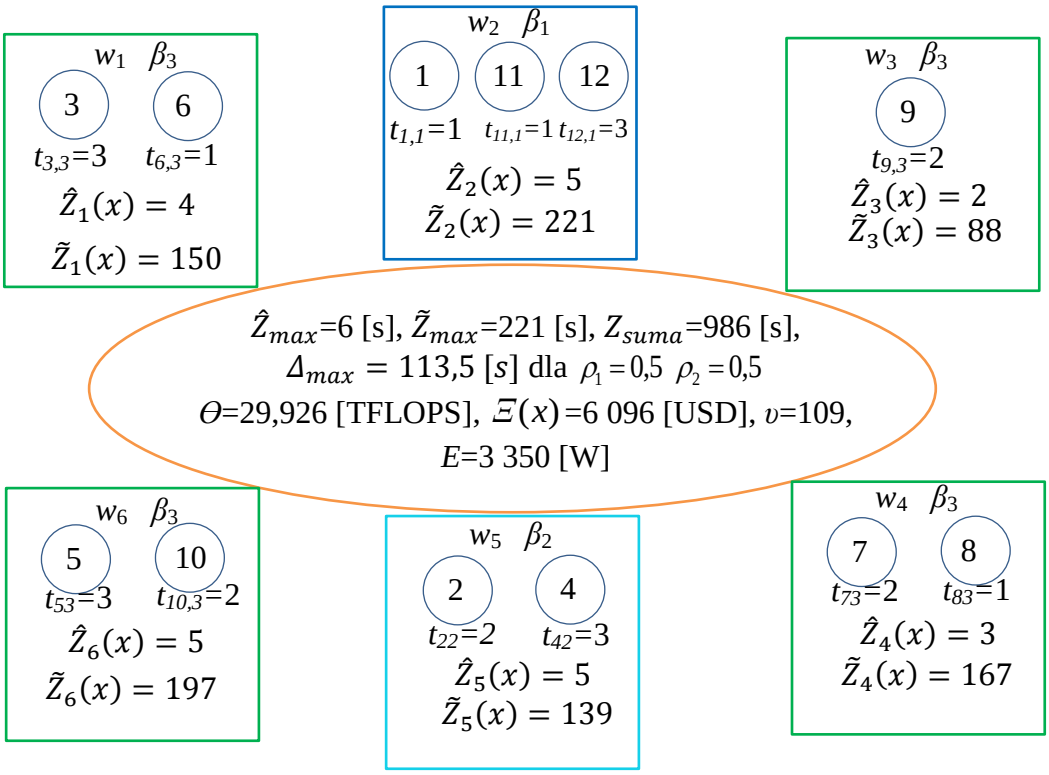
Rys. 21. Wartości Δ_{max} w zależności od wagi ρ_2 dla konfiguracji nr 124 i nr 278
Źródło: opracowanie własne

Na rysunku 22 przedstawiono graficzną interpretację przykładowej konfiguracji dopuszczalnej dla zagadnienia (3.1.1). Powyższą konfigurację można zapisać jako X^{α}=[2,5,1,5,6,1,4,4,3,6,2,2]^T, X^{β}=[3,1,3,3,2,3]^T. Konfiguracja cechuje się ważonym obciążeniem $\Delta_{max} = 113{,}5\ [s]$ dla $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$. Na podstawie danych wejściowych dla benchmarku nr 1 wyznaczono $\hat{Z}_{max}$=6 [s], $\tilde{Z}_{max}$=221 [s] oraz Z_{suma}=986 [s].

3.2. Zagwarantowanie założonej wydajności gridu

Interesariusze mający wpływ na podejmowanie decyzji mogą stawiać dodatkowe wymagania związane z zagwarantowaniem odpowiedniej jakości funkcjonowania gridu, które w konsekwencji wpływają na jakość usług obliczeniowych (ang. *Quality of Service*).

Istotne wymagania dotyczą: zagwarantowania planowanej wydajności gridu, nieprzekroczenia kosztów jego budowy, wymuszenia odpowiedniego stopnia rozproszenia modułów czy też postulatu nieprzekroczenia wielkości dostępnych zasobów sprzętowych, np. pamięci RAM czy pamięci dyskowej. Mogą wystąpić także inne wymagania, takie jak żądanie nieprzekroczenia zadanego obciążenia procesorów newralgicznego hosta, zapewnienie odpowiedniego poziomu dostępności gridu, czy też nieprzekroczenie limitu finansowego związanego z zakupem komputerów. Istotna jest także kwestia ekologiczna związana z ograniczeniem poboru mocy energii elektrycznej.



Rys. 22. Graficzna interpretacja konfiguracji gridu o specyfikacji
X^{α}=[2,5,1,5,6,1,4,4,3,6,2,2]^T, X^{β}=[3,1,3,3,2,3]^T
Źródło: opracowanie własne.

Uwzględnienie podzbioru dodatkowych ograniczeń wymaga sformułowania adekwatnego zagadnienia optymalizacji w oparciu o problem (3.1.1). Zakładając, że minimalizowane jest ważone obciążenie gridu, należałoby sformułować tyle zagadnień optymalizacji, ile jest kombinacji możliwych wymagań. W każdym problemie optymalizacji definiowany byłby inny zbiór rozwiązań dopuszczalnych. Takie podejście nie jest dogodne do wykorzystania przez administratora lub projektanta gridu, dlatego też proponuje się interaktywne definiowanie preferencji interesariuszy w aplikacji ADAIORG'14.

Uzasadnione w wielu sytuacjach decyzyjnych jest nakładanie na konfigurację gridu ograniczenia (2.2.11), aby sumaryczna wydajność $\Theta(x)$ nie była mniejsza niż wymagane minimum wydajności ϑ_{min} dla zadanego benchmarku wydajnościowego. W tym wypadku niezbędne są dodatkowe dane: $\vartheta = [\vartheta_1,...,\vartheta_j,...,\vartheta_J]$ oraz ϑ_{min}. Adekwatne zagadnienie minimalizacji $\Delta_{\max}$ z uwzględnieniem ograniczenia wydajnościowego (2.2.11) można sformułować, jak niżej.

Dla danych wejściowych: ρ_1, ρ_2, I, V, J, $\boldsymbol{T} = \left[t_{vj}\right]_{V\times J}$, $\boldsymbol{\tau} = [\tau_{vuik}]_{V\times V\times I\times I}$,

$\vartheta = [\vartheta_1,...,\vartheta_j,...,\vartheta_J]$, ϑ_{min}

należy wyznaczyć x^*, takie że:

$$\Delta_{max}(x^*) = \min_{x\in\hat{\boldsymbol{X}}} \Delta_{max}(x), \tag{3.2.1}$$

gdzie $\hat{\boldsymbol{X}} = \left\{ x \in \boldsymbol{B}_{\text{bin}}{}^{I(V+J)} \mid \sum_{i=1}^{I} x_{vi}^{\alpha} = 1 \text{ dla } v = \overline{1,V};\ \sum_{j=1}^{J} x_{ij}^{\beta} = 1 \text{ dla } i = \overline{1,I};\ \Theta(x) \geq \vartheta_{\text{min}} \right\}$.

Warto podkreślić, że NP-trudne zagadnienie (3.2.1) jest jednokryterialnym zadaniem optymalizacji zero-jedynkowej, którego binarna przestrzeń przeszukiwań zawiera $2^{I(V+J)}$ elementów, analogicznie do zagadnienia (3.1.1). Jednakże ze względu na ograniczenie (2.2.11), które nazywamy umownie wymaganiem nr 1, zbiór rozwiązań dopuszczalnych może być zbiorem pustym dla zbyt wysokiej wartości ϑ_{min}. Zbiór rozwiązań dopuszczalnych zawiera co najwyżej $I^V J^I$ elementów.

Proponowana metoda rozwiązania zadania (3.2.1) opiera się na algorytmie harmonicznym, który nie zawiera mechanizmów do dyskryminacji rozwiązań niedopuszczalnych. Dlatego też proponuje się zastosowanie funkcji kary do pogorszenia wartości funkcji *fitness* konfiguracji niedopuszczalnych. Wartość kary alternatywy niedopuszczalnej jest mniejsza, jeśli ograniczenie (2.2.11) jest przekroczone w mniejszym stopniu, a innej konfiguracji niedopuszczalnej – w większym.

Kodowanie całkowitoliczbowe konfiguracji powoduje, że do spełnienia pozostaje jedynie ograniczenie wydajnościowe. Z tego względu wartość maksymalizowanej funkcji sprawności *fitness* w algorytmie harmonicznym obliczana jest zgodnie z następującą zależnością:

$$\mathit{fitness}(x) = \begin{cases} -\Delta_{\max}(x) + P_{\max}, \text{ dla } x \in \hat{\boldsymbol{X}}; \\ -\Delta_{\max}(x) - \mu_1 P_1(x), \text{ dla } x \notin \hat{\boldsymbol{X}}; \end{cases} \tag{3.2.2}$$

gdzie:

$P_{\max}$ – oszacowanie górne funkcji celu $\Delta_{\max}$,

μ_1 – selektor wyboru wymagania nr 1 opisanego nierównością (2.2.11) [s/FLOPS], przy czym zachodzi:

$$\mu_1 = \begin{cases} 0, \text{ gdy decydent zrezygnuje z wymagania nr 1;} \\ \geq 1, \text{ gdy decydent wybierze wymaganie nr 1, } \mu_1 \text{ jest współczynnikiem funkcji kary,} \end{cases}$$

$P_1(x)$ – dodatnia wartość kary za niespełnienie ograniczenia wydajnościowego (2.2.11) przez konfigurację niedopuszczalną x.

W wypadku weryfikacji ograniczenia (2.2.11) nakładana jest kara na x, jak niżej:

$$P_1(x) = \begin{cases} 0, & \text{dla } \Theta(x) \geq \vartheta_{\min}; \\ \vartheta_{\min} - \Theta(x), & \text{dla } \Theta(x) < \vartheta_{\min}. \end{cases} \tag{3.2.3}$$

Jednym ze sposobów oszacowania $P_{\max}$ jest poniższa zależność:

$$P_{\max} = \rho_1 \hat{Z}_{\max}^{\max} + \rho_2 \tilde{Z}_{\max}^{\max}, \tag{3.2.4}$$

gdzie:

$\hat{Z}_{\max}^{\max} = \sum_{v=1}^{V} \max_{j=1,J} \{t_{vj}\}$ – oszacowanie od góry maksymalnej wartości obciążenia CPU, będąca sumą największych elementów w kolumnach macierzy ***T***;

$\tilde{Z}_{\max}^{\max} = \sum_{v=1}^{V} \sum_{u=1}^{U} \max_{i=1,I;\, k=1,I} \{\tau_{vuik}\}$ – oszacowanie od góry maksymalnej wartości obciążenia komunikacyjnego, przy czym oszacowanie jest sumą wszystkich elementów macierzy τ dla par modułów (v,u), które przydzielono do par węzłów powodujących największe obciążenie komunikacyjne między tymi modułami.

Wartość $\hat{Z}_{\max}^{\max}$ wynosi 34 [s], a $\tilde{Z}_{\max}^{\max}$ - 523 [s] dla macierzy T i τ z benchmarku nr 1. Załóżmy, że projektowany jest grid cechujący się minimalną wydajnością ϑ_{min}=20 [TFLOPS] wg testu *Linpack*. Niech w sześciu węzłach mogą być rozmieszczone hosty wybierane spośród trzech ich rodzajów o następujących szacowanych wydajnościach $\vartheta = [1{,}84;\ 4{,}848;\ 6{,}771]$ [TFLOPS]. Przy tych założeniach konfiguracja z rysunku 22 cechuje się łączną wydajnością $\Theta(x)$=29,926 [TFLOPS], a zatem spełnia ograniczenie wydajnościowe dla ϑ_{min}=20 [TFLOPS].

3.3. Uwzględnienie wymagań interesariuszy gridu

Oprócz żądania zagwarantowania założonej wydajności gridu, interesariusze mogą formułować rozmaite inne wymagania, np. za pomocą wykorzystania selektorów wymagań wprowadzanych interaktywnie przez interfejs graficzny aplikacji ADAIORG'14. Z tego powodu poniżej rozpatruje się nieco bardziej złożoną sytuację decyzyjną, w której opcjonalne ograniczenia dotyczą nieprzekroczenia kosztów zakupu hostów, zapewnienia odpowiedniego rozproszenia modułów, nieprzekroczenia

wielkości dostępnych zasobów pamięci RAM i pamięci dyskowej. Ponadto istnieje możliwość nałożenia ograniczenia na łączne zużycie energii elektrycznej przez komputery gridu.

Istotną rolę odgrywają w tym wypadku selektory wyboru ograniczeń, których wartości uzależnione są od interaktywnego wyboru wymagań przez interesariuszy. W tabeli nr 2 przedstawiono numery selektorów μ_k w odniesieniu do rozważanych wymagań projektowych gridu.

Tabela 2. Wartości selektorów wymagań jakościowych gridu

Nr selektora k	Uwzględnienie wymagania projektowego
1	Ograniczenie wydajnościowe (2.2.11)
2	Nieprzekroczenie kosztów zakupu hostów (2.2.12)
3	Zapewnienia minimalnego stopnia rozproszenia modułów (2.2.14)
4	Nieprzekroczenia wielkości zasobów w hostach (2.2.16)
5	Ograniczenie energetyczne (2.2.17)

Źródło: opracowanie własne.

Jeśli decydent wybierze k-te ograniczenie, to k-ty selektor otrzymuje wartość nie mniejszą niż 1 i pełni rolę współczynnika funkcji kary w zastosowanej metodzie optymalizacji konfiguracji gridu. Powyższe specyfikuje się, jak niżej:

$$\mu_k = \begin{cases} 0, \text{ gdy decydent zrezygnuje z wymagania nr } k, \\ \geq 1, \text{ gdy decydent wybierze wymaganie nr } k \text{ (wsp.funkcji kary),} \end{cases} \quad k = \overline{1,5}.$$

Zagadnienie optymalizacji obciążeń ze względu na kryterium $\Delta_{\max}$ przy uwzględnieniu opcjonalnych wymagań interesariuszy sformułowano następująco.

Dla danych wejściowych: $\rho_1, \rho_2, \mu_1, \mu_2, \mu_3, \mu_4, \mu_5$, V, I, J, $T = [t_{vj}]_{V\times J}$,

$\tau = [\tau_{vuik}]_{V\times V\times I\times I}$, $\vartheta = [\vartheta_1, ..., \vartheta_j, ..., \vartheta_J]$, ϑ_{min}, $\xi = [\xi_1, ..., \xi_j, ..., \xi_J]$, $\xi_{\max}$, $\upsilon_{\min}$,

$\tilde{D} = [\tilde{d}_{jr}]_{J\times 2}$, $C = [c_{vr}]_{V\times 2}$, $\varepsilon = [\varepsilon_1, ..., \varepsilon_j, ..., \varepsilon_J]$, $\varepsilon_{\max}$

należy wyznaczyć x^*, takie że:

$$\Delta_{max}(x^*) = \min_{x\in\tilde{\boldsymbol{X}}} \Delta_{max}(x), \tag{3.3.1}$$

gdzie

$$\tilde{\boldsymbol{X}} = \{x \in \boldsymbol{B}^{I(V+J)} | \sum_{i=1}^{I} x_{vi}^{\alpha} = 1 \text{ dla } v = \overline{1,V};\ \sum_{j=1}^{J} x_{ij}^{\beta} = 1 \text{ dla } i = \overline{1,I};\ \Theta(x) \geq \vartheta_{\min} \text{ dla } \mu_1 \neq 0;$$

$$\Xi(x) \leq \xi_{\max} \text{ dla } \mu_2 \neq 0; \quad \upsilon(x) \geq \upsilon_{\min} \text{ dla } \mu_3 \neq 0;$$

$$\kappa_{ir}(x) \geq 0, \quad i = \overline{1,I}, \; r = \overline{1,2} \text{ dla } \mu_4 \neq 0; \quad E(x) \leq \varepsilon_{\max} \text{ dla } \mu_5 \neq 0\}$$

przy czym:

$$\Theta(x) = \sum_{i=1}^{I}\sum_{j=1}^{J} \vartheta_j x_{ij}^{\beta}, \; \Xi(x) = \sum_{i=1}^{I}\sum_{j=1}^{J} \xi_j x_{ij}^{\beta}, \; \upsilon(x) = \sum_{v=1}^{V}\sum_{\substack{u=1 \\ u>v}}^{V}\sum_{i_1=1}^{I}\sum_{\substack{i_2=1 \\ i_1 \neq i_2}}^{I} x_{vi_1}^{\alpha} x_{ui_2}^{\alpha},$$

$$\kappa_{ir}(x) = \sum_{j=1}^{J} \tilde{d}_{jr} x_{ij}^{\beta} - \sum_{v=1}^{V} c_{vr} x_{vi}^{\alpha}, \quad i = \overline{1,I}, \; r = \overline{1,2}; \; E(x) = \sum_{i=1}^{I}\sum_{j=1}^{J} \varepsilon_j x_{ij}^{\beta}.$$

Binarna przestrzeń przeszukiwań NP-trudnego zagadnienia optymalizacji kombinatorycznej (3.3.1) zawiera $2^{I(V+J)}$ elementów. Zbiór rozwiązań dopuszczalnych może być zbiorem pustym, a dla pewnych instancji może zawierać $I^V J^I$ elementów.

Metoda rozwiązania problemu (3.3.1) z wykorzystaniem algorytmu harmonicznego wykorzystuje funkcję kary tak, aby każde rozwiązanie dopuszczalne było preferowane przed dowolnym wariantem niedopuszczalnym. Ponadto wartość kary alternatywy niedopuszczalnej jest mniejsza, jeśli ograniczenia są przekroczone w mniejszym stopniu, a innej konfiguracji - w większym.

Rozważa się opcjonalne wymaganie finansowe (2.2.12), aby łączny koszt zakupu komputerów $\Xi(x)$ nie przekroczył założonego w projekcie limitu finansowego $\xi_{\max}$. W tym wypadku należy oszacować koszty zakupu komputerów $\xi = [\xi_1, ..., \xi_j, ..., \xi_J]$ oraz limit $\xi_{\text{-max}}$, które to wartości są danymi wejściowymi do zagadnienia (3.3.1). Jeśli naruszenie ograniczenia (2.2.12) spowoduje, że zbiór rozwiązań dopuszczalnych będzie pusty, to wskazane jest podwyższenie limitu finansowego, a następnie można ponownie przystąpić do rozwiązania zadania (3.3.1).

Jeśli koszt $\Xi(x)$ konfiguracji gridu *x* przekroczy limit, to wartość *fitness* tej konfiguracji powinna być zmniejszona o karę $P_2(x)$, którą wyznacza się następująco:

$$P_2(x) = \begin{cases} 0, \text{ dla } \Xi(x) \leq \xi_{\max}, \\ \Xi(x) - \xi_{\max}, \text{ dla } \Xi(x) > \xi_{\max}. \end{cases} \tag{3.3.2}$$

Niech komputery o wydajnościach $\vartheta = [1{,}84; 4{,}848; 6{,}771]$ [TFLOPS] cechują się następującymi kosztami $\xi = [400; 896; 1504]$ [USD]. Konfiguracja z rysunku 22 cechuje się kosztem zakupu hostów $\Xi(x)$=6 096 [USD] i jest dopuszczalna dla limitu $\xi_{\max}$ =7 000 [USD].

Wydajnością 1,840 [TFLOPS] cechowała się w listopadzie 2014 roku konsola do gier *PlayStation 4* firmy *Sony* z 64-bitowym, 8-rdzeniowym procesorem *AMD Jaguar 1,6 GHz*, 8 [GB] RAM, procesorem graficznym *AMD Radeon* i 500 [GB] HD [399]. Koszt PS 4 w przeliczeniu na 1 [TFLOPS] wynosi w tym wypadku zaledwie 217 [USD]. Wprawdzie konsola nie jest przeznaczona do obliczeń uniwersalnych lub też realizacji dystrybucji zadań obliczeniowych i danych, to warto wspomnieć, że w 2010 roku skonstruowano z 1 760 konsol *PlayStation 3* superkomputer dla *US Air Force*, który cechował się wydajnością 500 [TFLOPS] [406]. Z powyższych powodów rozważa się *PlayStation 4*, na którym można realizować zadania w gridzie.

Natomiast wydajnością 4 848 [TFLOPS] cechuje się komputer *System Pentium G550 & AMD R9 290* zbudowany z komercyjnie dostępnych podzespołów o łącznej cenie 896 USD [383]. Komputer składa się z CPU I*ntel Pentium G550 2.6 GHz Dual-Core*, pamięci RAM 1 [GB] oraz dwóch procesorów graficznych *Gigabyte Radeon R9 290* z pamięcią 4 [GB]. Pamięć dyskową o pojemności 32 [GB] w podstawowej wersji hosta proponuje się zastąpić dyskiem *Western Digital WD30EURS* o pojemności 3 [TB] oraz zwiększyć wielkość pamięci RAM do 12 [GB]. Koszt podzespołów niezbędnych do uzyskania wydajności 1 [TFLOPS] to 185 [USD].

Największą wydajnością 6,771 [TFLOPS] cechuje się host *System AMD Sempron 145 & 3 GeForce GTX 760* zbudowany także z komercyjnie dostępnych podzespołów o łącznej cenie 1 504 [USD] [382]. Komputer składa się z CPU *AMD Sempron 145 2.8GHz Single-Core*, pamięci RAM 8 [GB] oraz trzech procesorów graficznych *Asus GeForce GTX 760* z pamięcią 2 [GB]. Komputer w wersji podstawowej nie jest wyposażony w pamięć dyskową, dlatego proponuje się rozszerzyć konfigurację komputera o dysk *Western Digital WD60EZRX* o pojemności 6 [TB] oraz zwiększyć wielkość pamięci RAM do 32 [GB]. Koszt podzespołów niezbędnych do uzyskania wydajności 1 [TFLOPS] to dla tego zestawu 223 [USD].

Warto podkreślić, że ceny sprzętu komputerowego umożliwiającego prowadzenie obliczeń z mocą 1 [TFLOPS] bardzo intensywnie obniżyły się w ostatnich 17 latach aż do poziomu 185 [USD] dla węzłów obliczeniowych „średniej" klasy w listopadzie 2014 roku. Superkomputer *ASCI Red* firmy *Intel* był pierwszym komputerem o mocy obliczeniowej nieco powyżej 1 [TFLOPS] w 1997 roku, przy czym zasadniczy wpływ na koszt jego budowy wynoszący 46 mln [USD] miało zainstalowanie ponad 6 000 procesorów *Pentium Pro* 200 MHz.

Superkomputer *Roadrunner* firmy IBM, który przekroczył jako pierwszy „barierę” 1 [PFLOPS] w 2008 roku, składał się z 12 960 procesorów *IBM PowerXCell 8i* oraz 6 480 procesorów *AMD Opteron dual-core*. Koszt sprzętu umożliwiającego prowadzenie obliczeń z wydajnością 1 [TFLOPS] zredukowano do ok. 100 tys. [USD].

Natomiast koszt sprzętu do prowadzenia obliczeń o wydajności 1 [TFLOPS] dla najszybszego w 2014 roku komputera na świecie *Tianhe-2* o mocy obliczeniowej 33,86 [PFLOPS], to 11 512 [USD]. Superkomputer zbudowany jest z 16 000 węzłów, w tym 32 000 12-rdzeniowych CPU *Xeon Ivy Bridge* 1,1 GHz oraz 48 000 57-rdzeniowych koprocesorów *Xeon Phi*. Warto podkreślić, że rdzeń CPU może wykonać do 16 operacji zmiennoprzecinkowych w jednym takcie dla czterech wątków.

Różnica pod względem kosztu sprzętu niezbędnego do obliczeń z intensywnością 1 [TFLOPS] między komputerem *System Pentium G550 & AMD R9 290* a superkomputerem *Tianhe-2* wynika ze znacznie większej łącznej pamięci RAM oraz pamięci dyskowej. Superkomputer posiada 1,34 [PB] pamięci RAM oraz 12,4 [PB] pamięci dyskowej [406].

Trzecim opcjonalnym wymaganiem (k=3) decydenta może być żądanie, aby co najmniej $\upsilon_{\min}$ par modułów przydzielono do różnych węzłów, przy czym $1 \le \upsilon_{\min} \le V(V\text{-}1)$. Konfiguracja z rysunku 22 cechuje się stopniem rozproszenia modułów $\upsilon(x)$=109, a w wypadku żądanego minimalnego stopnia rozproszenia $\upsilon_{\min}$=55 jest alternatywą dopuszczalną.

Zbyt wysoka wartość $\upsilon_{\min}$ może spowodować, że zbiór rozwiązań dopuszczalnych będzie pusty. Jeśli konfiguracja nie spełnia ograniczenia na minimalny stopień rozproszenia modułów (2.2.14), to adekwatna funkcja kary ma postać, jak niżej:

$$P_3(x) = \begin{cases} 0, & \text{dla } \upsilon(x) \ge \upsilon_{\min}, \\ \upsilon_{\min} - \upsilon(x), & \text{dla } \upsilon(x) < \upsilon_{\min}. \end{cases} \tag{3.3.3}$$

Kolejnym istotnym ograniczeniem (k=4) jest wymaganie (2.2.16), aby w każdym hoście dostępna była wystarczająca wielkość zasobów w odniesieniu do zapotrzebowania ze strony modułów.

Funkcje kary w wypadku braku wystarczającej ilości zasobów mają następującą postać:

$$P_{4ir}(x) = \begin{cases} 0, & \text{dla } \kappa_{\text{ir}}(x) \ge 0 \\ -\kappa_{\text{ir}}(x), & \text{dla } \kappa_{\text{ir}}(x) < 0 \end{cases} \quad i = \overline{1,I},\ r = \overline{1,2}. \tag{3.3.4}$$

Przekroczenie zarezerwowanego limitu pamięci RAM wiąże się z wystąpieniem odpowiedniego wyjątku w systemie operacyjnym, który nie zawsze jest prawidłowo obsługiwany przez współczesne systemy operacyjne. Powyższe może spowodować zawieszenie wykonywania odpowiednich modułów. Dlatego dla każdego *r*-tego rodzaju pamięci wyznacza się karę za przekroczenie jego wielkości, jak niżej:

$$P_{4r}(x) = \sum_{i=1}^{I} P_{\text{ir}}(x), \quad r = \overline{1,2};\ \text{dla } x \notin \tilde{\boldsymbol{X}}. \tag{3.3.5}$$

Niech w rozważanym benchmarku nr 1 macierz dostępnych zasobów, które ograniczone są do pamięci RAM i pamięci HD ma następującą postać:

$$\tilde{D} = \left[\tilde{d}_{jr}\right]_{J\times 3} = \begin{bmatrix} 8{,}256 & 0{,}5 \\ 20{,}000 & 3{,}0 \\ 38{,}000 & 6{,}0 \end{bmatrix}_{3\times 3} \tag{3.3.6}$$

Wiersz nr 1 odpowiada wielkościom zasobów w *PlayStation 4,* wiersz nr 2 – w *System Pentium G550 & AMD R9 290,* a wiersz nr 3 – w *System AMD Sempron 145 & 3 GeForce GTX 760*. W pierwszej kolumnie umieszczono łączne wielkości pamięci RAM dla CPU w [GB], w drugiej kolumnie - pojemności dysków w [TB]. Na podstawie macierzy dostępnych zasobów $\tilde{D}$ dla konfiguracji X^{α}=[2, 5, 1, 5, 6, 1, 4, 4, 3, 6, 2, 2]^T, X^{β}=[2, 3, 2, 1, 2, 3]^T można wyznaczyć łączną wielkość pamięci operacyjnej w gridzie jako 144,265 [GB] oraz całkowitą pojemność dysków - 21,5 [TB].

Niech macierz zapotrzebowań na pamięć RAM i HD $C = \left[c_{vr}\right]_{V\times 2}$ jest następująca:

$$C = \left[c_{vr}\right]_{V\times 2} = \begin{bmatrix} 0{,}5 & 0{,}01 \\ 0{,}3 & 0{,}30 \\ 0{,}4 & 0{,}02 \\ 0{,}3 & 0{,}02 \\ 0{,}5 & 0{,}03 \\ 1{,}7 & 0{,}03 \\ 2{,}7 & 0{,}02 \\ 0{,}3 & 0{,}03 \\ 0{,}4 & 0{,}04 \\ 0{,}5 & 0{,}02 \\ 0{,}3 & 0{,}03 \\ 0{,}5 & 0{,}02 \end{bmatrix}_{12\times 2} \tag{3.3.7}$$

Wartość zapotrzebowania przez wszystkie moduły na pamięć RAM to 8,4 [GB], a łączne zapotrzebowanie na pamięć HD wynosi 0,57 [TB]. Jeśliby zatem wszystkie moduły przydzielono do *PlayStation 4*, to nie wystarczyłoby ani pamięci RAM ani pamięci HD. Natomiast łączna rezerwa pamięci RAM w gridzie wynosi $M_1(x)$=135,865 [GB], a sumaryczna rezerwa pamięci dyskowej $M_2(x)$=20,93 [TB].

W tabeli nr 3 przedstawiono wartości rezerw pamięci w węzłach $\kappa_{ir}(x), i=\overline{1,6},\ r=\overline{1,2}$ dla konfiguracji z rysunku 22. Z tabeli wynika, że wielkości zasobów nie są przekroczone i spełnione jest ograniczenie (2.2.16). Warto zauważyć, że $M_1^{\min}(x)$ - rezerwa pamięci RAM w newralgicznym hoście ma wartość 5,265 [GB], a $M_2^{\min}(x)$ - rezerwa pamięci HD w newralgicznym hoście - 0,45 [TB]. Newralgicznym komputerem jest *PlayStation 4* usytuowany w węźle nr 4.

Tabela 3. Macierz rezerw zasobów $Kappa=[\kappa_{ir}]_{6\times 2}$

Nr węzła i	RAM [GB] r=1	HD [TB] r=2
1	17,900	2,95
2	36,700	5,94
3	19,600	2,96
4	5,265	0,45
5	19,400	2,68
6	37,000	5,95
$M_r(x)$	135,865	20,93
$M_r^{\min}(x)$	5,265	0,45

Źródło: opracowanie własne.

Piątym opcjonalnym wymaganiem (k=5) jest żądanie, aby ograniczenie (2.2.17) na pobór mocy energii nie zostało przekroczone. Jeśli komputery są eksploatowane przez kilka lat, to istotne znaczenie odgrywa zużycie energii. W wielu wypadkach jej koszt w tym okresie stanowi wyższą wartość niż koszt inwestycyjny związany z zakupem komputerów przy uwzględnieniu metodyki porównywania kosztów inwestycyjnych z kosztami eksploatacji.

Jeśli łączny pobór mocy w gridzie $E(x)$ [W] przekroczy założony limit $\varepsilon_{\max}$, to *fitness* tej konfiguracji jest zmniejszane o karę $P_5(x)$, którą wyznacza się następująco:

$$P_5(x)=\begin{cases}0, \text{ dla } E(x)\le\varepsilon_{\max},\\ E(x)-\varepsilon_{\max}, \text{ dla } E(x)>\varepsilon_{\max}.\end{cases} \tag{3.3.8}$$

W tym wypadku znany jest wektor poboru mocy przez komputery $\varepsilon=[\varepsilon_1,...,\varepsilon_j,...,\varepsilon_J]$ oraz limit $\varepsilon_{\max}$. Niech parametry trzech rodzajów dostępnych komputerów będą zapisane następująco w wektorze $\varepsilon=[250;600;650]$ [W]. Jeśli maksymalny pobór mocy $\varepsilon_{\max}$=3750 [W], to konfiguracja z rysunku 22 jest dopuszczalna, gdyż cechuje się zużyciem energii wynoszącym 3 350 [W]. Natomiast konfiguracja, w której do węzłów przydzielono najbardziej wydajne komputery o mocy

650 [W] będzie wariantem niedopuszczalnym ze względu na łączne zapotrzebowanie na energię wynoszące 3 900 [W].

Reasumując, kodując konfiguracje całkowitoliczbowo oraz uwzględniając ograniczenie wydajnościowe, finansowe, na rozproszenie modułów, a także ograniczenia na wielkości pamięci czy pobór mocy, wartość maksymalizowanej funkcji sprawności obliczana jest zgodnie z poniższą zależnością:

$$fitness(x) = \begin{cases} -\Delta_{\max}(x) + P_{\max}, \text{ dla } x \in \tilde{\boldsymbol{X}}, \\ -\Delta_{\max}(x) - \sum_{k=1}^{3} \mu_k P_k(x) - \mu_4 \sum_{r=1}^{2} P_{4r}(x) + \mu_5 P_5(x), \text{ dla } x \notin \tilde{\boldsymbol{X}}, \end{cases} \quad (3.3.9)$$

gdzie $P_k(x)$ – wartość kary za niespełnienie k-tego ograniczenia dla rozwiązania niedopuszczalnego, $k = \overline{1,5}$.

Minimalizując dowolne kryterium oceny konfiguracji gridu f dla zbioru rozwiązań dopuszczalnych $\tilde{\boldsymbol{X}}$, wartość *fitness* w algorytmie harmonicznym wyznaczamy, jak niżej:

$$fitness(x) = \begin{cases} -f(x) + f_{\max}, \text{ dla } x \in \tilde{\boldsymbol{X}}, \\ -f - \sum_{k=1}^{3} \mu_k P_k(x) - \mu_4 \sum_{r=1}^{2} P_{4r}(x) + \mu_5 P_5(x), \text{ dla } x \notin \tilde{\boldsymbol{X}}, \end{cases} \quad (3.3.10)$$

gdzie $f_{\max}$ jest maksymalną wartością funkcji f w zbiorze $\tilde{\boldsymbol{X}}$.

3.4. Dwukryterialna optymalizacja równoważenia obciążeń

Mimo że agregacja obciążeń $\hat{Z}_{\max}$ oraz $\tilde{Z}_{max}$ zachodzi w $\Delta_{\max}$ - ważonym obciążeniu newralgicznego hosta, to w tym wypadku istotnym dylematem jest wyznaczenie wartości wag. Z drugiej strony równoważenie obciążeń w gridzie może być osiągnięte za pomocą równoczesnej optymalizacji wybranych kryteriów oceniających jakość konfiguracji systemu. Można zatem rozważać minimalizację Z_{suma} - łącznego obciążeniu gridu oraz $Z_{\max}$ - łącznego obciążenia newralgicznego komputera. Ponadto można maksymalizować Θ - wydajność gridu.

Oprócz powyższych kryteriów wpływających na wydajność gridu wyróżnia się oszacowania kosztów, które także mogą podlegać minimalizacji. W szczególności minimalizacji zazwyczaj podlega K - koszt realizacji modułów w gridzie. Rozważać można także Ξ - koszt zakupu hostów. Ponadto minimalizacja E – poboru mocy

energii przez komputery gridu jest związana z redukcją kosztów eksploatacyjnych, a dodatkowo jest to istotne wymaganie z punktu widzenia ekologicznego.

Warto nadmienić, że ważną grupę miar stanowią kryteria wpływające na poziom bezpieczeństwa systemu. Decydent może preferować maksymalizację M_1 - rezerwy dostępnej wielkości pamięci RAM w gridzie lub M_2 – rezerwy dostępnej pojemności dysków twardych. Maksymalizacji może podlegać $M_1^{\min}$ - wielkość dostępnej pamięci RAM w newralgicznym komputerze lub $M_2^{\min}$ - rezerwa pamięci dyskowej w newralgicznym hoście. Do tej grupy zaliczyć można R - dostępności gridu, a także υ - stopień rozproszenia modułów.

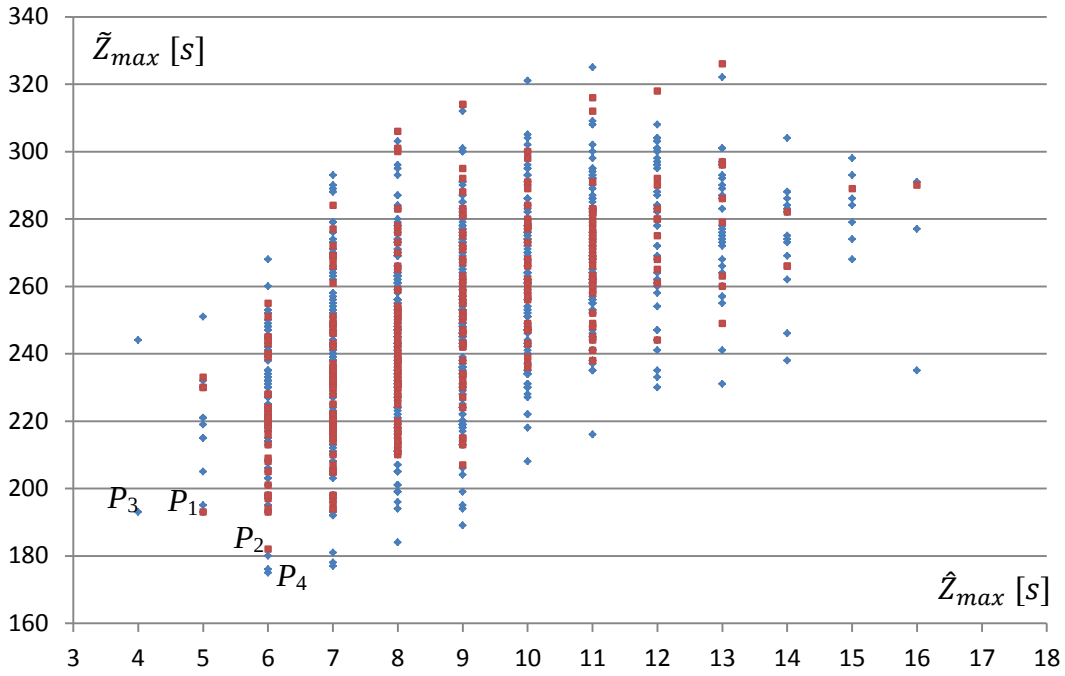
Każde powyższe kryterium może być uwzględnione w adekwatnych ograniczeniach, tak jak to przedstawiono dla ograniczeń problemu (3.3.1) w odniesieniu do Θ, Ξ, υ, κ_{ir} oraz E.

Na rysunku 23 zobrazowano przykładowe oceny obciążeń jako punkty $(\hat{Z}_{max}(x), \tilde{Z}_{max}(x))$ w odniesieniu do sześciokomputerowego gridu dla tysiąca losowo wygenerowanych konfiguracji. Kolorem czerwonym zaznaczono oceny dla 300 wygenerowanych konfiguracji w początkowym etapie.

Szczególnie interesujące są konfiguracje cechujące się ocenami P_3 i P_4. Konfiguracja nr 217 (X^{α}=[1,3,2,4,6,2,1,4,5,3,2,5]^T, X^{β}=[1,1,3,3,3,3]^T) o ocenie P_3=(4 [s]; 193 [s]) cechuje się $\Delta_{\max}$=98,5 [s] dla wektora wag [0,5; 0,5]^T, a także wartością $Z_{\text{suma}}(x)$ = 934,0 [s]. Ocena P_4 jest bardziej preferowana niż ocena P_1, gdyż posiada mniejsze obciążenie $\hat{Z}_{max}$, a $\tilde{Z}_{max}$ jest równe.

Z drugiej strony konfiguracja nr 963 (X^{α}=[4,6,1,3,3,4,5,5,6,1,2,2]^T, X^{β}=[2,3,2,1,1,1]^T) o ocenie P_4=(6 [s]; 175 [s]) jest również interesująca dla decydenta, gdyż cechuje się najniższym obciążeniem komunikacyjnym $\tilde{Z}_{max}$. Ponadto $\Delta_{\max}$=90,5 [s] dla wektora wag [0,5; 0,5]^T oraz wartością $Z_{\text{suma}}(x)$ = 992,0 [s].

Porównując konfiguracje nr 217 i 953, trudno jest jednoznacznie wybrać jedną z nich, gdyż konfiguracja nr 953 ma niższe obciążenie komunikacyjne niż konfiguracja nr 217, ale za to znacznie wyższe obciążenie związane z przetwarzaniem danych. W tej sytuacji decydent może posłużyć się dodatkowym kryterium do rozstrzygnięcia wyboru między obiema konfiguracjami. Za pomocą $\Delta_{\max}$ wybrana zostanie konfiguracja nr 953, ale ważone obciążenie Z_{suma} wskaże na konfigurację nr 217.



Rys. 23. Oceny obciążeń jako uporządkowane dwójki ($\hat{Z}_{max}$, $\tilde{Z}_{max}$) w odniesieniu do sześciokomputerowego gridu dla tysiąca wygenerowanych konfiguracji
Źródło: opracowanie własne.

Z punktu widzenia decydenta istotny jest wybór kryteriów do oceny konfiguracji. Jeśli proces decyzyjny jest oparty o co najmniej dwa kryteria, to problem optymalizacji konfiguracji gridu, który do tej pory sformułowano jako zagadnienia optymalizacji jednokryterialnej: (3.1.1), (3.2.1) czy (3.3.1), należy sformułować jako problem optymalizacji wielokryterialnej [18].

Z punktu widzenia interaktywnej aplikacji programistycznej usprawniającej proces decyzyjny, istotną rolę odgrywają selektory wyboru kryteriów, które to selektory otrzymują określone wartości w wyniku wyboru kryterium przez decydenta. W tabeli nr 4 przedstawiono numery selektorów v_n w odniesieniu do rozważanych kryteriów.

Trudność w agregacji kryteriów związana jest z różnymi jednostkami miary dla tych kryteriów, gdyż nie można wprost porównać sekund, jednostek monetarnych, FLOPS-ów, bajtów czy wartości liczbowych bez mian. W niektórych zagadnieniach można zastosować wagi, ale jest to związane z heurystycznym ich doborem, co może wpłynąć na jakość wyznaczonych rozwiązań.

Bez straty ogólności rozważań rozpatruje się jednoczesną minimalizację obciążeń $\hat{Z}_{max}$ oraz $\tilde{Z}_{max}$, przy czym zakłada się, że zbiór rozwiązań dopuszczalnych $\tilde{X}$ zdefiniowany jest tak, jak w zagadnieniu (3.3.1).

Tabela 4. Wartości selektorów v_n dla kryteriów do oceny gridu

Indeks selektora n	Kryterium oceny jakości gridu
1	$\hat{Z}_{max}$ – obciążenie związane z przetwarzaniem danych newralgicznego hosta [s]
2	$\tilde{Z}_{max}$ – obciążenie komunikacyjne newralgicznego komputera w gridzie [s]
3	$\Delta_{\max}$ – ważone obciążenie newralgicznego hosta [s]
4	Z_{max} – łączne obciążenie newralgicznego komputera w gridzie [s]
5	Z_{suma} – łączne obciążenie gridu [s]
6	Θ – wydajność gridu wg założonego benchmarku [FLOPS]
7	K – koszt przetwarzania danych i komunikacji w gridzie [JM]
8	Ξ – koszt zakupu hostów [JM]
9	R – dostępność gridu
10	υ – stopień rozproszenia modułów
11	M_1 – rezerwa dostępnej w gridzie wielkości pamięci RAM [B]
12	M_2 – rezerwa dostępnej w gridzie pojemności dysków twardych [B]

Źródło: opracowanie własne.

Niech φ oznacza kryterium wektorowe, takie że:

$$\varphi: \boldsymbol{X} \to \boldsymbol{R}^2, \tag{3.4.1}$$

gdzie:

$\boldsymbol{X}$ - przestrzeń konfiguracji gridu będąca zbiorem rozwiązań dopuszczalnych zagadnienia (3.3.1), $\tilde{\boldsymbol{X}} \subset \boldsymbol{X}$;

$\boldsymbol{R}^2$ - przestrzeń ocen konfiguracji, $\boldsymbol{R}$ - zbiór liczb rzeczywistych.

Ocena konfiguracji $x \in \tilde{\boldsymbol{X}}$ może być zapisana następująco:

$$\varphi(x) = [\hat{Z}_{max}(x),\ \tilde{Z}_{max}(x)]^T = [\varphi_1(x),\ \varphi_2(x)]^T \in \boldsymbol{R}^2 \ . \tag{3.4.2}$$

Zauważalny jest konflikt między zmniejszaniem obciążenia procesorów a zmniejszaniem obciążenia komunikacyjnego. Konfiguracje cechujące się mniejszym obciążeniem procesorów mogą być bardziej obciążone pod względem komunikacyjnym w sytuacji, gdy moduły są bardzo rozproszone. Z kolei skupienie modułów w jednym węźle redukuje obciążenie komunikacyjne, ale zwiększa obciążenie procesorów newralgicznego hosta.

Rozwiązanie zaistniałej sytuacji konfliktowej wymaga rozważenia *zbioru ocen*, w którym można porównywać oceny konfiguracji. *Zbiór ocen* $\boldsymbol{\Omega}$ to obraz zbioru konfiguracji dopuszczalnych $\tilde{\boldsymbol{X}}$ dla funkcji oceny φ, co można zapisać jak niżej [17]:

$$\boldsymbol{\Omega} = \varphi(\tilde{\boldsymbol{X}}) = \{\omega \in \boldsymbol{R}^2 \mid \omega = \varphi(x),\ x \in \tilde{\boldsymbol{X}}\}. \tag{3.4.3}$$

Wielokryterialny wybór konfiguracji gridu związany jest z porządkowaniem przestrzeni przeszukiwań $\boldsymbol{R}^2$ za pomocą relacji dominowania $\chi \subset \boldsymbol{R}^2 \times \boldsymbol{R}^2$. Jeśli decydent preferuje jednoczesną minimalizację dwóch kryteriów obciążeń gridu $\hat{Z}_{max}$ oraz $\tilde{Z}_{max}$, to rekomendowaną konfiguracją dopuszczalną $x^* \in \widetilde{\boldsymbol{X}}$ będzie taka, która cechuje się najmniejszą wartością obu kryteriów. Konfiguracja x^* nazywana jest wówczas *dominującą* w sensie relacji $\chi^{\leq} \subset \boldsymbol{R}^2 \times \boldsymbol{R}^2$ dla zagadnienia polioptymalizacji obciążenia gridu, które to zagadnienie jest uporządkowaną trójką ($\widetilde{\boldsymbol{X}}$, φ, $\chi^{\leq}$).

Relację dominowania $\chi^{\leq}$ w zbiorze ocen $\boldsymbol{\Omega} \subset \boldsymbol{R}^2$ definiuje się następująco [293]:

$$\chi^{\leq} = \{(a,b) \in \boldsymbol{\Omega} \times \boldsymbol{\Omega} |\, a_n \leq b_n \text{ dla } n = \overline{1,2}\}, \tag{3.4.4}$$

gdzie:
$a = [a_1, a_2]^T \in \boldsymbol{\Omega}$,
$b = [b_1, b_2]^T \in \boldsymbol{\Omega}$.

Relacja dominowania $\chi^{\leq}$ generowana jest przez stożek $\Lambda_{\leq} \subset \boldsymbol{R}^2$ [17], zdefiniowany, jak niżej [18]:

$$\Lambda_{\leq} = \{\lambda = [\lambda_1, \lambda_2]^T \in \boldsymbol{R}^2 \mid \lambda_n \geq 0, n = \overline{1,2}\}. \tag{3.4.5}$$

Ze względu na fakt, że zbiór ocen $\boldsymbol{\Omega}$ jest ograniczony, to *dominująca konfiguracja gridu* x^* w zagadnieniu polioptymalizacji ($\widetilde{\boldsymbol{X}}$, φ, $\chi^{\leq}$) spełnia warunek, jak niżej:

$$\varphi_n(x^*) = \min_{x \in \widetilde{\boldsymbol{X}}} \varphi_n(x) \text{ dla } n = \overline{1,2}. \tag{3.4.6}$$

Oceną dominującą $\omega^* = \varphi(x^*)$ nazywamy obraz konfiguracji dominującej x^*, a zbiór rozwiązań dominujących oznaczamy przez $\boldsymbol{X}_D^{\leq}$. Ocena dominująca ω^* dominuje pozostałe oceny w sensie przyjętej relacji porządku $\chi^{\leq}$.

Twierdzenie 3.1

Istnieje co najwyżej jeden wynik dominujący ω^* w sensie relacji $\chi^{\leq}$ w zbiorze ocen $\boldsymbol{\Omega}$ dla funkcji kryterium φ zdefiniowanej za pomocą (3.4.1) oraz zbioru rozwiązań dopuszczalnych $\widetilde{\boldsymbol{X}}$ określonego za pomocą (3.3.1).

Dowód: Ponieważ para ($\boldsymbol{\Omega}$, $\chi^{\leq}$) jest zbiorem uporządkowanym, to z własności zbiorów uporządkowanych [293] wynika, że zawiera co najwyżej jeden element najmniejszy. Element najmniejszy jest wynikiem dominującym ω^* w zbiorze $\boldsymbol{\Omega}$, a zatem istnieje co

najwyżej jeden wynik dominujący ω^* w sensie relacji $\chi^{\preceq}$, która jest relacją porządku, co kończy dowód. □

Pojedynczej ocenie dominującej ω^* może odpowiadać wiele konfiguracji dominujących. Jednakże eksperymenty numeryczne z różnorodnymi instancjami wykazały, że zazwyczaj ocena dominująca nie istnieje (rys. 23). Powyższa sytuacja wynika z konfliktu między kryteriami cząstkowymi i jest często obserwowana także w innych zagadnieniach optymalizacji wektorowej [21, 33].

Zamiast konfiguracji dominujących proponuje się wyznaczenie konfiguracji optymalnych w sensie *Pareto* (rozwiązania sprawne, efektywne, *Pareto*-optymalne). Dla *N* kryteriów cząstkowych konfigurację $x^{\preceq} \in \tilde{\boldsymbol{X}}$ nazywamy *optymalną w sensie Pareto* wtedy i tylko wtedy, gdy w zbiorze rozwiązań dopuszczalnych $\tilde{\boldsymbol{X}}$ nie istnieje rozwiązanie *x**, takie że $\varphi_n(x^*) \le \varphi_n(x^{\preceq})$ dla $n = \overline{1,N}$ oraz istnieje $k \in \overline{1,N}$, takie że $\varphi_k(x^*) < \varphi_k(x^{\preceq})$ [8].

Element niezdominowany $\omega^{ND} \in \boldsymbol{\Omega}$ w sensie relacji $\chi^{\preceq}$ w zbiorze $\boldsymbol{\Omega}$, cechuje się tym, że nie istnieje $\omega^* \in \boldsymbol{\Omega}$, $\omega^{ND} \neq \omega^*$, takie że $(\omega, \omega^{ND}) \in \chi^{\preceq}$. Ocena konfiguracji *Pareto*-optymalnej jest zatem elementem niezdominowanym w sensie relacji $\chi^{\preceq}$ [17]. Przykładami ocen *P*-optymalnych na rysunku 23 są punkty P_4 i P_5. W praktyce obliczeniowej zbiór rozwiązań sprawnych $\boldsymbol{X}_{\preceq}^{\boldsymbol{ND}}$ może być bardzo liczny, dlatego wskazane jest wyznaczenie reprezentacji $\boldsymbol{X}^{\boldsymbol{sel}} \subset \boldsymbol{X}_{\preceq}^{\boldsymbol{ND}}$, z której decydent może wybrać jedną konfigurację.

Zagadnienia wyznaczania reprezentacji konfiguracji *Pareto*-optymalnych dyskutowano w pracach [35, 88, 216, 366]. Sformułowane poniżej oryginalne zagadnienie optymalizacji dwukryterialnych różni się od problemów rozważanych w literaturze przedmiotu zarówno pod względem przyjętych kryteriów, jak i pod względem nałożonych ograniczeń na konfigurację dopuszczalną.

Dla danych wejściowych: $\mu_1, \mu_2, \mu_3, \mu_4, \mu_5$, V, I, J, $T = [t_{vj}]_{V \times J}$, $\tau = [\tau_{vuik}]_{V \times V \times I \times I}$, $\vartheta = [\vartheta_1, ..., \vartheta_j, ..., \vartheta_J]$, ϑ_{min}, $\xi = [\xi_1, ..., \xi_j, ..., \xi_J]$, $\xi_{\max}$, $\upsilon_{\min}$, $\tilde{D} = [\tilde{d}_{jr}]_{J \times 2}$, $C = [c_{vr}]_{V \times 2}$, $\varepsilon = [\varepsilon_1, ..., \varepsilon_j, ..., \varepsilon_J]$, $\varepsilon_{\max}$

należy wyznaczyć $\boldsymbol{X}^{\boldsymbol{sel}}$ reprezentację *Pareto*-optymalnych konfiguracji gridu w zagadnieniu optymalizacji wielokryterialnej w postaci uporządkowanej trójki:

$$(\tilde{\boldsymbol{X}}, \varphi, \chi^{\leq}), \tag{3.4.7}$$

1) $\tilde{\boldsymbol{X}}$ – zbiór konfiguracji dopuszczalnych

$$\tilde{\boldsymbol{X}} = \{x \in \boldsymbol{B}_{\text{bin}}^{I(V+J)} \mid \sum_{i=1}^{I} x_{vi}^{\alpha} = 1 \text{ dla } v = \overline{1,V};\ \sum_{j=1}^{J} x_{ij}^{\beta} = 1 \text{ dla } i = \overline{1,I};\ \Theta(x) \geq \vartheta_{\min} \text{ dla } \mu_1 \neq 0;$$

$$\Xi(x) \leq \xi_{\max} \text{ dla } \mu_2 \neq 0; \quad \upsilon(x) \geq \upsilon_{\min} \text{ dla } \mu_3 \neq 0;$$

$$\kappa_{ir}(x) \geq 0, \quad i = \overline{1,I},\ r = \overline{1,2} \text{ dla } \mu_4 \neq 0; \quad E(x) \leq \varepsilon_{\max} \text{ dla } \mu_5 \neq 0\}$$

2) φ – kryterium wektorowe

$$\varphi : \boldsymbol{X} \to \boldsymbol{R}^2,$$

gdzie $\varphi(x) = [\hat{Z}_{max}(x),\ \tilde{Z}_{max}(x)]^T = \omega \in \boldsymbol{R}^2$ dla $x \in \tilde{\boldsymbol{X}}$

3) $\chi^{\leq}$ – relacja dominowania w $\boldsymbol{R}^2$

$$\chi^{\leq} = \{(a,b) \in \boldsymbol{\Omega} \times \boldsymbol{\Omega} \mid a_n \leq b_n \text{ dla } n = \overline{1,2}\}, \text{ gdzie } \boldsymbol{\Omega} = \varphi(\tilde{\boldsymbol{X}}).$$

Zagadnienie optymalizacji wektorowej (3.4.7) cechuje się binarnymi zmiennymi decyzyjnymi. Przestrzeń przeszukiwań zawiera $2^{I(V+J)}$ elementów. Zbiór rozwiązań dopuszczalnych może być zbiorem pustym, przy czym dla pewnych instancji może zawierać $I^V J^I$ elementów. Można pokazać, że jest to problem NP-trudny.

Twierdzenie 3.2.

Jeśli zbiór konfiguracji dopuszczalnych $\tilde{\boldsymbol{X}}$ nie jest pusty, to zbiór wyników $\boldsymbol{\Omega} = \varphi(\tilde{\boldsymbol{X}})$ jest zbiorem skończonym i ograniczonym.

Dowód: Dla niepustego zbioru konfiguracji dopuszczalnych $\tilde{\boldsymbol{X}}$ jego obraz $\boldsymbol{\Omega} = \varphi(\tilde{\boldsymbol{X}})$ także nie jest zbiorem pustym. Ponieważ liczba kardynalna (liczność) zbioru $\tilde{\boldsymbol{X}}$ nie może być większa niż $I^V J^I$, to liczba kardynalna jego obrazu $\boldsymbol{\Omega} = \varphi(\tilde{\boldsymbol{X}})$ także nie może przekroczyć $I^V J^I$. Zatem $\boldsymbol{\Omega}$ jest zbiorem skończonym (o skończonej liczbie kardynalnej).

Obciążenie procesorów $\hat{Z}_{max}(x)$ w newralgicznym hoście ze względu na założenie o ograniczonych wartościach skumulowanych czasów wykonania modułów na komputerach, ma ograniczenie dolne oraz górne. Podobnie $\tilde{Z}_{max}(x)$ – obciążenie komunikacyjne newralgicznego hosta, ze względu na założenie o ograniczonych wartościach czasów przesyłania danych, cechuje się ograniczeniem dolnym oraz ograniczeniem górnym. Zatem zbiór $\boldsymbol{\Omega}$ jest ograniczony od dołu oraz od góry.

Zbiór wyników $\boldsymbol{\Omega}$ jest więc zbiorem skończonym i ograniczonym, co kończy dowód. □

Zagadnienia (3.4.7) można traktować jako podstawowe zagadnienie polioptymalizacji konfiguracji gridu, które podlega modyfikacjom w wypadku zmiany oczekiwań interesariuszy procesu decyzyjnego.

W szczególności można rozważać inne zagadnienie optymalizacji dwukryterialnej, np. $\varphi(x)=[Z_{max}(x), Z_{\text{suma}}(x)]^T$, w którym minimalizuje się obciążenie newralgicznego komputera oraz łączne obciążenia gridu. Ponadto można wykorzystać inne kryteria do adaptacyjnego formułowania problemu polioptymalizacji w zależności od preferencji decydenta. Przykładowo w optymalizacji z trzema kryteriami $[\hat{Z}_{max}, \tilde{Z}_{max}, Z_{\text{suma}}]$ minimalizuje się obciążenie procesorów w newralgicznym hoście, obciążenie komunikacyjne oraz łączne obciążenie gridu.

Poniżej sformułowano oryginalne zagadnienie optymalizacji wielokryterialnej uwzględniające scenariusz wyboru przez decydenta kombinacji pięciu kryteriów cząstkowych, za pomocą którego to wyboru konstruuje się funkcję wektorową do oceny jakości konfiguracji gridu. Rozważa się minimalizację czterech kryteriów: $\hat{Z}_{max}$, $\tilde{Z}_{max}, Z_{\text{suma}}$ i Ξ, a także maksymalizację M_1 – rezerwy pamięci RAM w gridzie.

Dla danych wejściowych: $\mu_1, \mu_2, \mu_3, \mu_4, \mu_5, V, I, J, T = \left[t_{vj}\right]_{V\times J}$, $\tau = [\tau_{vuik}]_{V\times V\times I\times I}$, $\vartheta = [\vartheta_1,...,\vartheta_j,...,\vartheta_J]$, ϑ_{min}, $\xi = [\xi_1,...,\xi_j,...,\xi_J]$, $\xi_{\max}$, $\upsilon_{\min}$, $\tilde{D} = \left[\tilde{d}_{jr}\right]_{J\times 2}$, $C = \left[c_{vr}\right]_{V\times 2}$, $\varepsilon = [\varepsilon_1,...,\varepsilon_j,...,\varepsilon_J]$, $\varepsilon_{\max}$

należy wyznaczyć $\boldsymbol{X}^{sel}$ reprezentację *Pareto*-optymalnych konfiguracji gridu w zagadnieniu optymalizacji wielokryterialnej w postaci uporządkowanej trójki:

$$(\tilde{\boldsymbol{X}}, \tilde{\varphi}, \chi^{\preceq}), \tag{3.4.8}$$

1) $\tilde{\boldsymbol{X}}$ – zbiór konfiguracji dopuszczalnych określony w zagadnieniu (3.4.7)

2) $\tilde{\varphi}$ – kryterium wektorowe

$$\tilde{\varphi}: \boldsymbol{X} \to \boldsymbol{R}^5$$

gdzie: $\tilde{\varphi}(x) = [\hat{Z}_{max}(x), \tilde{Z}_{max}(x), Z_{\text{suma}}(x), \Xi(x), -M_1(x)]^T \in \boldsymbol{R}^5$ dla $x \in \tilde{\boldsymbol{X}}$

3) $\chi^{\preceq}$ – relacja dominowania w $\boldsymbol{R}^5$

$$\chi^{\preceq} = \{(a,b) \in \boldsymbol{\Omega}\times\boldsymbol{\Omega} \mid a_n \le b_n \text{ dla } n = \overline{1,5}\}, \text{ gdzie } \boldsymbol{\Omega} = \tilde{\varphi}(\tilde{\boldsymbol{X}}).$$

NP-trudne zagadnienie optymalizacji wektorowej (3.4.8) cechuje się binarnymi zmiennymi decyzyjnymi. Przestrzeń przeszukiwań zawiera $2^{I(V+J)}$ elementów.

Można rozważać inny scenariusz wyboru kryteriów przez decydenta i wówczas należy sformułować adekwatne zagadnienie optymalizacji wielokryterialnej. W aplikacji ADAIORG'14 uwzględniono możliwość wyboru kryteriów cząstkowych spośród pięciu kryteriów, co jest prawdopodobnie pierwszą tak zaawansowaną implementacją w zakresie optymalizacji konfiguracji gridu [32, 366].

3.5. Zagadnienie wyznaczanie konfiguracji kompromisowych

Zbiór konfiguracji *Pareto*-optymalnych w zagadnieniach (3.4.7) i (3.4.8) jest na ogół bardzo liczny, a decydent chciałby wybrać jedną z nich do wdrożenia w gridzie. Pojedyncza konfiguracja pożądana jest także w wypadku automatycznej rekonfiguracji systemu. W tym wypadku na podstawie wyników uzyskanych za pomocą aplikacji ADAIORG'14 zasoby programistyczno-sprzętowe mogą być zreorganizowane. W szczególności moduły programistyczne mogą być przemieszczone do węzłów, do których przydzielono dodatkowe procesory, pamięć RAM czy pamięć dyskową.

Reorganizacja zasobów może być zaimplementowana także w chmurze obliczeniowej. Natomiast w klasycznych gridach wymagana jest nieco większa modyfikacja oprogramowania zarządzającego zasobami. Przykładowo w gridzie *Comcute* oprogramowanie modułów klasy *W* i *S* modyfikowane jest pod kątem osiągnięcia możliwości systemów agentowych, w których moduły programistyczne mogłyby przemieszczać się wśród zadanego zbioru hostów.

Interesującym podejściem do wyboru pojedynczego rozwiązania spośród alternatyw sprawnych jest optymalizacja kompromisowa [17]. Kompromis między kryteriami cząstkowymi $\varphi_1,...,\varphi_n,...,\varphi_N$ zależy od minimalizacji przyjętej miary odległości między oceną konfiguracji a punktem idealnym [252]. Zawężając rozważania do zagadnienia optymalizacji dwukryterialnej ($\widetilde{\boldsymbol{X}}$, φ, $\chi^{\leq}$) w postaci (3.4.7), można zdefiniować *punkt idealny* (*utopijny*) ω^{o} jako następującą ocenę [18]:

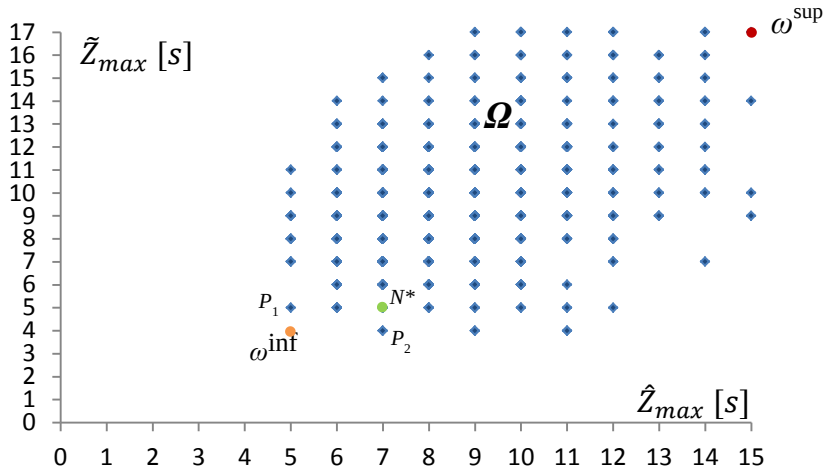
$$\omega^{\inf} = [\omega_1^{\inf},...,\omega_n^{\inf},...,\omega_N^{\inf}]^T, \tag{3.5.1}$$

taką że:

$$\omega_n^{\inf} = \inf_{x \in \widetilde{\boldsymbol{X}}} \varphi_n(x), \quad n = \overline{1,2}. \tag{3.5.2}$$

Punkt idealny jest celem, który nie zawsze może być osiągnięty za pomocą metod polioptymalizacji i zazwyczaj $\omega^{\inf}$ nie należy do zbioru ocen $\boldsymbol{\Omega}$ [32]. Jeśli $\omega^{\inf} \epsilon \boldsymbol{\Omega}$, to punkt idealny jest *oceną dominującą* w sensie relacji $\chi^{\leq}$ [17]. Taka sytuacja jest najbardziej pożądana, gdyż decydent akceptuje tylko jedną ocenę konfiguracji. Natomiast konfiguracji dominujących może być więcej, przy czym cechują się one identyczną oceną. Istnienie konfiguracji dominujących wyklucza istnienie konfiguracji *Pareto*-optymalnych i odwrotnie.

Na rysunku 24 przedstawiono przykładowy zbiór ocen $\boldsymbol{\Omega}$, dla którego zaznaczono punkt utopijny $\omega^{\inf}$, ocenę antyidealną $\omega^{\sup}$ [18], ocenę nadir N^* [17] oraz oceny *Pareto*-optymalne P_1, P_2. W tym wypadku punkt idealny nie należy do zbioru ocen.



Rys. 24. Charakterystyczne oceny konfiguracji gridu dla wybranej instancji zagadnienia polioptymalizacji (3.4.7)
Źródło: opracowanie własne.

Ocena idealna $\omega^{\inf} = [\omega_1^{\inf}, \omega_2^{\inf}]^T$ jest kresem dolnym zbioru ocen zgodnie z zależnością (3.5.2), a współrzędne punktu $\omega^{\inf}$ można oszacować teoretycznie. Można także dwukrotnie wyznaczyć rozwiązania suboptymalne za pomocą algorytmu harmonicznego o nazwie HSΔ_{max}, za pomocą którego wyznacza się konfiguracje suboptymalne dla zagadnienia (3.3.1) przy minimalizacji kryterium $\hat{Z}_{max}$, a następnie - $\tilde{Z}_{max}$. Ponieważ nie jest możliwe zagwarantowanie wyznaczenia dokładnego minimum dla dużych instancji zagadnienia (3.3.1), uzyskane wartości suboptymalne należy interpretować jako przybliżoną wartość współrzędnych punktu idealnego.

Teoretyczne oszacowanie od dołu $\omega_1^{\inf}$ opiera się na spostrzeżeniu, że najkorzystniejsza sytuacja z punktu widzenia minimalnego obciążenia $\hat{Z}_{max}$ dotyczy przydziału najdłuższej realizowanego modułu pod względem t_{vj} do najbardziej

wydajnego komputera. Odpowiada to konfiguracji, w której newralgiczny host obciążony jest tylko jednym modułem. Wprawdzie moduł ten cechuje się najdłuższym czasem wykonywania, ale i tak jest to najkorzystniejsza sytuacja, gdyż rozważany moduł wykonywany jest na najszybszym dla niego komputerze. Pozostałe moduły przedzielone są do pozostałych węzłów.

Wadą oszacowania teoretycznego jest fakt, że dla pewnych instancji przydział newralgicznego modułu do najbardziej wydajnego komputera nie gwarantuje, że ten host będzie najbardziej obciążony, Powodem jest przydział pozostałych modułów do nieobciążonych komputerów, który może spowodować, że obciążenie co najmniej jednego z komputerów będzie większe niż obciążenie najbardziej wydajnego komputera dla najdłużej wykonywanego modułu.

Rozpatrywana konfiguracja jest rozwiązaniem dopuszczalnym dla zagadnienia (3.1.1), ale może być konfiguracją niedopuszczalną dla problemu (3.3.1). Oszacowanie teoretyczne kresu dolnego $\hat{Z}_{max}$ wyznacza się zatem następująco:

$$\omega_1^{\inf} = \min_{j=\overline{1,J}} \max_{v=\overline{1,V}} \{ t_{vj} \}. \tag{3.5.3}$$

Wartość drugiej współrzędnej punktu idealnego $\omega_2^{\inf}$ wyznacza się poprzez minimalizację kryterium $\tilde{Z}_{max}$ w zadaniu optymalizacji jednokryterialnej z ograniczeniami (3.3.1). Kres dolny $\tilde{Z}_{max}$ można oszacować także za pomocą rozwiązania zagadnienia (3.1.1), w którym nie występują ograniczenia dodatkowe. Wówczas wartości wag w algorytmie HSΔ_{max} wynoszą odpowiednio 0 dla $\hat{Z}_{max}$ oraz 1 dla $\tilde{Z}_{max}$.

Teoretyczny sposób oszacowania kresu dolnego obciążenia komunikacyjnego jest następujący. Najkorzystniejsza sytuacja z punktu widzenia minimalizacji obciążenia komunikacyjnego $\tilde{Z}_{max}$ dotyczy przydziału wszystkich modułów do jednego węzła. Wówczas oszacowanie kresu dolnego $\tilde{Z}_{max}$ przyjmuje wartość zero, co zapisujemy dla porządku, jak niżej:

$$\omega_2^{\inf} = 0. \tag{3.5.4}$$

Do normalizacji przestrzeni kryterialnej wykorzystuje się także punkt antyidealny $\omega^{\inf}$, który wskazuje cele niepożądane w procesie optymalizacji. Współrzędne punktu antyidealnego w wypadku minimalizacji definiuje się, jak niżej:

$$\omega_n^{\sup} = \sup_{x \in \hat{\boldsymbol{X}}} \varphi_n(x), \quad n = \overline{1,2}. \tag{3.5.5}$$

Ponieważ zbiór ocen $\boldsymbol{\Omega}$ jest ograniczony od góry i od dołu (twr. 3.2) dla niepustego zbioru $\tilde{\boldsymbol{X}}$, to zachodzi:

$$\omega_n^{\inf} = \min_{x \in \tilde{\boldsymbol{X}}} \varphi_n(x), \quad n = \overline{1,2}. \tag{3.5.6}$$

$$\omega_n^{\sup} = \max_{x \in \tilde{\boldsymbol{X}}} \varphi_n(x), \quad n = \overline{1,2}. \tag{3.5.7}$$

W celu wyznaczenia współrzędnych punktu antyidealnego problemu ($\tilde{\boldsymbol{X}}$, φ, $\chi^{\leq}$), należy wyznaczyć maksimum $\hat{Z}_{max}$ dla (3.3.1), a następnie - maksimum $\tilde{Z}_{max}$. Najkorzystniejsza sytuacja z punktu widzenia maksymalizacji obciążenia $\hat{Z}_{max}$ dotyczy przydziału wszystkich modułów do najmniej wydajnego komputera. Teoretyczne oszacowanie od góry kresu górnego $\hat{Z}_{max}$ wyznacza się zatem następująco:

$$\omega_1^{\sup} = \hat{Z}_{\max}^{\max} = \sum_{v=1}^{V} \max_{j=1,J} \{ t_{vj} \}. \tag{3.5.8}$$

Wartość $\omega_2^{\sup}$ drugiej współrzędnej punktu antyidealnego wyznacza się poprzez rozwiązanie zadania (3.3.1) przy maksymalizacji $\tilde{Z}_{max}$. Kres górny $\tilde{Z}_{max}$ można oszacować także za pomocą algorytmu HSΔ_{max} w odniesieniu do zmodyfikowanego zagadnienia (3.1.1), w którym maksymalizuje się obciążenie komunikacyjne.

Teoretyczny sposób oszacowania od góry kresu górnego obciążenia komunikacyjnego jest następujący. Najkorzystniejsza sytuacja z punktu widzenia maksymalizacji obciążenia komunikacyjnego $\tilde{Z}_{max}$ polega na takim rozproszeniu modułów między węzły, że wszystkie pary modułów komunikują się przez sieć, co zapisujemy, jak niżej:

$$\omega_2^{\sup} = \tilde{Z}_{\max}^{\max} = \sum_{v=1}^{V} \sum_{u=1}^{U} \max_{i=1,I;\, k=1,I} \{ \tau_{vuik} \}. \tag{3.5.9}$$

Odległość oceny $\omega \in \boldsymbol{\Omega}$ od oceny idealnej $\omega^{\inf}$ wyznaczana jest w przestrzeni znormalizowanej, gdyż wielkości obciążeń $\hat{Z}_{max}$ oraz $\tilde{Z}_{max}$ mogą znacząco się różnić. Miarą odległości jest wówczas norma $\left\| (\omega - \omega^{\inf}) \right\|$ [17]. Ponieważ druga współrzędna punktu idealnego może przyjmować wartość równą zero, to nie można stosować klasycznej normalizacji [18]. W tej sytuacji *normalizacja przedziałowa* jest bardziej adekwatna, a wartość *n*-tego znormalizowanego obciążenia wyznacza się następująco:

$$\bar{\varphi}_n(x) = \frac{\varphi_n(x) - \omega_n^{\inf}}{\omega_n^{\sup} - \omega_n^{\inf}}, \quad \omega_n^{\sup} > \omega_n^{\inf}, \; n = \overline{1,2}. \tag{3.5.10}$$

Znormalizowana ocena idealna $\overline{\omega}^{\inf}$ cechuje się współrzędnymi (0; 0), a znormalizowany punkt antyidealny $\overline{\omega}^{\sup}$ - (1; 1). Znormalizowana ocena $\overline{\omega}$ należy do hipersześcianu $[0, 1]^2$.

Konfiguracja kompromisowa $x^{komp} \in \tilde{\boldsymbol{X}}$ jest optymalna w sensie *Pareto*, a jej poszukiwanie zachodzi w znormalizowanej przestrzeni $\overline{\boldsymbol{\Omega}}$ wg tzw. *p-normy* ς_p [17]:

$$\varsigma_p(\overline{\omega}) = \left\| \overline{\omega} - \overline{\omega}^{\inf} \right\|_p = \left(\sum_{n=1}^{N} \left(\overline{\omega} - \overline{\omega}^{\inf} \right)^p \right)^{1/p}, \tag{3.5.11}$$

gdzie p – zadany parametr, p =1,2,...,

Konfiguracja kompromisowa $x^{komp} \in \tilde{\boldsymbol{X}}$ dla $\overline{\omega}^{komp} = \overline{\varphi}(x^{komp})$ minimalizuje *p*-normę ς_p, co można zapisać następująco:

$$\varsigma_p(\overline{\omega}^{komp}) = \inf_{\omega \in \boldsymbol{\Omega}} \varsigma_p(\overline{\omega}). \tag{3.5.12}$$

Konfiguracja Salukwadze to rozwiązania kompromisowe dla parametru $p = 2$, które cechują się tym, że znajduje się najbliżej oceny utopijnej pod względem odległości *Euklidesa* [18]. Znormalizowana ocena konfiguracji *Salukwadze* cechuje się następującą wartością *p-normy*:

$$\varsigma_2(\overline{\omega}^{Sal}) = \inf_{\omega \in \boldsymbol{\Omega}} \varsigma_2(\overline{\omega}). \tag{3.5.13}$$

Twierdzenie 3.3 [32]

Dla parametru p =2, minimalizacji kryteriów oraz normalizacji przedziałowej zachodzi, jak niżej:

$$\varsigma_2(\overline{\omega}) = \sqrt{\sum_{n=1}^{N} \left(\frac{\varphi_n(x) - \omega_n^{\min}}{\omega_m^{\max} - \omega_n^{\min}} \right)^2}. \tag{3.5.14}$$

W wypadku zadanego punktu idealnego i oceny antyidealnej oraz podstawiając $\omega_n=\varphi_n(x)$, odległość jest funkcją zależną od konfiguracji gridu, co można zapisać następująco:

$$\varsigma_2(x) = \sqrt{\sum_{n=1}^{2} \left(\frac{\varphi_n(x) - \omega_n^{\min}}{\omega_m^{\max} - \omega_n^{\min}} \right)^2}. \tag{3.5.15}$$

Problem wyznaczania x^{Sal} konfiguracji *Salukwadze* dla zadanego punktu idealnego oraz punktu antyidealnego, można sformułować w postaci zagadnienia optymalizacji jednokryterialnej, jak niżej.

Dla danych: $\mu_1, \mu_2, \mu_3, \mu_4, \mu_5, V, I, J, T = \left[t_{vj}\right]_{V\times J}, \tau = [\tau_{vuik}]_{V\times V\times I\times I}$,
$\vartheta = [\vartheta_1,...,\vartheta_j,...,\vartheta_J]$, ϑ_{min}, $\xi = [\xi_1,...,\xi_j,...,\xi_J]$, $\xi_{\max}$, $\upsilon_{\min}$, $\tilde{D} = \left[\tilde{d}_{jr}\right]_{J\times 2}$, $C = \left[c_{vr}\right]_{V\times 2}$,
$\varepsilon = [\varepsilon_1,...,\varepsilon_j,...,\varepsilon_J]$, $\varepsilon_{\max}$, , $\omega^{\inf}$, $\omega^{\sup}$
należy wyznaczyć x^{Sal}, takie że

$$\varsigma_2(x^{Sal}) = \min_{x\in \boldsymbol{X}} \varsigma_2(x), \tag{3.5.16}$$

gdzie $\tilde{\boldsymbol{X}}$ jest zbiorem rozwiązań dopuszczalnych zdefiniowanym w (3.4.7)

NP-trudne zagadnienie optymalizacji jednokryterialnej (3.5.16) cechuje się binarnymi zmiennymi decyzyjnymi. Cechą charakterystyczną rozpatrywanego problemu jest możliwość wyznaczenia konfiguracji kompromisowej dla p=2, które to rozwiązanie jest alternatywą *Pareto*-optymalną. Alternatywnym podejściem do wyznaczenia konfiguracji *Salukwadze* jest rozwiązanie zagadnienia optymalizacji wektorowej (3.4.7) lub (3.4.8), a następnie wybranie ze zbioru $\boldsymbol{X}^{\text{sel}}$ konfiguracji o najmniejszej odległości od punktu idealnego w znormalizowanej przestrzeni kryterialnej.

Warto podkreślić, że teoretyczne oszacowania punktu idealnego $\omega^{\inf} = \left[\min_{j=1,J}\max_{v=1,V}\left\{t_{vj}\right\}; 0\right]$ oraz oceny antyidealnej $\omega^{\sup} = \left[\hat{Z}_{\max}^{\max}, \tilde{Z}_{\max}^{\max}\right]$ są obliczane po wprowadzeniu macierzy $T = \left[t_{vj}\right]_{V\times J}$, $\tau = [\tau_{vuik}]_{V\times V\times I\times I}$ przez ADAIORG'14.

W rezultacie funkcja celu ma postać:

$$\xi_2(x) = \sqrt{\left(\frac{\hat{Z}_{\max}(x) - \min_{j=1,J}\max_{v=1,V}\left\{t_{vj}\right\}}{\hat{Z}_{\max}^{\max} - \min_{j=1,J}\max_{v=1,V}\left\{t_{vj}\right\}}\right)^2 + \left(\frac{\tilde{Z}_{\max}(x)}{\tilde{Z}_{\max}^{\max}}\right)^2}. \tag{3.5.17}$$

Zmieniając parametr p z 2 na 1 lub $p\to\infty$, zmienia się p-normę, a w konsekwencji funkcję celu w zagadnieniu (3.5.16). Konfiguracje kompromisowe mogą zatem różnić się między sobą dla różnych parametrów p. Warto zauważyć, że każda konfiguracja kompromisowa dla $p\in[1,\infty)$ jest efektywna. Natomiast jeśli $p\to\infty$, to rozwiązanie kompromisowe może nie być konfiguracją P-optymalną [17].

3.6. Wnioski i uwagi

Problemy optymalizacji konfiguracji gridu opierają się na minimalizacji ważonego obciążenia przetwarzania danych i ich komunikacji przy uwzględnieniu istotnych

ograniczeń. Agregacja obu obciążeń $\hat{Z}_{\max}$ oraz $\tilde{Z}_{\max}$ zachodzi w $\Delta_{\max}$ - ważonym obciążeniu newralgicznego hosta, co umożliwia sformułowanie zagadnienia optymalizacji jednokryterialnej.

Można także rozważać minimalizację łącznego obciążeniu gridu lub łącznego obciążenia newralgicznego komputera. Ponadto można maksymalizować wydajność gridu, minimalizować koszt realizacji modułów w gridzie, koszt zakupu hostów lub pobór mocy energii przez komputery gridu. Decydent może preferować maksymalizację rezerwy dostępnej wielkości pamięci RAM w gridzie lub rezerwy dostępnej pojemności dysków twardych. Maksymalizacji może podlegać także wielkość dostępnej pamięci RAM w newralgicznym komputerze lub rezerwa pamięci dyskowej w newralgicznym serwerze. Ponadto można rozważać maksymalizację dostępności gridu lub stopnia rozproszenia modułów.

Warto wspomnieć, że każde kryterium może być uwzględnione w adekwatnych ograniczeniach, co uwzględniono w ograniczeniach problemu (3.3.1) w odniesieniu do Θ, Ξ, υ, κ_{ir} oraz E. Mogą wystąpić także inne wymagania, takie jak żądanie nieprzekroczenia zadanego obciążenia procesorów newralgicznego hosta, zapewnienie odpowiedniego poziomu dostępności gridu, czy też nieprzekroczenie limitu związanego z zakupem komputerów. Istotna jest także kwestia ekologiczna związana z ograniczeniem poboru mocy energii elektrycznej.

W rozdziale sformułowano dwa oryginalne zagadnienia wyznaczania reprezentacji konfiguracji optymalnych w sensie *Pareto* z dwoma i pięcioma kryteriami. Zaproponowano metody oszacowania wartości współrzędnych punktu idealnego. Postawiono także problem wyznaczania kompromisowych konfiguracji *Salukwadze* dla dwóch kryteriów.

4. ALGORYTMY HARMONICZNE DO WYZNACZANIA OPTYMALNYCH KONFIGURACJI GRIDOWYCH

Zastosowanie algorytmów do optymalizacji w problemach inżynierskich nabrało tempa w ostatniej dekadzie. Pochodzące z początku lat czterdziestych ubiegłego wieku metody matematyczne, takie jak programowanie liniowe, programowanie nieliniowe czy programowanie dynamiczne są w stanie skutecznie wyznaczać optymalne rozwiązania globalne jedynie w szczególnych zagadnieniach, np. dla problemu liniowego czy zadania transportowego [280].

Niestety ich zakres zastosowań nie obejmuje klasy problemów NP-trudnych [116], w których rozwiązanie optymalne nie może być wyznaczone w czasie wielomianowym. W tych zagadnieniach przestrzeń przeszukiwań, a w konsekwencji czas rozwiązywania problemu rośnie wykładniczo wraz z rozmiarem danych wejściowych, co sprawia, że klasyczne metody nie mogą być stosowane [243].

4.1. Taksonomia metod optymalizacji

Ze względu na to, czy metoda gwarantuje wyznaczenie optimum globalnego, metody optymalizacji można podzielić na *dokładne* i *przybliżone* [2, 198, 355]. Metody przybliżone są zazwyczaj metaheurystykami wyznaczającymi rozwiązania przybliżone, którymi są alternatywy nieznacznie gorsze niż optimum w sensie funkcji celu. W literaturze przedmiotu tego rodzaju warianty nazywamy *suboptymalnymi* [9, 251].

Ze względu na zakres zastosowań metody optymalizacji można podzielić na:

- *uniwersalne*, które umożliwiają wyznaczać rozwiązań dla szerokiej klasy problemów optymalizacji [254];
- *dedykowane* dla konkretnych problemów [27].

Na rysunku 25 przedstawiono taksonomię metod optymalizacji. Metody uniwersalne i dokładne cechują się zazwyczaj nieakceptowalnym czasem obliczeń dla instancji o dużych rozmiarach [38]. Natomiast metody dedykowane i dokładne są trudne w opracowaniu dla różnorodnych zagadnień optymalizacji. Ponadto mają ograniczony zakres zastosowań. Zaletą ich jest krótki czas wyznaczania rozwiązań optymalnych [63, 256]. Podobnie jest z metodami dedykowanymi i przybliżonymi, które są wprawdzie szybkie, ale cechują się ograniczonym zastosowaniem i wyznaczają

zazwyczaj rozwiązania niskiej jakości. Ponadto metody tej klasy są trudne w opracowaniu [90].

Luo, Wu i *Yang* zwrócili uwagę na wady tradycyjnych metod optymalizacji: programowania liniowego, programowania dynamicznego i programowania nieliniowego [233]. Zasadnicze wady to: długi czas obliczeń, duża zajętość pamięci, a także konieczność linearyzacji modelu. Ponadto kłopotliwe jest wykorzystanie skomplikowanych instrumentów różniczkowych (hesjan lub jakobian) oraz konieczność starannego wyboru rozwiązania startowego [233].

Kryterium		*Czy metoda gwarantuje wyznaczenie optimum globalnego?*	
		dokładne	przybliżone
Zakres zastosowań metody	uniwersalne	• *Programowanie liniowe (algorytm simpleks);* • *Programowanie nieliniowe* *algorytm Newtona, algorytm GRG …* • *Programowanie dynamiczne;* • *Metoda podziału i ograniczeń;* • *Przeszukiwanie grafu wszerz lub przeszukiwanie grafu w głąb;* • *Metoda minimax …*	• *Metaheurystyki* ✓ *algorytmy ewolucyjne,* ✓ *algorytmy genetyczne;* ✓ ***algorytmy harmoniczne**;* ✓ *algorytmy kolonii mrówek;* ✓ *programowanie genetyczne;* ✓ *symulowane wyżarzanie;* ✓ *sztuczne sieci neuronowe;* ✓ *tabu search …*
	dedykowane	• *Metoda transportowa;* • *Specyficzne metody grafowe:* ✓ *najkrótszej trasy Dijkstry,* ✓ *maksymalnego przepływu Forda-Fulkersona;*	• *Specjalizowane heurystyki*: ✓ *wybór najbliższego sąsiada dla problemu komiwojażera;* ✓ *priorytet wykonania najkrótszego zadania w szeregowaniu zadań;* ✓ *algorytm A* do wyznaczania najkrótszej trasy;*

Rys. 25. Taksonomia metod optymalizacji jednokryterialnej [239]

Na tym tle atrakcyjne są metody uniwersalne i przybliżone, do których należą metaheurystyki [258]. *Metaheurystyki* to ogólne schematy algorytmów, które można dostosować do rozwiązywania praktycznie każdego problemu optymalizacji. Co więcej, metaheurystyki wyznaczają zazwyczaj rozwiązania wysokiej jakości, szczególnie po ich adaptacji, w której uwzględnia się wiedzę dotyczącą rozwiązywanego zagadnienia [250, 259].

Algorytmy metaheurystyczne są algorytmami sztucznej inteligencji, które samodzielnie uczą się, naśladując inteligentne procesy i zachowania zaobserwowane w społeczeństwie, przyrodzie, fizyce czy chemii [274, 305]. W wielu przypadkach, gdy instancje problemów są dużych rozmiarów, metaheurystyki uważane są za najbardziej

skuteczne metody optymalizacji [276]. Zaletą działania metaheurystyk jest przechodzenie przez lokalne optima często znajdujące się w obszarach odległych od rozwiązań globalnych, co można osiągnąć poprzez zbadanie całej przestrzeni przeszukiwania za pomocą inteligentnych losowych operatorów, np. mutacji w algorytmie genetycznym czy improwizacji w algorytmie harmonicznym [253, 278]. Warto jednak podkreślić, że metaheurystyki nie gwarantują wyznaczenia optimum globalnego w zadanym czasie [279, 335].

W metaheurystykach istotne jest osiągnięcie kompromisu podczas przeszukiwania przestrzeni możliwych rozwiązań między dwoma ważnymi aspektami procesu wyznaczania optimum: dywersyfikacją i intensyfikacją [275]. *Dywersyfikacja* odnosi się do randomizacji składników dodanych do deterministycznych wartości w celu zbadania przestrzeni przeszukiwania w sposób zróżnicowany [283]. *Intensyfikacja* oznacza poprawę istniejących częściowych rozwiązań przez penetrację sąsiedztwa takich rozwiązań, na przykład za pomocą mechanizmów elitaryzmu w algorytmie ewolucyjnym czy też przeszukiwania sąsiedztwa o zadanej liczebności w *tabu search* [244, 281].

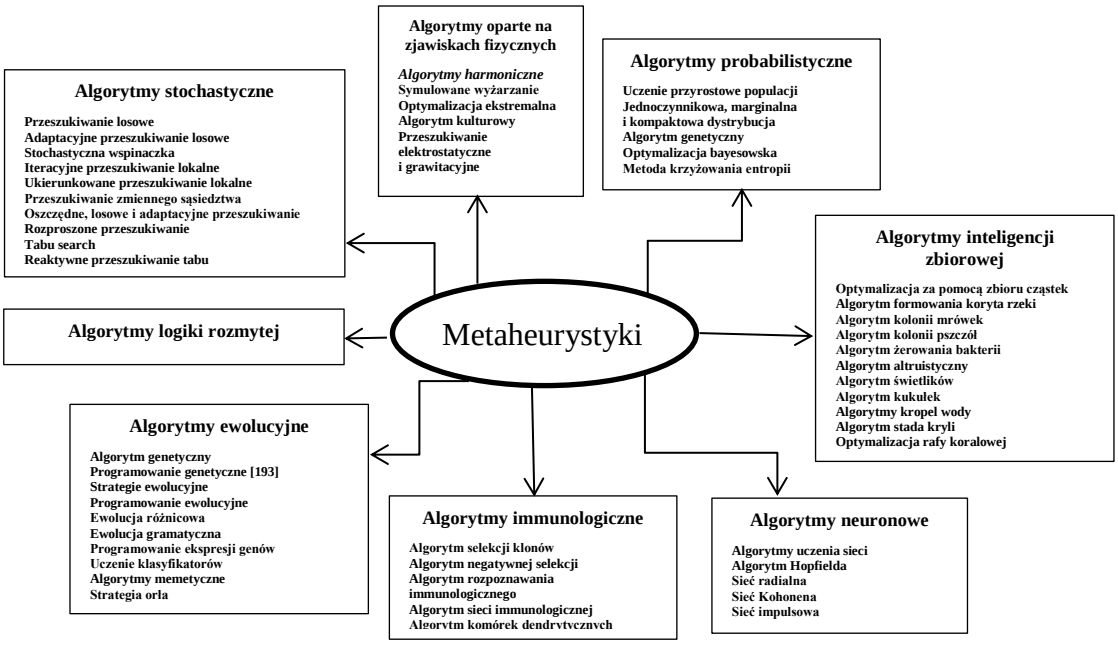
Jeśli zróżnicowanie przeszukiwania jest zbyt duże, to wysoka liczba penetrowanych obszarów może zmniejszyć szybkość zbieżności algorytmu [282]. Jeśli konstruując algorytm, zapewnia się niskie zróżnicowanie przeszukiwania, to istnieje znaczne ryzyko pozostawienia nieodkrytych obszarów lub nawet utknięcia w lokalnym optimum daleko od optymalnych rozwiązań [319]. Zasadnicze usprawnienia metaheurystyk odbywają się w wyniku wykorzystania wiedzy nabytej podczas eksploracji przestrzeni możliwych alternatyw oraz dostrajania parametrów, które sterują działaniem algorytmu, tak aby poprawić jego zbieżność [291, 317].

Na rysunku 26 przedstawiono klasyfikację metaheurystyk zaproponowaną przez *Manjarresa, Geema* i ich współpracowników [240]. Metaheurystyki podzielono na osiem klas na podstawie reguł działania.

4.2. Rola algorytmów harmonicznych i ich zastosowania

Algorytmy harmoniczne *HS* (ang. *harmony search*) należą do klasy metaheurystyk modelujących zjawiska fizyczne, w tym proces gry na instrumentach muzycznych [119, 324]. Warto podkreślić, że liczba publikacji w zakresie tych algorytmów dynamicznie zwiększyła się w ciągu ostatniej dekady [117-142]. Algorytm przeszukiwania

harmonicznego jest metaheurystyką inspirowaną procesem doboru najlepszego brzmienia podczas improwizacji muzyków jazzowych [331]. Koncepcja *harmony search* została zaproponowana przez *Zong Woo Geema* w 2000 roku [129]. Asumptem do skonstruowania algorytmu harmonicznego było spostrzeżenie, że muzycy jazzowi podczas improwizowanych sesji muzycznych wybierają *ad hoc* odpowiednie linie melodyczne, aby uzyskać najlepsze brzmienie podczas wykonywania utworu [136, 341-349].



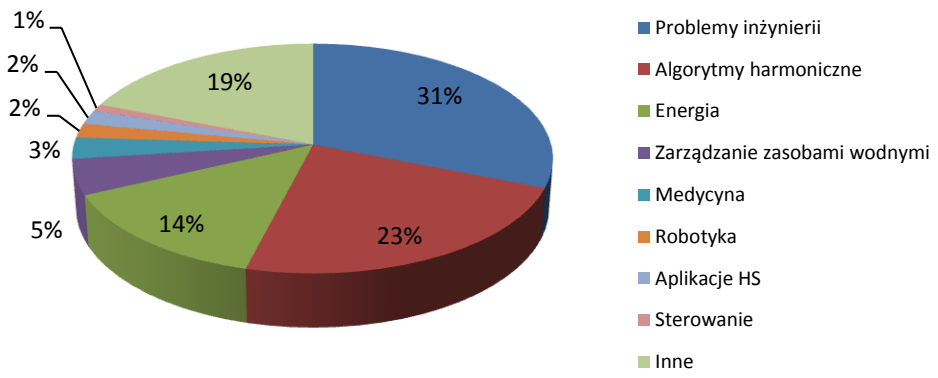
Rys. 26. Podstawowe rodzaje metaheurystyk [140]

Możemy intuicyjnie, aczkolwiek mało precyzyjnie, porównać utwór muzyczny do problemu optymalizacji, a wykonanie tego utworu do działania algorytmu harmonicznego [351]. Natomiast muzyk grający na instrumencie w zespole może odpowiadać zmiennej decyzyjnej, a nuta przez niego zagrana - wartości zmiennej decyzyjnej [237]. Najlepsze brzmienie uzyskane przez zespół można przyrównać do optimum globalnego problemu optymalizacji [232, 234].

Przegląd różnorodnych zastosowań algorytmu przeszukiwania harmonicznego zamieszczono w [3, 24, 240, 356]. Spektrum rozwiązanych do tej pory problemów optymalizacji za pomocą algorytmu harmonicznego jest bardzo szerokie (rys. 27). Najczęściej algorytmy te wykorzystywane są w zagadnieniach inżynierskich (31%) [363, 365, 368], a także energetyce (14%) [359, 360] czy zarządzaniu zasobami wodnymi (5%) [240]. Mniejsza liczba zastosowań dotyczy medycyny (3%) [377],

robotyki (2%) [361] i sterowania (1%). Ponadto aż 23% prac odnosi się do charakterystyki samych algorytmów harmonicznych z pominięciem ich aplikacyjności [240, 370, 374]. Analiza dotyczyła 160 artykułów indeksowanych w bazach danych: *Elsevier, IEEE* i *Springer* [240]. Powyższa analiza, a także analiza literatury przedmiotu przeprowadzona przez autora rozprawy wykazały, że nie są jeszcze dostępne publikacja z zakresu zastosowań algorytmu harmonicznego w gridach.

Do ciekawszych zastosowań inżynierskich algorytmu harmonicznego należy zaliczyć minimalizację kosztu wykonania konstrukcji stalowej za pomocą wyboru odpowiednich profili stalowych belek i kolumn [303]. Wymagania dotyczą ograniczeń związanych z przestrzeganiem reguł wykonania konstrukcji. Algorytmy harmoniczne zastosowano także do optymalizacji procesu produkcji stalowych wymienników ciepła pod kątem ich zdolności adaptacji do różnych warunków pracy [96-100].



Rys. 27. Zastosowania algorytmów harmonicznych [240]

Algorytmy HS zastosowano do optymalizacji bezprzewodowej sieci czujników pod kątem zwiększenia ich żywotności i zasięgu [235]. Uzyskano odpowiednie rozmieszczenie czujników w celu minimalizacji zużycia energii, zachowując łączność między węzłami i zapewniając własność pokrycia obszarów punktów dostępowych.

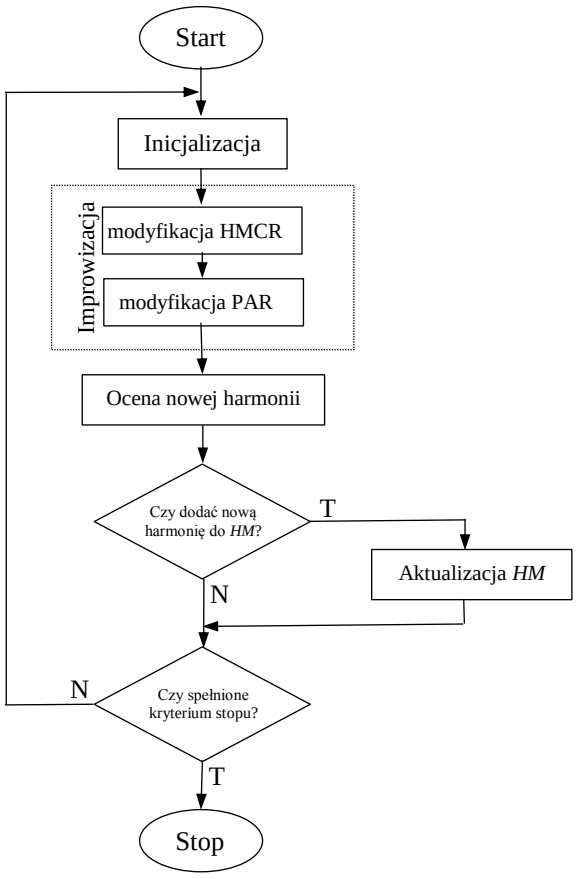
Natomiast *Geem* i *Cho* zastosowali algorytm harmoniczny do zaprojektowania sieci wodociągowej, minimalizując koszt jej budowy [126]. Rozważania kontynuowano w licznych pracach [28, 29, 115].

Ayob et al. wykorzystali algorytm harmoniczny do wyznaczania harmonogramów dyżurów pielęgniarek [26]. Ponadto, algorytm zastosowano do wyznaczania tras

pojazdów [137]. Interesujące aspekty licznych zastosowań algorytmu HS opisano w pracach [143, 144, 149, 150, 375, 376].

4.3. Charakterystyka algorytmów harmonicznych

Generowanie nowych rozwiązań na podstawie pamięci harmonicznej (ang. *a harmony memory HM*) przypomina działanie algorytmu symulowanego wyżarzania [44, 107]. Z drugiej strony podobieństwo między pamięcią harmoniczną a populacją nawiązuje do algorytmu genetycznego, przy czym improwizacja odpowiada mutacji [22, 108]. Z kolei wybór z pamięci harmonicznej ma cechy krzyżowania genetycznego, przy czym w algorytmie harmonicznym brana jest pod uwagę cała pamięć *HM*, a nie jedynie dwa osobniki tak jak w algorytmie genetycznym [10, 110]. Diagram podstawowej wersji algorytmu przeszukiwania harmonicznego zamieszczono na rysunku 28 [240].



Rys. 28. Diagram podstawowej wersji algorytmu harmonicznego do optymalizacji jednokryterialnej [240]

Niech zadane będą dwa wektory ograniczeń dla ciągłych zmiennych decyzyjnych: wektor ograniczeń dolnych $l = [l_1, \dots, l_j, \dots, l_{J_{max}}]^T$ oraz wektor ograniczeń górnych

$u = [u_1, \ldots, u_j, \ldots, u_{J_{max}}]^T$, przy czym $l_j \in \boldsymbol{R}$, $u_j \in \boldsymbol{R}$, $l_j \leq u_j$ dla $j = \overline{1, J_{max}}$. Podstawowa wersja algorytmu harmonicznego umożliwia wyznaczenie pojedynczego rozwiązania dla jednokryterialnego zagadnienia optymalizacji z ciągłymi zmiennymi decyzyjnymi, które to zagadnienie sformułowano, jak niżej [112]:

$$\min_{x \in \boldsymbol{X}} f(x), \tag{4.3.1}$$

gdzie:

$f(x)$ – wartość funkcji celu dla rozwiązania $x \in \boldsymbol{X}$, $f: \boldsymbol{R}^{J_{max}} \rightarrow \boldsymbol{R}$;

x – wektor zmiennych decyzyjnych, $x=[x_1,\ldots,x_j,\ldots,x_{J_{max}}]^T$, przy czym spełnione powinny być ograniczenia $l_j \leq x_j \leq u_j$, $j= \overline{1, J_{max}}$;

J_{max} – zadana liczba zmiennych decyzyjnych;

$\boldsymbol{X}$ – zbiór rozwiązań dopuszczalnych.

Inicjalizacja pamięci harmonicznej (rys. 28) zachodzi po wprowadzeniu danych oraz ustaleniu następujących parametrów algorytmu harmonicznego [163]:

- *HMS* (ang. *Harmony Memory Size*) – wielkość pamięci harmonicznej;
- *HMCR* (ang. *Harmony Memory Considering Rate*) – prawdopodobieństwo wystąpienia zdarzenia losowego, że wartość zmiennej decyzyjnej podczas improwizacji (konstruowania rozwiązania) losowana jest ze wszystkich wartości z pamięci *HM*; zakłada się rozkład równomierny losowania, $HMCR \in [0,1]$;
- *PAR* (ang. *Pitch Adjusting Rate*) – tempo modyfikacji wylosowanej zmiennej decyzyjnej z pamięci *HM*, $PAR \in [0; 1]$;
- $NG_{\max}$ (ang. *Number of Generations (Improvisations)*) – maksymalna liczba generacji (improwizacji, iteracji) weryfikowana za pomocą kryterium stopu;
- *BW* (ang. *Bandwidth of Generations*) – szerokość przedziału modyfikacji wartości zmiennej decyzyjnej wylosowanej z pamięci *HM*, przy czym nowa wartość zmiennej decyzyjnej jest modyfikowana o wartość z przedziału $[-BW, BW]$;

W pamięci *HM* przechowuje się *HMS* losowo wygenerowanych rozwiązań o J_{max} współrzędnych rzeczywistych oraz odpowiadające im wartości funkcji sprawności *fitness*(x) [166]. W wypadku minimalizacji funkcji celu maksymalizowana sprawność rozwiązania jest wartością funkcji celu ze znakiem przeciwnym. Jeśli nałożone są ograniczenia na rozwiązania, to jego sprawność zmniejszana jest o odpowiednią karę w wypadku naruszenia ograniczeń. Sprawność każdego rozwiązania może być zwiększona o pewną wartość tak, aby przyjmowała wartości nieujemne [169].

Pamięć algorytmu harmonicznego jest zazwyczaj modelowana jako macierz o *HMS* wierszach i J_{max}+1 kolumnach. Podczas inicjalizacji pamięci *HM* współrzędne zmiennych decyzyjnych generowane są losowo według zależności, jak niżej [172]:

$$x_j^i = l_j + r_{2j}(u_j - l_j), \quad i = \overline{1, HMS}, \quad j = \overline{1, N}, \tag{4.3.2}$$

gdzie:

i – numer wiersza (jednocześnie numer rozwiązania) w pamięci *HM*;

r_{2j} – liczba losowa z przedziału [0; 1] do modyfikacji j-tej zmiennej decyzyjnej.

Pamięć *HM* reprezentowana jest zatem przez następującą macierz [176]:

$$HM = \begin{bmatrix} x_1^1 & \dots & x_j^1 & \dots & x_{J_{max}}^1 & fitness(x^1) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x_1^i & \dots & x_j^i & \dots & x_{J_{max}}^i & fitness(x^i) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x_1^{HMS} & \dots & x_j^{HMS} & \dots & x_{J_{max}}^{HMS} & fitness(x^{HMS}) \end{bmatrix}$$

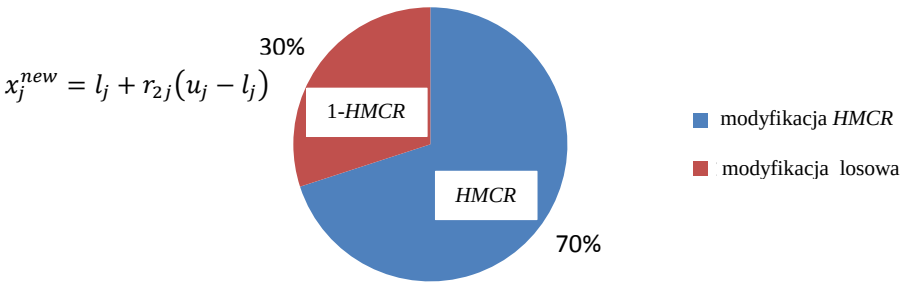
Improwizacja w algorytmie harmonicznym polega na wyznaczeniu nowego rozwiązania $x^{new} = [x_1^{new}, \dots, x_j^{new}, \dots, x_{J_{max}}^{new}]^T$, które może być wprowadzone do pamięci harmonicznej [179]. W tym celu dla każdej zmiennej decyzyjnej w wektorze x^{new} zostaje wyznaczona nowa wartość za pomocą trzech dywersyfikujących modyfikacji z wykorzystaniem parametrów *HMCR* i *PAR* [180].

Parametr *HMCR* to prawdopodobieństwo wystąpienia zdarzenia losowego, że wartość zmiennej decyzyjnej do improwizacji jest losowana ze wszystkich dźwięków znajdujących się w pamięci *HM*. Zakłada się rozkład równomierny losowania [11]. W przeciwnym razie, czyli z prawdopodobieństwem 1-*HMCR*, wartości losowane są ze zbioru wszystkich możliwości, także i tych, których nie ma w pamięci [4]. Improwizacja z parametrem *HMCR* zwiększa różnorodność rozwiązań, co sprzyja osiągnięciu globalnego optimum [7, 12, 181].

Modyfikacja z parametrem *HMCR* polega na tym, że dla każdej j-tej zmiennej decyzyjnej generuje się liczbę losową r_{1j} z przedziału [0; 1], a następnie bada się, czy $r_{1j} \leq HMCR$. Jeśli nierówność nie jest spełniona, to nowa wartość j-tej zmiennej x_j^{new} wyznaczana jest losowo zgodnie z poniższą zależnością [184]:

$$x_j^{new} = l_j + r_{2j}(u_j - l_j). \tag{4.3.3}$$

W powyższym wypadku nie jest wykorzystywana pamięć *HM*. Zatem im wyższa wartość parametru *HMCR*, tym mniejszy wpływ czynnika losowego na wartości nowej improwizacji x^{new}, a większy - pamięci harmonicznej [182]. Powyższą regułę proponuje się nazwać *regułą binarnej ruletki*, dla której obwód podzielono na dwie części zgodnie z parametrem *HMCR* (rys. 29).



Rys. 29. Reguła binarnej ruletki z parametrem *HMCR*=0,7
Źródło: Opracowanie własne.

Jeśli $r_{1j} \leq HMCR$, to losowany jest numer wiersza k z macierzy *HM* zgodnie z rozkładem jednostajnym [185]. Następnie weryfikuje się z wykorzystaniem improwizacji typu *PAR*, czy *j*-ta zmienna decyzyjna x_j^k z wektora zmiennych decyzyjnych nr k może być nową współrzędną x_j^{new} w konstruowanym wektorze improwizacji x^{new}. W tym wypadku wykorzystuje się zawartość pamięci *HM* [13, 14].

Jeśli wygenerowana kolejna liczba losowa spełnia nierówność $r_{3j} \leq PAR$, to zachodzi improwizacja z losowym wykorzystaniem pamięci *HM*, jak niżej [37, 373]:

$$x_j^{new} = x_j^k + r_{4j} BW, \tag{4.3.4}$$

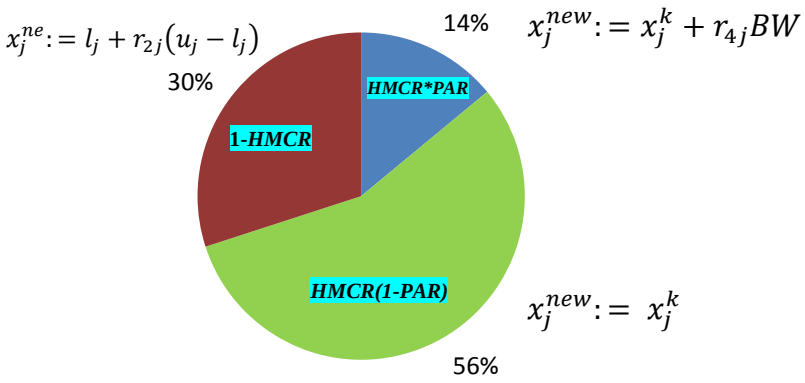
gdzie r_{4j} jest liczbą losową z przedziału [-1; 1].

Parametr BW wyznacza szerokość przedziału modyfikacji wartości zmiennej decyzyjnej x_j^k wylosowanej z pamięci *HM* [15, 16]. Nowa wartość zmiennej decyzyjnej modyfikowana jest o wartość wylosowaną z przedziału [$-BW$; BW]. Jeśli nowa wartość $x_j^{new} > u_j$, to podstawia się $x_j^{new} := u_j$. Natomiast w wypadku przekroczenia dolnego ograniczenia $x_j^{new} < l_j$, zachodzi podstawienie $x_j^{new} := l_j$ [25, 190].

Jeśli jednak $r_{3j} > PAR$, to zachodzi podstawienie $x_j^{new} := x_j^k$ [50]. Powyższa modyfikacja zmiennej decyzyjnej jest ukierunkowana na wykorzystanie niezmienionej informacji z pamięci harmonicznej. Odpowiada to sytuacji, w której muzyk dobierając dźwięk, korzysta wyłącznie z zagranych wcześniej przez siebie dźwięków [52, 64]. Im większa wartość parametru *PAR*, tym większe prawdopodobieństwo zaakceptowania w nowej improwizacji wartości wylosowanej zmiennej x_j^k z pamięci *HM* [51, 53].

Realizację obu modyfikacji typu *HMCR* i *PAR* podczas improwizacji można przedstawić za pomocą *reguły trójwartościowej ruletki* (rys. 30), za pomocą której możliwe są trzy sposoby wyznaczania nowej wartości zmiennej decyzyjnej

w konstruowanej alternatywie. Jeśli $r_{1j} \geq HMCR$, to nowa wartość j-tej zmiennej x_j^{new} wyznaczana jest zgodnie z zależnością (4.3.3) [56, 111]. W przeciwnym razie bada się, czy $r_{3j} \leq PAR$. Jeśli nierówność jest spełniona, to x_j^{new} wyznaczana jest zgodnie z zależnością (4.3.4). Jeśli $r_{3j} > PAR$, to $x_j^{new} := x_j^k$ [83, 86].



Rys. 30. Reguła trójwartościowej ruletki dla *HMCR*=0,7 oraz *PAR*=0,2
Źródło: Opracowanie własne.

Dla liczb losowych r_{1j}, r_{2j}, r_{3j}, r_{4j} procedurę improwizacji w odniesieniu do j-tej zmiennej decyzyjnej można zapisać za pomocą następującej zależności [196]:

$$x_j^{new} := \begin{cases} l_j + r_{2j}(u_j - l_j) & \text{dla } r_{1j} > HMCR; \\ x_j^k + r_{4j}BW & \text{dla } r_{1j} \leq HMCR, r_{3j} \leq PAR; \\ x_j^k & \text{dla } r_{1j} \leq HMCR, r_{3j} > PAR. \end{cases} \quad (4.3.5)$$

Improwizacja losowa z pamięcią dla $r_{1j} < HMCR$, $r_{3j} < PAR$ może być scharakteryzowana za pomocą alternatywnej zależności do (4.3.4), jak niżej [57, 76]:

$$x_j^{new} := x_j^k + (r_{5j} - r_{6j})BW|u_j - l_j|, \quad (4.3.6)$$

gdzie r_{5j}, r_{6j} są liczbami losowymi z przedziału [0; 1].

Warto podkreślić, że wybór losowy zwiększa dywersyfikację i w konsekwencji stwarza możliwość odnalezienia optimum globalnego, ponieważ obszarem poszukiwań jest cały dostępny zakres rozwiązań [62, 358]. Natomiast podczas wyboru x_j^k z pamięci *HM* numer rozwiązania *k* losowany jest zgodnie z rozkładem równomiernym [63, 197]. Alternatywną metodą selekcji losowej może być reguła ruletki z uwzględnieniem wartości *fitness*, którą to regułę stosuje się w algorytmach genetycznych [69, 199].

Po dokonaniu oceny nowej harmonii, bada się czy wyznaczona improwizacja x^{new} jest lepsza pod względem wartości *fitness* od najgorszej alternatywy w pamięci

harmonii [158], Jeśli tak, to x^{new} zastępuje najgorsze rozwiązanie z archiwum *HM*. Jeśli nie, to aktualizacja pamięci nie zachodzi [177, 200].

Kryterium stopu polega na sprawdzeniu czy numer improwizacji *NG* jest równy założonemu maksimum $NG_{\max}$ lub przekroczony został maksymalny czas obliczeń [164, 207]. Jeśli tak, to algorytm kończy działanie. Jeśli nie, to proces improwizacji jest kontynuowany [65-67].

Na tym tle interesujące są wyniki porównania algorytmu harmonicznego z innymi algorytmami [85, 94, 208]. *Kim et al.* eksperymentalnie wykazali, że za pomocą algorytmu harmonicznego uzyskano wyniki wyższej jakości niż za pomocą algorytmu genetycznego w odniesieniu do problemu optymalizacji sieci wodociągowej [187]. Natomiast autor rozprawy potwierdził tą hipotezę eksperymentalną w odniesieniu do optymalizacji wybranych funkcji matematycznych rozważanych w pracy *de Jonga* [78].

Na rysunku 31 przedstawiono wyniki eksperymentów numerycznych przeprowadzonych w odniesieniu do testowych zagadnień optymalizacji liniowej i nieliniowej z ciągłymi zmiennymi decyzyjnymi. Na rysunku 31a) zaprezentowano minimalizację funkcji celu w zagadnieniu programowania liniowego, jak niżej:

$$\min \quad \sum_{j=1}^{5} x_j, \tag{4.3.7}$$

przy ogr.

$$1 \le x_1 \le 4\ ;\ 4 \le x_2 \le 7\ ;\ 2 \le x_3 \le 3\ ;\ 2 \le x_4 \le 3\ ;\ 1 \le x_5 \le 2\ ;\ x_1, x_2, x_3, x_4, x_5 \in \boldsymbol{R};$$

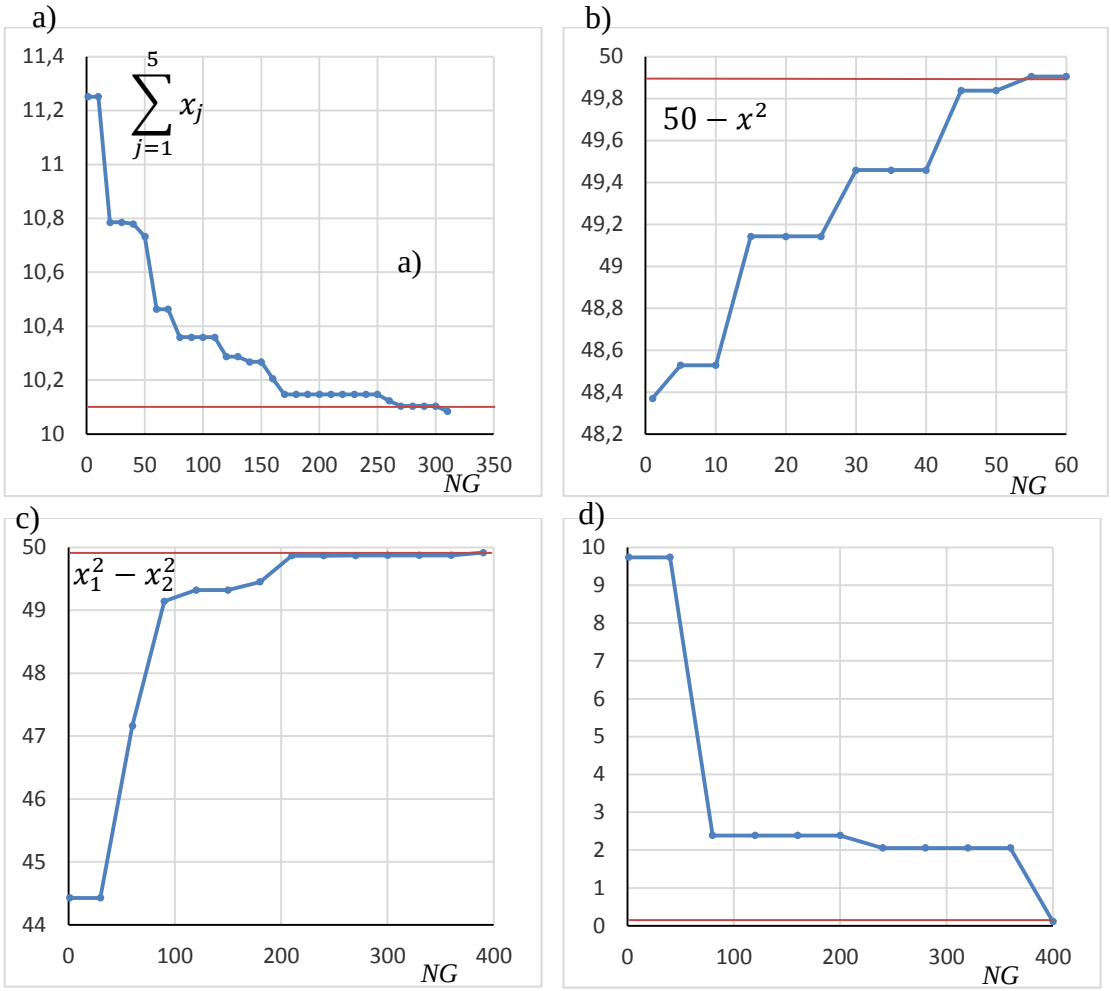
Optymalna wartość funkcji celu wynosi 10 dla $x=[1, 4, 2, 2, 1]^T$. Algorytm harmoniczny wyznaczył rozwiązanie suboptymalne z założoną dokładnością 0,1 po 310 improwizacjach, które to rozwiązanie cechuje się wartością funkcji celu 10,08. Czas obliczeń wyniósł 0,047 s na komputerze klasy PC z procesorem *Intel Core 2 Duo*. W algorytmie harmonicznym przyjęto: *HMCR*=0,8; *PAR*=0,2; *BW*=0,2; *HMS*=10.

Natomiast za pomocą algorytmu ewolucyjnego firmy *Frontline Systems, Inc* [404] wyznaczono wartość funkcji celu 12,216 dopiero po 10 s. Z kolei algorytm *simpleks* [404] wyznaczył w porównywalnym czasie do czasu działania algorytmu harmonicznego dokładne rozwiązanie optymalne.

Na rysunku 31b) zaprezentowano maksymalizację funkcji celu w poniższym zagadnieniu programowania nieliniowego, które rozpatrywano w pracy *de Jonga* [78].

$$\min \quad (50 - x^2), \tag{4.3.8}$$

$$\text{przy ogr. } -5 \le x \le 5\ ;\ x \in \boldsymbol{R};$$



Rys. 31. Zbieżność algorytmu harmonicznego do optimum z zadaną dokładnością dla czterech instancji
Źródło: Opracowanie własne.

Optymalna wartość funkcji celu wynosi 50 dla x^*=0. Algorytm genetyczny w wersji *De Jonga* [78] wyznaczył wartość funkcji celu 49,87 w 62 iteracji, ale już podczas dalszych obliczeń nie poprawił tego rozwiązania. Natomiast algorytm harmoniczny wyznaczył rozwiązanie suboptymalne po 55 improwizacjach z dokładnością do 0,1. Wartość funkcji celu wynoszącą 49,91 wyznaczono dla parametrów: *HMCR*=0,8; *PAR*=0,2; *BW*=0,2; *HMS*=10. Po 1000 improwizacjach otrzymano rozwiązanie suboptymalne z dokładnością do 0,000001 w czasie 0,05 s.

Z kolei nieliniowy algorytm *uogólnionego zredukowanego gradientu* GRG firmy *Frontline Systems, Inc* [403] wyznaczył w porównywalnym czasie do czasu działania algorytmu harmonicznego optymalną wartość funkcji celu z dokładnością do 0,000001.

Na rysunku 31c) zaprezentowano maksymalizację funkcji celu w kolejnym testowym zagadnieniu programowania nieliniowego, jak niżej [78]:

$$\min \quad (x_1^2 - x_2^2), \tag{4.3.9}$$
$$\text{przy ogr. } -5 \leq x_1^2 \leq 5; -5 \leq x_2^2 \leq 5; x_1 , x_2 \in \boldsymbol{R};$$

Optymalna wartość funkcji celu wynosi 50 dla czterech rozwiązań: (-5; -5), (-5, 5), (5; -5), (5, 5). Algorytm genetyczny *De Jonga* wyznaczył wartość funkcji celu 49,91 po 617 iteracjach, a następnie kontynuując obliczenia aż do iteracji nr 1000 poprawił tę sprawność jedynie o 0,004 [78]. Natomiast algorytm harmoniczny wyznaczył po 390 improwizacjach rozwiązanie suboptymalne z dokładnością do 0,1. Wartość funkcji celu wynoszącą 49,92 wyznaczono dla parametrów: *HMCR*=0,8; *PAR*=0,2; *BW*=0,2; *HMS*=10. Po 1000 improwizacjach wyznaczono rozwiązanie z dokładnością do 0,001 w czasie 0,05 s. Z kolei algorytm GRG wyznaczył w porównywalnym czasie optymalną wartość funkcji celu z dokładnością do 0,0001.

Na rysunku 31d) zaprezentowano minimalizację funkcji celu w nieco bardziej złożonym zagadnieniu programowania nieliniowego, jak niżej [78]:

$$\min \quad 100(x_2 - x_1^2)^2 + (1 - x_1)^2, \tag{4.3.10}$$
$$\text{przy ogr. } -10 \leq x_1^2 \leq 10; -10 \leq x_2^2 \leq 10; x_1 , x_2 \in \boldsymbol{R};$$

Optymalna wartość funkcji celu wynosi 0 dla rozwiązania (1; 1). Algorytm harmoniczny wyznaczył po 400 improwizacjach rozwiązanie z dokładnością do 0,1. Dostrojono podstawowe parametry następująco: *HMCR*=0,4; *PAR*=0,6; *BW*=0,2; *HMS*=10. Po 2000 improwizacjach wyznaczono rozwiązanie z dokładnością do 0,01 w czasie 0,05 s. Zaskakująco gorszy wynik uzyskano za pomocą algorytmu GRG, który potrzebował aż prawie 8 s do wyznaczenia porównywalnej klasy wyników. Natomiast algorytm genetyczny wyznaczył minimalną wartość funkcji celu 0,02 po ok. 70 s.

Na podstawie uzyskanych wyników eksperymentów numerycznych, stwierdza się, że algorytm harmoniczny przewyższa algorytm genetyczny pod względem jakości wyznaczanych rozwiązań oraz czasu obliczeń. Natomiast w porównaniu do dokładnych algorytmów: algorytmu *simpleks* i algorytmu programowania nieliniowego GRG, algorytm harmoniczny wyznacza rozwiązania zbliżonej jakości w zadanym czasie obliczeń. Warto jednak podkreślić, że algorytm harmoniczny jest uniwersalny i może być zastosowany do różnorodnych problemów optymalizacji, podczas gdy algorytm simpleks czy GRG są dedykownane tylko do wybranych zagadnień [81, 82, 210].

4.4. Algorytm harmoniczny do jednokryterialnej optymalizacji konfiguracji gridowych

Do jednokryterialnej optymalizacji konfiguracji gridowych zagadnień (3.1.1), (3.2.1) oraz (3.3.1) opracowano wersję algorytmu harmonicznego oznaczonego jako HSΔmax, w którym wprowadzono szereg istotnych modyfikacji w odniesieniu do jego podstawowej wersji [105, 114, 211], m.in. kodowanie całkowitoliczbowe konfiguracji oraz zmodyfikowaną procedurę improwizacji.

W celu zredukowania przestrzeni przeszukiwań zastosowano wektor przydziału modułów X^{α}. Ponadto wykorzystano wektor przydziału rodzajów komputerów X^{β}. W ten sposób zapewnia się spełnienie ograniczeń w zagadnieniu (3.1.1), a w innych zagadnieniach optymalizacji przestrzeń przeszukiwań jest redukowana do $I^V J^I$ konfiguracji.

W kryterium stopu bada się dodatkowo czy nie zaszła przedwczesna zbieżność algorytmu, co może się zdarzyć, kiedy stan obliczeń utknie w maksimum lokalnym funkcji sprawności [113, 212]. Jeśli podczas NG_{lok} iteracji nie poprawiono optimum, to generowana jest nowa pamięć *HM* niezbędna do kontynuowania obliczeń. 20% najlepszych rozwiązań z poprzedniej *HM* pozostaje w nowej *HM*. Ponadto 40% rozwiązań konstruowanych jest za pomocą genetycznej operacji krzyżowania. Rozwiązania losowane są ze „starej" pamięci zgodnie z regułą ruletki. Pozostałe 40% rozwiązań generowanych jest losowo zgodnie z poniższą zależnością:

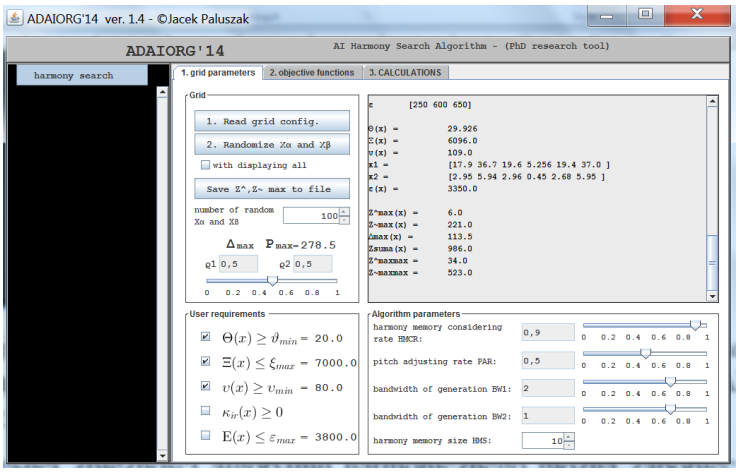
$$x_m^{new} = \begin{cases} \lfloor 1 + r_{2m}(I-1) + 0{,}5 \rfloor & \text{dla } m = \overline{1,V} \\ \lfloor 1 + r_{2m}(J-1) + 0{,}5 \rfloor & \text{dla } m = \overline{V+1, V+I} \end{cases} \tag{4.4.1}$$

gdzie:

$r_{2m} \in [0;1]$;

$\lfloor x \rfloor$ – zaokrąglenie w dół liczby rzeczywistej do liczby naturalnej, np. $\lfloor 3{,}2 \rfloor = 3$.

Algorytm HSΔmax zaimplementowano w aplikacji ADAIORG'14, której interfejs graficzny przedstawiono na rysunku 32. Dane wejściowe do problemu optymalizacji gridu wczytywane są z pliku tekstowego za pomocą przycisku „*1 Read grid config"*. Pamięć harmoniczna *HM* wyznaczana jest po wciśnięciu przycisku „*2 Randomize Xα i Xβ"*. Spośród wygenerowanych w sposób losowy $HMS_{\max}$ konfiguracji wybiera się *HMS* elementów o najwyższej wartości *fitness*. Na rysunku 32 $HMS_{\max}$ =100 oraz *HMS*=10. Parametry algorytmu harmonicznego ustawiane są w sekcji „*Algorithm parameters*", a zalecane wartości to *HMCR*=0,9 oraz *PAR*=0,5.



Rys. 32. Interfejs graficzny implementacji algorytmu harmonicznego *HSΔmax* do optymalizacji jednokryterialnej konfiguracji gridu w aplikacji ADAIORG'14 Źródło: opracowanie własne.

Przedwczesnej zbieżności algorytmu harmonicznego można zapobiec za pomocą dodatkowego warunku wprowadzenia improwizacji do tablicy harmonicznej. Jeśli wartość *fitness* skonstruowanej improwizacji jest wyższa niż najmniejsza wartość *fitness* „najgorszej" konfiguracji w tablicy, to sprawdza się, czy w tablicy istnieje identyczna konfiguracja jak improwizacja. Jeśli nie, to improwizacja wprowadzana jest do tablicy harmonicznej. Jeśli już istnieje taka alternatywa, to improwizacja nie jest wprowadzana, gdyż w przeciwnym razie tablica może zawierać dwa, trzy, a z czasem więcej identycznych elementów.

Ważnym parametrem algorytmu harmonicznego jest *BW*, który reprezentuje połowę szerokości przedziału modyfikacji zmiennej decyzyjnej [213, 310, 318]. Istotnym rozszerzeniem w proponowanych algorytmach jest wprowadzenie dwóch parametrów *BW1* i *BW2*, za pomocą których z prawdopodobieństwem (1-*HMCR*)*PAR* modyfikowane są zmienne decyzyjne. W wypadku zmiennej określającej węzeł wykonywania modułu wykorzystuje się parametr *BW1*, a w wypadku zmiennej określającej rodzaj komputera w węźle – parametr *BW2*.

Opracowano nowe zależności na wyznaczanie w każdej improwizacji wartości zmiennych decyzyjnych zgodnie z trójwartościową regułą ruletki [320, 327]. Sposób wyznaczania *v*-tej zmiennej decyzyjnej reprezentującej przydział modułu do węzła podczas procedur improwizacji typu *HMCR* i *PAR*, można zapisać za pomocą następującej zależności:

$$x_v^{new} := \begin{Bmatrix} \lfloor 1 + r_{2v}(I-1) + 0{,}5 \rfloor & \text{dla } r_{1v} > HMCR \\ adm1\lfloor x_v^k + r_{4v} BW1 \rfloor & \text{dla } r_{1v} \le HMCR, r_{3v} \le PAR \\ x_v^k & \text{dla } r_{1v} \le HMCR, r_{3v} > PAR \end{Bmatrix} \quad v = \overline{1,V}, \qquad (4.4.2)$$

gdzie:

r_{1v}, r_{2v}, r_{3v} $\in [0;1]$, $r_{4v} \in [-1;1]$;

$$adm1(x) = \begin{cases} 1 & \text{dla } x < 1; \\ x & \text{dla } 1 \le x \le I; \\ I & \text{dla } x > I. \end{cases}$$

Natomiast sposób wyznaczania *i*-tej zmiennej decyzyjnej reprezentującej przydział rodzaju komputera do węzła podczas procedury improwizacji typu *HMCR* i *PAR*, można zapisać za pomocą następującej zależności:

$$x_i^{new} := \begin{cases} \lfloor 1 + r_{2i}(J-1) + 0{,}5 \rfloor & \text{dla } r_{1i} > HMCR \\ adm2\lfloor x_i^k + r_{4i} BW2 \rfloor & \text{dla } r_{1i} \le HMCR, r_{3i} \le PAR \\ x_i^k & \text{dla } r_{1i} \le HMCR, r_{3i} > PAR \end{cases}, \; i = \overline{V+1, V+I}, \quad (4.4.3)$$

gdzie:

$$adm2(x) = \begin{cases} 1 & \text{dla } x < 1; \\ x & \text{dla } 1 \le x \le J; \\ J & \text{dla } x > J. \end{cases}$$

Jeśli podczas eksploracji przestrzeni konfiguracji w kolumnie macierzy *HM* wszystkie wartości zmiennych decyzyjnych są identyczne, to 20% współrzędnych jest modyfikowana losowo zgodnie z zależnością (4.4.1) w celu zwiększenia dywersyfikacji.

Aby wnikliwiej przedstawić działanie algorytmu harmonicznego HSΔmax, omówiony zostanie następujący eksperyment numeryczny. W instancji testowej zagadnienia optymalizacji (3.3.1) rozpatruje się minimalizację $\Delta_{\max}$ ważonego obciążeniu newralgicznego hosta dla $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$. W szczególności, jeśli $\rho_1 = 1, \rho_2 = 0$, to zachodzi minimalizacja $\hat{Z}_{max}$, a w wypadku $\rho_1 = 0$ $\rho_2 = 1$ - minimalizacja $\tilde{Z}_{max}$.

Niech *V*=12, *I*=6, *J*=3, a macierz czasów przetwarzania modułów ma postać (2.3.3). Natomiast macierz komunikacji między modułami ma postać (2.3.5). Ponadto rozpatruje się trzy ograniczenia: wydajnościowe, kosztowe i wymaganie dotyczące rozproszenia modułów. Niech $\vartheta = [1{,}84; 4{,}848; 6{,}771]$ [TFLOPS], $\vartheta_{\min} = 20$ [TFLOPS], $\xi = [400; 896; 1504]$ [USD] oraz $\xi_{\max} = 7000$ [USD]. Ponadto założono, że $v_{\min}$ wynosi 55. Specyfikacja wszystkich danych wejściowych dostępna jest w pliku *benchmarkNr1.txt* opisanym w dodatku. Natomiast selektory $\mu_1 = 1, \mu_2 = 1, \mu_{23} = 1$ ustawiane są interaktywnie w aplikacji ADAIORG'14 (rys. 32). Instancję problemu (3.3.1) można sformułować następująco.

Dla danych wejściowych: $\rho_1 = 0{,}5,\ \rho_2 = 0{,}5,\ \mu_1 = 1, \mu_2 = 1, \mu_3 = 1, \mu_4 = 0, \mu_5 = 0$, V=12, I=6, J=3, $T = [t_{vj}]_{V \times J}$ zgodne z (2.3.3), $\tau = [\tau_{vuik}]_{V \times V \times I \times I}$ zgodne z (2.3.5), $\vartheta = [1{,}84; 4{,}848; 6{,}771]$ [TFLOPS], $\vartheta_{\min}$ =20 [TFLOPS], $\xi = [400; 896; 1504]$ [USD], $\xi_{\max} = 7000$ [USD], $\upsilon_{\min}$=80

należy wyznaczyć x^*, takie że:

$$\Delta_{max}(x^*) = \min_{x \in \tilde{\boldsymbol{X}}} 0{,}5\hat{Z}_{max}(x) + 0{,}5\tilde{Z}_{max}(x), \tag{4.4.4}$$

gdzie $\tilde{\boldsymbol{X}} = \{x \in \boldsymbol{B}_{\boldsymbol{bin}}^{90} | \sum_{i=1}^{6} x_{vi}^{m} = 1 \text{ dla } v = \overline{1,12};\ \sum_{j=1}^{3} x_{ij}^{\pi} = 1 \text{ dla } i = \overline{1,6};\ \Theta(x) \geq 20 \text{; dla } \mu_1 \neq 0;$

$\Xi(x) \leq 7000 \text{ dla } \mu_2 \neq 0; \quad \upsilon(x) \geq 80 \text{ dla } \mu_3 \neq 0\}$

Instancja testowa (4.4.4) cechuje się 90 binarnymi zmiennymi decyzyjnymi, a zatem binarna przestrzeń przeszukiwań zawiera $1{,}24*10^{27}$ elementów. 18 ograniczeń równościowych zostało spełnionych w wyniku zastosowania kodowanych całkowitoliczbowego. Możliwe konfiguracje gridu, których jest $1{,}59*10^{12}$, powinny spełniać 3 ograniczenia nierównościowe.

Ewaluacja funkcji *fitness*, która obejmuje wyznaczenie wartości funkcji celu oraz funkcji kar, dla 10^6 konfiguracji zajmuje ok. 117 [s] na komputerze klasy *PC/Windows 7/Intel Core 2 Duo P7450 2,13GHz*. Ponieważ wszystkich konfiguracji jest $1{,}59*10^{12}$, to metoda pełnego przeglądu wymagałaby prawie 6 lat obliczeń. Natomiast algorytm harmoniczny *HSΔmax* w czasie poniżej minuty (59,11 [s]) wyznaczył rozwiązanie o najmniejszej wartości funkcji celu, którego nie udało się wcześniej wyznaczyć ani za pomocą algorytmu genetycznego ani za pomocą metody podziału i oszacowań w porównywalnym czasie obliczeń.

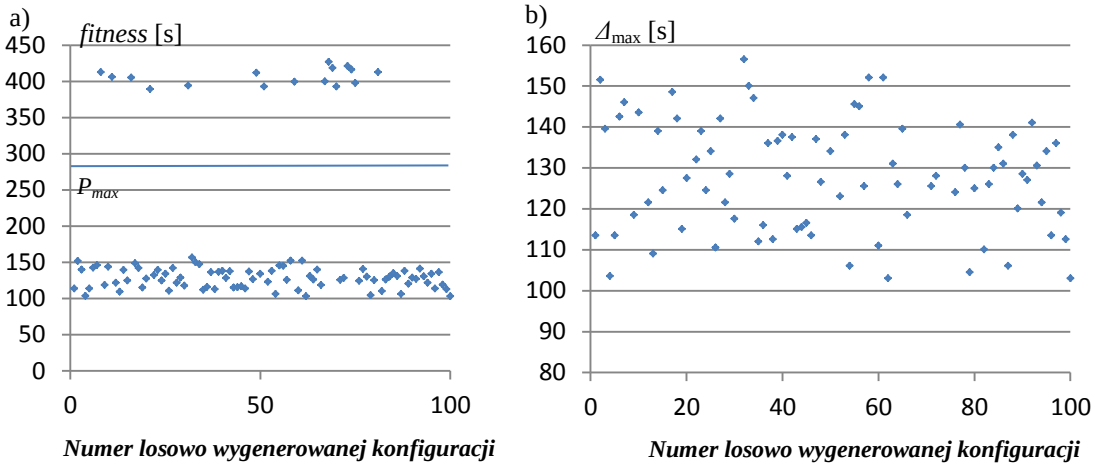
Metodę losowego generowania konfiguracji gridu zastosowano do wyznaczenia pamięci początkowej *HM*. Po wygenerowaniu 100 rozwiązań do *HM* kwalifikowane są konfiguracje o minimalnych wartościach $\Delta_{\max}$ oraz minimalnych obciążeniach $\hat{Z}_{max}$ i $\tilde{Z}_{max}$, a następnie tablica jest uzupełniana o kolejne konfiguracje najlepsze pod względem funkcji celu $\Delta_{\max}$.

Ze względu na ograniczenia maksymalizowana jest funkcja sprawności, jak niżej:

$$fitness(x) = \begin{cases} -\Delta_{\max}(x) + P_{\max}, \text{ dla } x \in \tilde{\boldsymbol{X}}, \\ -\Delta_{\max}(x) - \sum_{k=1}^{3} \mu_k P_k(x), \text{ dla } x \notin \tilde{\boldsymbol{X}}, \end{cases} \tag{4.4.5}$$

gdzie $P_{\max}$ - maksymalna wartość $\Delta_{\max}$, która dla instancji wynosi 278,5 [s].

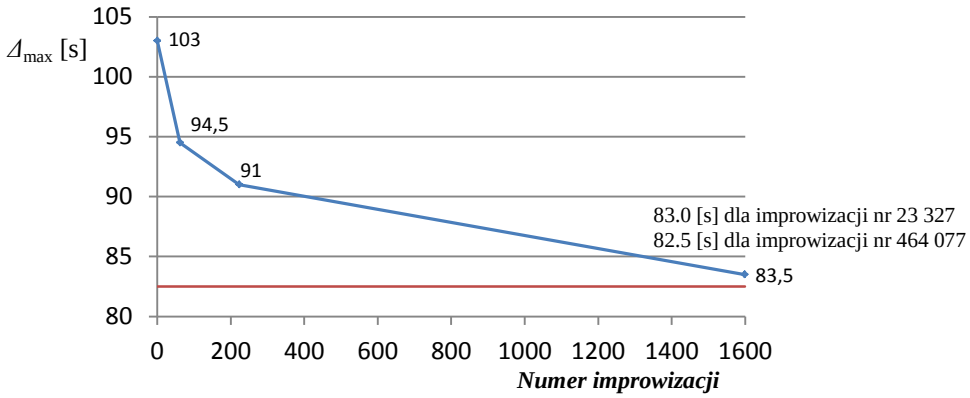
Na rysunku 33 a) zaprezentowano wartości *fitness* dla 100 losowo wygenerowanych konfiguracji. Powyżej linii oznaczającej wartość $P_{\max}$ zobrazowano oceny 17 konfiguracji niedopuszczalnych. Natomiast na rysunku 33 b) zaznaczono wartości ważonego obciążenia $\Delta_{\max}$ dla 83 rozwiązań dopuszczalnych. Najmniejszą wartość obciążenia 103 [s] wyznaczono w 61 iteracji.



Rys. 33. Wartości *fitness* i Δ_{max} dla stu losowo wygenerowanych konfiguracji: a) *fitness* b) ważone obciążenie Δ_{max} dla konfiguracji dopuszczalnych
Źródło: opracowanie własne.

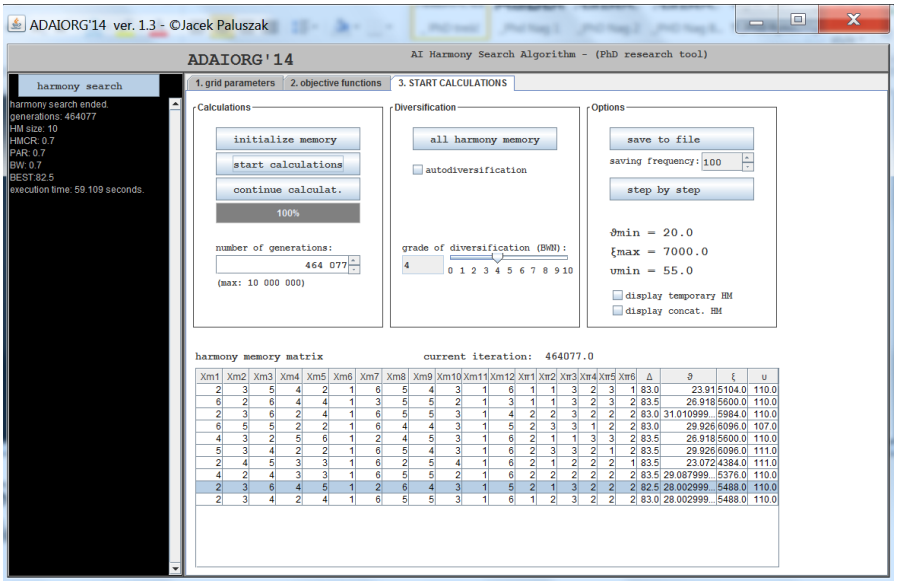
Wszystkie konfiguracje w wygenerowanej pamięci harmonicznej spełniają nałożone na nie ograniczenia. Przyjęto następujące wartości parametrów: *HMCR*=0,9, *PAR*=0,5, *BW1*=1, *BW2*=2 oraz *HMS*=10. Na podstawie początkowej pamięci *HM* algorytm harmoniczny wyznaczył konfigurację cechującą się obciążeniem 94,5 [s] po 62 improwizacjach (rys. 34).

Kolejna poprawa do wartości 91,0 [s] nastąpiła po 223 iteracjach. Natomiast konfigurację o obciążeniu ważonym 83,5 [s] wyznaczono po 1599 improwizacjach, co zajęło 0,28 [s]. Po 23 327 improwizacjach poprawiono ocenę na 83 [s]. Ostatnia poprawa ważonego obciążenia nastąpiła dopiero po 464 077 improwizacjach do wartości 82.5 [s], co zajęło 59,11 [s]. Wyniki zapisano do pliku wyjściowego (opcja *save to file*).



Rys. 34. Zbieżność wartości funkcji celu Δ_{max} do rozwiązania optymalnego dla wyników uzyskanych dla algorytmu harmonicznego *HSΔmax*
Źródło: opracowanie własne.

Na rysunku 35 przedstawiono interfejs aplikacji prezentujący stan pamięci *HM* po 464 077 improwizacjach, w którym znajduje się specyfikacja wyznaczonej suboptymalnej konfiguracji gridu.

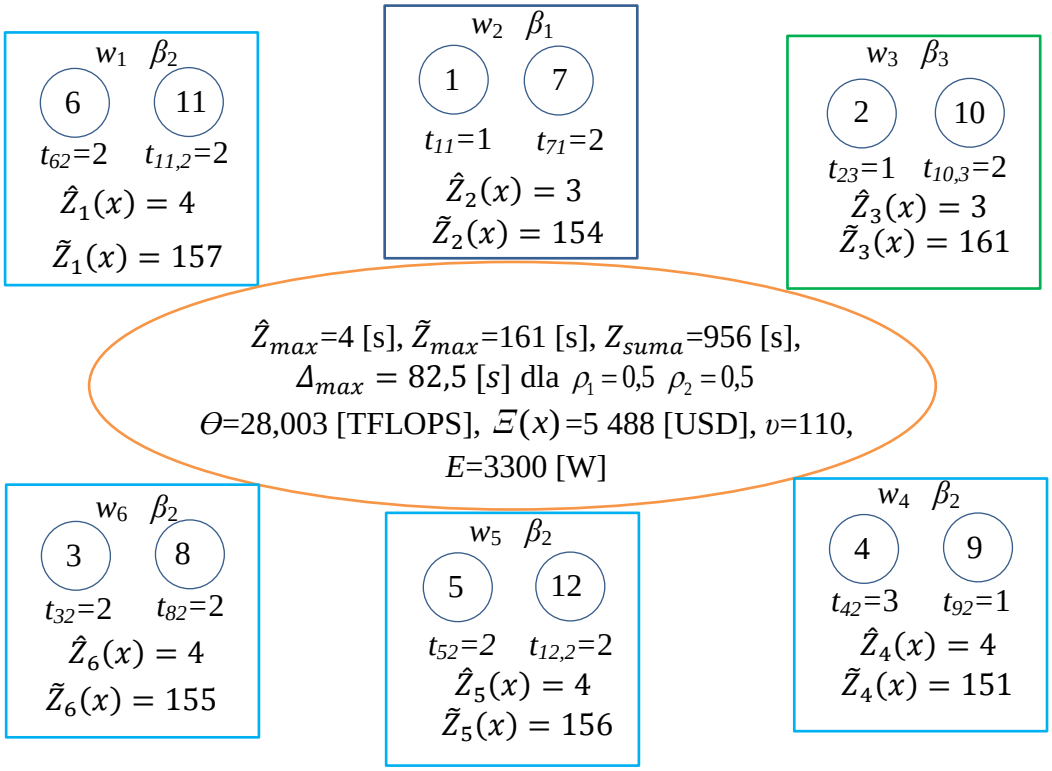


Rys. 35. Specyfikacja pamięci *HM* z rozwiązaniem optymalnym po 464 077 iteracjach
Źródło: opracowanie własne.

Na rysunku 36 zaprezentowano wyznaczoną konfigurację, którą można zapisać jako X^{α}=[2, 3, 6, 4, 5, 1, 2, 6, 4, 3, 1, 5]^T, X^{β}=[2, 1, 3, 2, 2, 2]^T. Wartość ważonego obciążenia $\Delta_{max} = 82{,}5\ [s]$ dla $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$ jest konsekwencją minimalnego obciążenia procesorów w newralgicznym hoście $\hat{Z}_{max}$=4 [s] oraz niewielkiego obciążenia komunikacyjnego w newralgicznym komputerze $\tilde{Z}_{max}$=161 [s]. Najbardziej obciążone procesory są w komputerach przydzielonych do węzłów nr 1, 4, 5 i 6. Natomiast najbardziej obciążony komunikacją jest komputer w węźle nr 3. Wartość

łączna obciążenia Z_{suma}=956 [s], a sumaryczna wydajność Θ=28,003 [TFLOPS]. Koszt komputerów $\Xi(x)$=5 488 [USD], stopień rozproszenia υ=110, a zużycie energii elektrycznej E=3300 [W].

W porównaniu do konfiguracji gridu z rysunku 23 konfiguracja suboptymalna cechuje się mniejszym obciążeniem CPU o 33,3%, mniejszym obciążeniem komunikacyjnym o 27,1%, a także mniejszym ważonym obciążeniem o 27,3%. Również łączne obciążenie jest mniejsze o 3,0%, a koszt zakupu komputerów - o 9,97%. Porównywalne są wartości stopnia rozproszenia oraz pobór mocy energii. Natomiast sumaryczna wydajność wszystkich komputerów jest niższa o 6,4%.



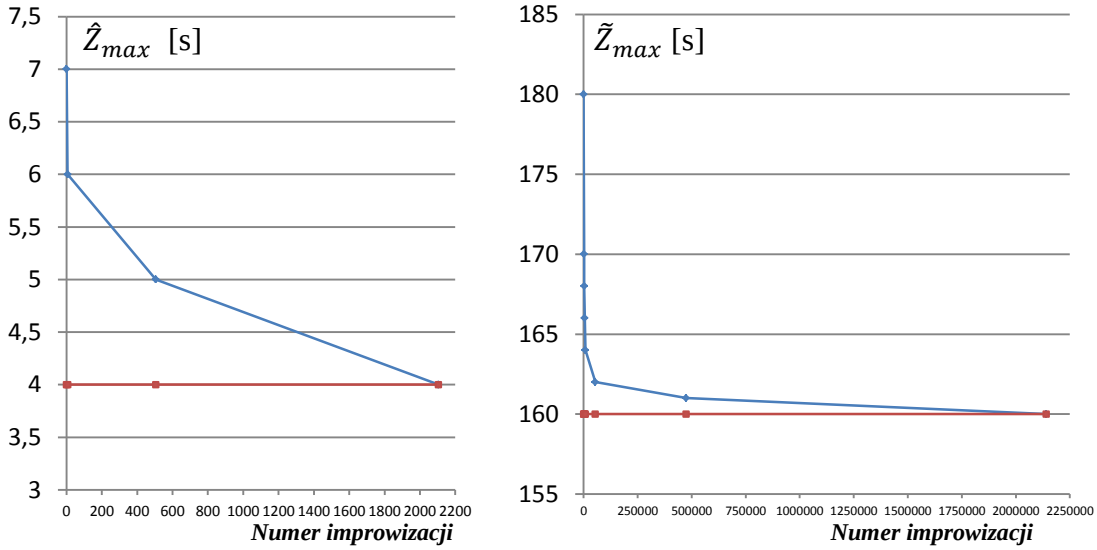
Rys. 36. Diagram suboptymalnej konfiguracji gridu pod względem ważonego obciążenia $\Delta_{max} = 82{,}5$ [s], przy czym V=12, I=6, J=3, $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$.
Źródło: opracowanie własne.

Za pomocą algorytmu HSΔmax wyznaczono także minimalną wartość obciążenia procesorów w newralgicznym komputerze oraz minimalną wartość obciążenia komunikacyjnego w newralgicznym hoście przy nałożonych ograniczeniach: wydajnościowych, finansowych i wymaganiu odpowiedniego stopnia rozproszenia. Jednym z rozwiązań optymalnych pod względem $\hat{Z}_{max}$ jest konfiguracja z rysunku 36.

Zbieżność wyników uzyskanych za pomocą algorytmu przedstawiono na rysunku 37a), na którym przedstawiono osiągnięcie optimum o wartości 4 [s] w 2 105 iteracji

(9,33 [s]). Alternatywna konfiguracja minimalizująca obciążenie $\hat{Z}_{max}$=4 [s] jest parą wektorów X^{α}=[6, 4, 1, 2, 3, 4, 5, 4, 2, 6, 3, 5]^T, X^{β}=[3, 2, 2, 3, 3, 1]^T. Ponadto $\tilde{Z}_{max}$=197 [s], Z_{suma}=927 [s], $\Delta_{max} = 100{,}5\ [s]$ dla $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$, Θ=31,849 [TFLOPS], $\Xi(x)$ =6 704 [USD], υ=107 oraz E=3400 [W].

O ile wyznaczanie minimalnego obciążenia procesorów newralgicznego komputera za pomocą algorytmu harmonicznego jest stosunkowo szybkie dla rozpatrywanej instancji testowej, o tyle wyznaczanie obciążenia komunikacyjnego newralgicznego węzła jest już znacznie dłuższe, podobnie jak poszukiwania minimum ważonego obciążenia Δ_{max}. Na rysunku 37b) zobrazowano zbieżność wyznaczonych ocen $\tilde{Z}_{max}$ za pomocą algorytmu HSΔmax.



Rys. 37. Zbieżność wartości funkcji celu do wartości optymalnej dla wyników uzyskanych za pomocą algorytmu harmonicznego *HSΔmax*
a) minimalizacja $\hat{Z}_{max}$ b) minimalizacja $\tilde{Z}_{max}$
Źródło: opracowanie własne.

Konfiguracja suboptymalna cechująca się $\tilde{Z}_{max}$= 160 [s] została wyznaczona po 2 140 084 improwizacjach, co zajęło 245 [s]. Wyznaczona konfiguracja suboptymalna X^{α}=[5, 6, 6, 2, 2, 1, 5, 3, 3, 4, 1, 6]^T, X^{β}=[2, 2, 3, 3, 1, 3]^T cechuje się następującymi parametrami: $\hat{Z}_{max}$=5 [s], Z_{suma}=902 [s], $\Delta_{max} = 82{,}5\ [s]$ dla $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$, Θ=31,849 [TFLOPS], $\Xi(x)$=6 704 [USD], υ=107 oraz E=3400 [W].

Za pomocą algorytmu HS można wyznaczyć *HMS* konfiguracji suboptymalnych. W tym wypadku wyznaczono jeszcze jedną konfigurację o $\tilde{Z}_{max}$= 160 [s]. Cechą charakterystyczną algorytmu harmonicznego jest fakt, że prowadzi równolegle *HMS* poszukiwań rozwiązania suboptymalnego [77, 214]. Każdy wiersz w pamięci *HM*

odpowiada stanowi jednej trajektorii przeszukiwań, która zmienia się wraz z kolejnymi iteracjami [93, 217].

Jeśli najlepsze rozwiązanie utknie w minimum lokalnym, to pozostałe rozwiązania są wykorzystywane do dalszych poszukiwań i ich sprawność jest systematycznie poprawiana [106, 218]. Zazwyczaj po pewnym czasie rozwiązanie najlepsze w sensie *fitness*, które tkwi w maksimum lokalnym tej funkcji, staje się najgorsze i jest zastępowane przez nową improwizację [154]. W ten sposób algorytm omija lokalne optima.

Algorytm harmoniczny może „przejść" przez obszar niedopuszczalny, będąc w zbiorze rozwiązań dopuszczalnych [160, 219]. Jeśli lokalna poprawa rozwiązań w pamięci harmonicznej doprowadzi do utknięcia w pewnym obszarze wszystkich trajektorii przeszukiwań, to wówczas improwizacje intensywniej wykorzystują sposób losowego wyznaczania wartości zmiennych decyzyjnych [178].

Jednym ze sposobów na powyższe jest zmniejszanie wartości parametru *HMCR* wraz z numerem improwizacji *NG* po to, aby coraz więcej wartości było wyznaczanych losowo z ewentualnym uwzględnieniem przedziałów *BW1 i BW2* [206, 236]. Podobnie, zwiększanie parametru *PAR* powoduje, że coraz mniej improwizacji wyznaczanych jest z uwzględnieniem przedziałów *BW1 i BW2*, a coraz więcej z całkowicie losowymi wartościami [238, 277]. Ponadto zwiększanie *BW1* lub *BW2* wraz z numerem improwizacji ułatwia wyjście z obszarów, które są trudne do opuszczenia przez trajektorie przeszukiwań [306].

Warto jednak podkreślić, że powyższą strategię stosuje się w końcowej fazie przeszukiwań algorytmu harmonicznego, gdyż zbyt wczesne jej rozpoczęcie prowadzi do wolnego procesu poprawy funkcji celu. Kryterium rozpoczęcia automatycznej modyfikacji parametrów *HMCR, PAR*, *BW1 i BW2* jest przekroczenie NG_{lok} - założonej liczby iteracji bez poprawy jakiegokolwiek rozwiązania w pamięci *HM*. Zazwyczaj $NG_{lok}=0{,}2NG_{max}$, gdzie wartość NG_{max} zależy od założonego czasu obliczeń na określonym komputerze. Rozpoczęcie zmniejszania *HMCR* zachodzi dla $NG>0{,}4NG_{max}$.

Warto także dodać, że algorytm HSΔmax może być przystosowany do optymalizacji dowolnej funkcji celu w zagadnieniach (3.3.1), (3.3.2) oraz (3.3.1). Dlatego podczas wyznaczania wartości funkcji *fitness* w aplikacji ADAIORG'14 stosuje się zależność (3.3.10).

Złożoność obliczeniowa algorytmu HSΔmax zależy od nakładów na wyznaczenie wartości funkcji skalarnych oraz od złożoności procedury weryfikacji dopuszczalności

konfiguracji. Dla zagadnień (3.1.1), (3.2.1) i (3.3.1) cechuje się złożonością obliczeniową $O(n^6)$.

Wyznaczenie wartości $\hat{Z}(x)$ posiada złożoność $O(n^3)$, a $\tilde{Z}(x)$ - $O(n^5)$, gdzie n=max{*I, V, J*}. Zatem nakłady obliczeniowe na wyznaczenie ważonego obciążenia są proporcjonalne do n^5. Złożoność obliczeniowa procedury weryfikacji spełnienia ograniczeń zależy od złożoności procedur wyznaczania: łącznej wydajności, kosztów zakupu komputerów, stopnia rozproszenia, zajętości pamięci *RAM* i *HD*, a także poboru mocy. Można pokazać, że złożoność obliczeniowa procedury weryfikacji jest $O(n^2)$.

Inicjalizacja pamięci *HM*, w której *KL* razy losowo generuje się konfiguracje gridu cechuje się złożonością obliczeniową $O(n^5)$, gdzie n=max{*I, V, J, KL*}. Natomiast procedura improwizacji i aktualizacji pamięci *HM*, cechuje się złożonością obliczeniową $O(n^6)$, przy czym n=max{*I, V, J,* $NG_{\max}$}. Zazwyczaj decydujący wpływ na czas obliczeń ma maksymalna liczba improwizacji $NG_{\max}$ [248, 307].

4.5. Wybrane metody optymalizacji wielokryterialnej

Nieliczne próby zastosowań algorytmów harmonicznych do wyznaczania rozwiązań optymalnych w sensie *Pareto* odnotowuje się od 2011 roku [241]. Nieznane są jednak zastosowania algorytmów harmonicznych do rozwiązywania zagadnień polioptymalizacji konfiguracji gridu. Warto zatem przeanalizować metody ogólne optymalizacji wektorowej, w których ewentualnie można zastosować algorytmy harmoniczne do rozważanych problemów polioptymalizacji wykorzystania zasobów.

Metody optymalizacji wielokryterialnej rozwinęły się bardzo intensywnie w zagadnieniach optymalizacji ciągłej, w których minimalizuje się *N* kryteriów skalarnych $\varphi_1,\ldots,\varphi_n,\ldots,\varphi_N$, przy ograniczeniach nierównościowych i równościowych, jak niżej [92]:

$$g_m(x) \geq 0,\ m = \overline{1,M}; \tag{4.5.1}$$

$$g_l(x) = 0,\ l = \overline{1,L}. \tag{4.5.2}$$

W zagadnieniu liniowym optymalizacji wielokryterialnej skalarne funkcje celu oraz ograniczenia (4.5.1) i (4.5.2) są liniowe. Ponadto zmienne decyzyjne są ciągłe, tzn. przyjmują wartości ze zbioru liczb rzeczywistych [92].

Za pomocą *metody Bensona* można wyznaczyć wszystkie rozwiązania *Pareto*-optymalne dla liniowych problemów optymalizacji wektorowej [39]. Ponadto

w literaturze przedmiotu rozpatrywanych jest szereg wielokryterialnych metod programowania liniowego opartych na algorytmie simpleks [23, 195]. Warto odnotować także, że *Ida* zaproponował metodę promieniową dla tej klasy zagadnień [161]. Szczególny przypadek zagadnienia polioptymalizacji liniowej, a mianowicie problem transportowy dla dwóch kryteriów optymalizacji rozważali *Aneja* i *Nair* [21].

Mavrotas i *Diakoulaki* zaproponowali *metodę podziału i ograniczeń* do liniowej polioptymalizacji zagadnienia, w którym zmienne decyzyjne mogą przyjmować wartość 0 lub 1, podobnie jak w zagadnieniach optymalizacji konfiguracji gridu [245]. Jednakże kryteria optymalizacji oraz ograniczenia w zagadnieniach rozpatrywanych w rozprawie są na ogół nieliniowe. Z tego powodu metody *Mavrotasa-Diakoulakiego* nie można zastosować do wyznaczania gridowych konfiguracji efektywnych.

Klein i *Hannan* rozważali algorytm do wyznaczania rozwiązań *Pareto*-optymalnych w wektorowych zagadnieniach liniowego programowania, w których zmienne decyzyjne przyjmują wartości całkowitoliczbowe [189, 328]. Niestety, w zagadnieniach optymalizacji wykorzystania zasobów kryteria są nieliniowe.

Z kolei *Melih* i *Azizoğlu* zaproponowali metodę wyznaczania zbioru rozwiązań niezdominowanych dla nieliniowego zagadnienia programowania całkowitoliczbowego [249]. Ponadto *Przybylski, Gandibleux* i *Ehrgott* opracowali adekwatny algorytm rekurencyjny [286, 287], a *Lokman* i *Köksalan* - alternatywną metodę dla tej klasy zagadnień [229]. Metody te cechują się jednak długim czasem obliczeń dla nawet niewielkiej liczby zmiennych decyzyjnych [229, 249, 287].

Zagadnienie programowania zero-jedynkowego oraz zagadnienie programowania całkowitoliczbowego należą do klasy problemów optymalizacji kombinatorycznej (dyskretnej) [43], do rozwiązania których zazwyczaj stosuje się metodę podziału i ograniczeń [84]. Wspomnianą metodę zastosowano do wyznaczania trasy w wielokryterialnym problemie komiwojażera [170]. Natomiast *Kirlik* i *Sayın* skonstruowali alternatywną metodę do wyznaczania zbioru rozwiązań niezdominowanych [188]. Niestety powyższe metody wymagają także długiego czasu obliczeń dla większych instancji.

Bez względu na powyższe klasy zagadnień polioptymalizacji powszechnie stosowane są metody uniwersalne, takie jak: metoda sumy ważonej, metoda ε-ograniczeń, czy metody punktów referencyjnych [70].

Metoda sumy ważonej opiera się na liniowej kombinacji kryteriów skalarnych i polega na transformacji zagadnienia optymalizacji wielokryterialnej ($\tilde{\boldsymbol{X}}$, φ, $\chi^{\leq}$) w zadanie optymalizacji jednokryterialnej, jak niżej [225]:

$$\min_{x \in \tilde{\boldsymbol{X}}} \Delta(x), \tag{4.5.3}$$

gdzie:

$$\Delta(x) = \sum_{n=1}^{N} \rho_n \varphi_n(x),$$

$$\sum_{n=1}^{N} \rho_n = 1; \ \rho_n \geq 0, \rho_n \leq 0, n = \overline{1, N}.$$

Ponieważ rozważany w rozprawie zbiór ocen $\boldsymbol{\Omega}$ nie jest ciągły ani tym bardziej wypukły, to metoda sumy ważonej nie gwarantuje wyznaczenia konfiguracji sprawnej [36]. Zasadniczą modyfikacją powyższego podejścia jest metoda promieniowa polegająca na systematycznym dobieraniu wag i wielokrotnym rozwiązaniu zagadnienia (4.5.3). Metodę sumy ważonej zastosowano w rozprawie do minimalizacji dwóch kryteriów: $\hat{Z}_{max}$ i $\tilde{Z}_{max}$, agregując je za pomocą kryterium Δ_{max} w zagadnieniach (3.1.1), (3.2.1) i (3.3.1). Uzyskane konfiguracje dla zadanego wektora wag nie muszą być Pareto-optymalne.

Nieco szerszy zakres zastosowań posiada *metoda ε-ograniczeń*, w której minimalizuje się wybrane kryterium cząstkowe $\varphi_k(x)$, a na pozostałe kryteria skalarne nałożone są ograniczenia [17, 246], jak niżej:

$$\min_{x \in \tilde{\boldsymbol{X}}} \varphi_k(x), \tag{4.5.4}$$

przy ogr.:

$$\varphi_n(x) \leq \varepsilon_n, \ n = \overline{1, N}, \ k \neq n.$$

Ponieważ $\boldsymbol{\Omega}$ nie jest zbiorem ciągłym i wypukłym ani zbiorem ciągłym i wklęsłym, to metoda ε-ograniczeń również nie gwarantuje wyznaczenia konfiguracji efektywnej w rozpatrywanym zagadnieniu polioptymalizacji konfiguracji gridu [247].

Warto wspomnieć o zaawansowanej implementacji *metody ε-ograniczeń* [246], a także o zastosowaniu jej w wielokryterialnym programowaniu całkowitoliczbowym [247]. Usprawnienie tej metody zaproponowali *Zhang* i *Reimann* [369]. Natomiast *Laumanns*, *Thiele* i *Zitzler* omówili wykorzystanie metody *ε-ograniczeń* do adaptacyjnego doboru parametrów dla metaheurystyk [209].

Metoda celów referencyjnych lub *programowanie celowe* polega na transformacji zagadnienia polioptymalizacji w zagadnienie optymalizacji jednokryterialnej z wykorzystaniem celu referencyjnego ω^{ref} [354]:

$$\min_{x \in \tilde{X}} \delta(x), \tag{4.5.5}$$

gdzie $\delta(x) = \left[\sum_{n=1}^{N} \left| \omega_n^{ref} - \varphi_n(x) \right|^p \right]^{\frac{1}{p}}$.

Wartość parametru *p* spełnia ograniczenie $1 \le p < \infty$. Wadą rozważanego podejścia jest fakt, że niewłaściwy cel referencyjny na ogół uniemożliwia wyznaczenie rozwiązania *Pareto*-optymalnego. Kolejną wadą jest pominięcie normalizacji przestrzeni wielokryterialnej.

W metodzie *ważonych metryk* minimalizuje się kombinację liniową metryki dla zadanego parametru *p*, $1 \le p < \infty$, jak niżej [18]:

$$\min_{x \in \tilde{X}} \tilde{\delta}(x), \ldots\ldots \qquad \ldots\ldots\ldots\ldots (4.5.6)$$

gdzie:

$$\tilde{\delta}(x) = \left[\sum_{n=1}^{N} \rho_n \left| \omega_n^{ref} - \varphi_n(x) \right|^p \right]^{\frac{1}{p}},$$

$$\sum_{n=1}^{N} \rho_n = 1; \ \rho_n \ge 0, \ \rho_n \le 0, \ n = \overline{1, N}.$$

Wadą metody ważonych metryk jest dylemat związany z wyborem celu referencyjnego. Ponadto nie zachodzi normalizacja przestrzeni ocen.

W innym wariancie metody celów referencyjnych wymaga się, aby punkt referencyjny należał do zbioru ocen. Wówczas minimalizacji podlega maksymalna *n*-ta różnica między współrzędną oceny a współrzędną zdominowanego punktu odniesienia [18]. Punktem odniesienia może być ocena antyidealna lub wybrana arbitralnie ocena dopuszczalna, np. ocena najlepszego rozwiązania podczas dotychczasowych przeszukiwań [17]. Metoda polega na transformacji problemu polioptymalizacji do następującego zagadnienia optymalizacji jednokryterialnej [18]:

$$\min_{x \in \tilde{X}} H(x), \tag{4.5.7}$$

$$\text{przy ogr.: } \varphi_n(x) \le \omega_n^{ref}, \ n = \overline{1, N},$$

gdzie $H(x) = \sum_{n=1}^{N} \max\left\{0, \omega_n^{ref} - \omega_n(x)\right\}$.

W niektórych metodach wykorzystuje się punkt *nadir*. Przy minimalizacji kryteriów cząstkowych punkt *nadir* jest kresem górnym zbioru ocen efektywnych [17], co można zdefiniować, jak niżej:

$$N_n^* = \sup_{x \in \tilde{X}_N^{\tilde{\varepsilon}}} \varphi_n(x), n = \overline{1, N}. \tag{4.5.8}$$

Metoda Wierzbickiego polega na sprowadzeniu zagadnienia polioptymalizacji do następującego problemu optymalizacji jednokryterialnej przy wykorzystaniu aż trzech punktów charakterystycznych przestrzeni kryterialnej: punktu *nadir*, punktu idealnego i punktu referencyjnego preferowanego przez decydenta. Pomocnicze zagadnienie minimalizacji można sformułować następująco [354]:

$$\min_{x \in \tilde{X}} \max_{n=\overline{1,N}} \breve{W}_n(x), \tag{4.5.9}$$

gdzie:

$$\breve{W}_n(x) = \left[\frac{\varphi_n - z_n}{N_n^* - \omega_n^o} \right] + \lambda \sum_{n=1}^{N} \frac{\varphi_n}{N_n^* - \omega_n^o}, n = \overline{1, N};$$

z_n – n-ta współrzędna punktu referencyjnego, który jest preferowany przez decydenta,

λ – współczynnik regularyzacji, λ>0, λ to zazwyczaj niewielka liczba rzeczywista [354].

Oprócz poszukiwania rozwiązań *Pareto*-optymalnych, wyznaczane mogą być alternatywy leksykograficzne [17]. *Metoda leksykograficzna* polega na rozwiązaniu zagadnienia polioptymalizacji w N etapach. Początkowo minimalizuje się najbardziej preferowane przez decydenta kryterium cząstkowe, a na wartości pozostałych kryteriów nakłada się ograniczenia tak jak w metodzie *ε-ograniczeń* [17]. Wyznaczony zbiór rozwiązań podlega zawężeniu podczas minimalizacji kolejnego kryterium przy uwzględnieniu ograniczeń na pozostałe kryteria. Za konfigurację leksykograficzną (hierarchiczną) przyjmuje się konfiguracje wyznaczoną podczas rozwiązywania N-tego zagadnienia minimalizacji. Zmiana przez decydenta priorytetów prowadzi zazwyczaj do uzyskania innych konfiguracji.

Warto podkreślić, że metody optymalizacji wielokryterialnej są implementowane na komputerach równoległych, w środowisku gridowym oraz w chmurach obliczeniowych. W szczególności *Lemesre*, *Dhaenens* i *Talbi* opracowali równoległą wersję algorytmu wyznaczania rozwiązań *Pareto*-optymalnych w odniesieniu do dwóch kryteriów [216].

Steuer zaproponował podział metod polioptymalizacji na metody: *a priori*, *interaktywne* i *a posteriori* ze względu na etap procesu decyzyjnego, w którym decydent ujawnia swoje preferencje [325].

Jeśli decydent określa swoje preferencje odnośnie kryteriów optymalizacji jeszcze przed rozpoczęciem procesu decyzyjnego, to taka sytuacja cechuje metody *a priori*. Zazwyczaj decydent podaje dodatkowe parametry metod skalaryzacji, w tym wektor wag, wektor parametrów ε lub punkt referencyjny. Istotną determinantą jest rodzaj poszukiwanych rozwiązań: *Pareto*-optymalnych, leksykograficznych lub kompromisowych. W wypadku rozwiązań *Pareto*-optymalnych, decydent może zażądać pojedynczego elementów, reprezentacji zbioru *Pareto* lub całego zbioru rozwiązań niezdominowanych. Metody *a priori* stosuje się w ściśle sformułowanych sytuacjach decyzyjnych.

Metody interaktywne rozwinęły się w latach osiemdziesiątych wraz z rozwojem interfejsów graficznych, które umożliwiły ocenę *online* przez decydenta wyznaczanych rozwiązań. Decydent po fazie obliczeń przez komputer, wyraża swoje preferencje, wskazując najbardziej wartościowe rozwiązania, stopy substytucji między przedziałami wartości kryteriów lub zmieniając parametry stosowanej metody skalaryzacji (punkt referencyjny, wektor wag, czy wektor ε). Po interakcji z decydentem wykonywana jest faza obliczeniowa, w której uwzględnia się informacje podane przez decydenta. Warto podkreślić, że decydent może zmieniać swoje preferencje w wyniku zrozumienia istotnych relacji między kryteriami. W efekcie decydent steruje procesem optymalizacji. Dialog decydenta z komputerem kończy się w wypadku akceptacji rozwiązania przez decydenta lub też po upływie zadanego czasu na podjęcie decyzji.

Warto podkreślić, że algorytm harmoniczny jest predystynowany do wykorzystania w metodach interaktywnych ze względu na pamięć harmoniczną *HM*. W aplikacji ADAIORG'14 decydent może wielokrotnie zmieniać wagi funkcji oceniających obciążenia gridu w fazie dialogu, a następnie kontynuować obliczenia. Decyzja o wartościach wag podejmowana jest po analizie konfiguracji zapisanych w pamięci harmonicznej.

W metodach *a posteriori* wyznacza się reprezentację zbioru rozwiązań niezdominowanych lub cały zbiór, po czym decydent w celu wyboru jednej alternatywy proponuje dodatkowe kryterium, podaje parametr rozwiązań kompromisowych lub wprost wskazuje preferowane rozwiązanie.

Analogicznie do taksonomii metod optymalizacji jednokryterialnej można wyróżnić wielokryterialne metaheurystyki w klasie metod przybliżonych i ogólnych. *Ruzika* i *Wiecek* omówili metaheurystyki polioptymalizacji powstałe na początku tego stulecia [302]. Obecnie najbardziej intensywnie rozwijają się badania nad wielokryterialnym algorytmami ewolucyjnymi [228, 309, 323]. Nieco wolniej postępują prace nad polioptymalizacją za pomocą *tabu search* [33]. Na tym tle do „najmłodszych" metaheurystyk polioptymalizacji należy algorytm harmoniczny [87].

Pierwszym wielokryterialnym algorytmem ewolucyjnym był algorytm VEGA (ang. *Vector Evaluated Genetic Algorithm*), w którym populacja konfiguracji dzielona jest na *N* podzbiorów ze względu na kryteria cząstkowe oraz mechanizm selekcji. Natomiast mutacja i krzyżowanie zachodzą w całej populacji [68].

Goldberg wprowadził *warstwową procedurę nadawania rang* w celu posortowania poziomów dominacji w populacji [146]. Niezależnie *Fonseca* i *Fleming* w algorytmie MOGA (ang. *Multi-Objective Genetic Algorithm*) zaproponowali alternatywną procedurę nadawaniu rang w oparciu o *liczbę dominujących osobników* [104].

W wielokryterialnym algorytmie niszowania NPGA (ang. *Niched Pareto Genetic Algorithm*) wykorzystano procedurę niszowania w odniesieniu do rozstrzygania, które osobniki o jednakowej randze należy zakwalifikować do populacji potomnej [155]. W 2000 roku *Deb* połączył procedurę nadawania rang i mechanizm niszowania w algorytmie NSGA-II [30, 79, 80]. Natomiast jeden z pierwszych wielokryterialnych algorytmów ewolucyjno-neuronowych opracowano w 1995 roku w odniesieniu do polioptymalizacji przydziału zadań [33, 109].

Z kolei w algorytmie SPEA (ang. *Strength Pareto Evolutionary Algorithm*) wprowadzono *archiwum*, aby nie „gubić" rozwiązań efektywnych podczas optymalizacji [371]. Ranga alternatywy zależy od liczby dominujących je rozwiązań pochodzących z archiwum zgodnie z procedurą *Fonseci-Fleminga* [165]. Ponadto rozwiązania z archiwum oraz z bieżącej populacji podlegają *selekcji elitarystycznej* [71, 371]. Kolejne wersje tej klasy algorytmu to: SPEA 2 z 2001 oraz SPEA 2+ z 2004 roku [204, 290].

Od algorytmów ewolucyjnych, które są modyfikacjami algorytmów genetycznych, nieco odbiegają wektorowe strategie ewolucyjne: algorytm PAES (ang. *Pareto Archived Evolution Strategy*) oraz algorytm PESA (ang. *Pareto Envelope-based Selection Algorithm*) [72]. W algorytmie PAES zastosowano miary zagęszczenia

wyznaczającą liczebności ocen w hiperześcianach tworzących przestrzeń wielokryterialną. Preferowane są oceny z mniej zagęszczonych hiperześcianów. Usunięcie rozwiązania z archiwum oparte jest na mierze zagęszczenia [72].

Przegląd wielokryterialnych algorytmów przeszukiwania tabu zamieszczono w [244], a innych metaheurystyk, w tym algorytmów symulowanego wyżarzania w [41, 87]. *Farina* i *Amato* omówili zasady wykorzystania logiki rozmytej w polioptymalizacji [95].

4.6. Wielokryterialne algorytmy harmoniczne do wyznaczania konfiguracji *Pareto*-optymalnych

Ricart, *Hüttemann*, *Lima* i *Barán* zaproponowali w 2011 roku, prawdopodobnie jako pierwsi, dwa zaawansowane warianty algorytmu harmonicznego do wyznaczania rozwiązań optymalnych w sensie *Pareto* [295]. Algorytmy zastosowano do minimalizacji sześciu problemów testowych z dwoma kryteriami [371].

W pierwszym wariancie harmonicznego algorytmu polioptymalizacji w każdej iteracji konstruuje się jedną improwizację, która jest wprowadzana do *HM*, jeśli cechuje się niższym rankingiem w sensie *Fonseci-Fleminga* niż rozwiązanie o najwyższej randze w repozytorium. W drugim wariancie konstruuje się tymczasową pamięć harmoniczną, która zawiera *HMS* improwizacji. Ponadto wykorzystuje się procedurę z algorytmu SPEA2 do redukcji liczby rozwiązań z 2*HMS* do wielkości *HM* [295].

Eksperymenty numeryczne przeprowadzono dla następujących parametrów: $HMS=100$, $HMCR=0{,}95$, $PAR=0{,}1$ oraz $BW=0{,}01\Delta x$, gdzie Δx jest szerokością przedziału wartości zmiennych decyzyjnych [295]. Natomiast w jednym zagadnieniu najlepsze wyniki uzyskano dla $BW=0{,}4\Delta x$. Ponadto podano rekomendowane przedziały wartości dla następujących parametrów: $HMCR \in [0{,}70;\ 0{,}95]$, $PAR \in [0{,}1;\ 0{,}5]$ oraz $BW \in [0{,}01\Delta x;\ 0{,}10\Delta x]$. Stwierdzono, że uzyskane wyniki są porównywalne z wynikami wyznaczonymi za pomocą algorytmu ewolucyjnego NSGA-II dla 6 instancji [295].

Warto także wspomnieć o dwóch wcześniejszych pracach. *Geem* zaproponował w 2010 roku wykorzystanie algorytmu harmonicznego do planowania harmonogramu projektu o minimalnym czasie wykonania oraz minimalnych kosztach [123]. Czas zakończenia projektu skorelowano z kosztem jego wykonania. Problem wielokryterialny sprowadzono do zagadnienia optymalizacji jednokryterialnej, do którego rozwiązania wykorzystano klasyczny algorytm harmoniczny. Algorytm

wyznaczył rozwiązanie o nieco wyższej jakości niż algorytm genetyczny czy algorytm kolonii mrówek [123].

Wcześniej o możliwości zastosowania algorytmu harmonicznego do polioptymalizacji wspomnieli *Kim, Baek, Jo* i *Park* w 2004 roku do planowania eksploatacji sieci wodociągowych przez okres dwudziestu lat [186]. Analizowano pięć kryteriów: koszty inwestycyjne (wymiany, rewitalizacji i naprawy), koszty pompowania wody i łączną ilość wycieków z nieszczelności w ciągu roku (koszt ok. 300 mln USD w Korei Płd.). W algorytmie harmonicznym zasugerowano wprowadzenie procedury sortowania ocen. Niestety, w rozważanej pracy brakuje opisu algorytmu harmonicznego w wersji wielokryterialnej. Nie podano także wyników eksperymentów numerycznych z tym algorytmem i interpretacji uzyskanych wyników *Pareto*-optymalnych.

Sivasubramani i *Swarup* zaproponowali w 2011 roku, a więc równocześnie do zespołu *Ricarta*, wielokryterialny algorytm harmoniczny opierający się na procedurze nadawania rang *Goldberga*, a nie procedurze *Fonseci-Fleminga* [322]. Zaproponowali *względną funkcję zagęszczenia* do selekcji rozwiązań cechujących się rangą graniczną. Algorytm zastosowano do minimalizacji kosztów wytwarzania ogrzewania, a także do minimalizacji emisji zanieczyszczeń. Stwierdzono, że uzyskane wyniki przewyższają rezultaty wyznaczone za pomocą algorytmu NSGA II pod względem równomierności rozproszenia ocen niezdominowanych. Przyjęto parametry: HMS = 20, $HMCR$ = 0,85 oraz $NG_{\max}$ = 1000. Natomiast parametr PAR zwiększano liniowo od 0,2 do 1 wraz z liczbą improwizacji NG, jak niżej [322]:

$$PAR(NG) = PAR_{\min} + \frac{PAR_{\max} - PAR_{\min}}{NG_{\max}} NG, \tag{4.6.1}$$

gdzie $PAR_{\min}$=0,2; $PAR_{\max}$=1.

Wzrost parametru PAR odpowiada zwiększeniu intensywności modyfikacji korzystających z zawartości pamięci harmonicznych w stosunku do improwizacji losowych.

Aby zwiększyć intensywność przeszukiwań, szerokość przedziału modyfikacji zmiennej BW maleje wykładniczo od 0,9 do 0,45 wraz ze wzrostem NG, jak niżej:

$$BW(NG) = BW_{\max} \exp\left(\frac{\ln \frac{BW_{\min}}{BW_{\max}}}{NG_{\max}} NG \right), \tag{4.6.2}$$

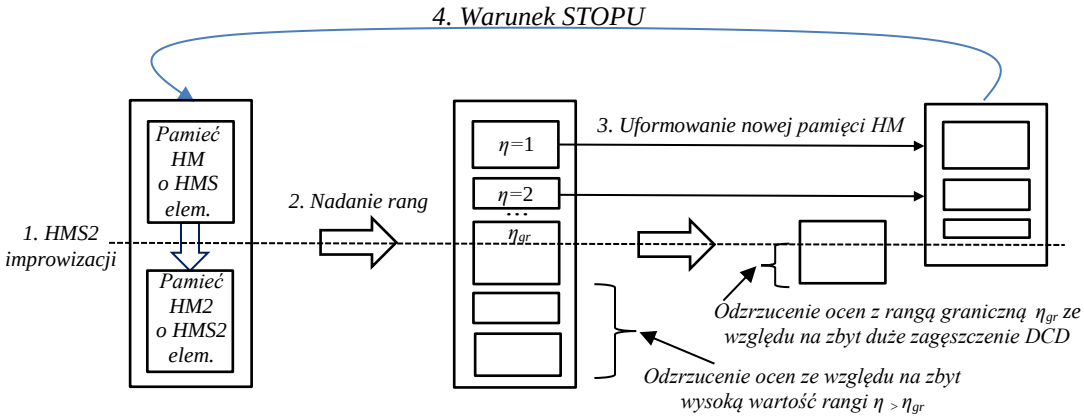
gdzie $BW_{\min}$=0,4; $BW_{\max}$=0,9.

Fesanghary et al. zaproponowali wielokryterialny algorytm harmoniczny do minimalizacji kosztu konstrukcji i eksploatacji budynków mieszkalnych, a także do minimalizacji emisji CO_2 [98]. Wartość pierwszego kryterium - koszt utrzymania budynku można zredukować, zmniejszając wydatki odnośnie: inwestycji, zużycia energii, eksploatacji, zarządzania oraz napraw. Natomiast ograniczenie emisji CO_2 (drugie kryterium) można osiągnąć poprzez zwiększenie powyższych kosztów [98].

Luo et al. wykorzystali algorytm harmoniczny do polioptymalizacji systemu oczyszczania wód gruntowych [232]. Ponadto wielokryterialny algorytm HS zastosowali *Kougias* i *Theodossiou* do minimalizacji czasu pracy pompy oraz minimalizacji zanieczyszczeń przepływającego strumienia wody [192]. Z kolei *Salcedo-Sanz et al.* zaproponowali wielokryterialny algorytm harmoniczny do planowania położenia dróg jednokierunkowych [304].

Opracowane w ramach rozprawy wielokryterialne algorytmy harmoniczne do polioptymalizacji konfiguracji gridowych opierają się na dwóch pamięciach harmonicznych. Pierwsza pamięć harmoniczna pełni rolę archiwum. Druga pamięć harmoniczna jest pamięcią tymczasową, która zawiera *HMS2* utworzonych improwizacji z archiwum (rys. 38). W szczególności pamięć ta może być jednoelementowa.

Archiwum *HM* generowane jest następująco. Po losowym wygenerowaniu rozwiązań wywoływana jest procedura nadawania rang dla alternatyw dopuszczalnych. Natomiast dla rozwiązań niedopuszczalnych wyznaczane są kary za przekroczenie ograniczeń. Następnie obliczane są wartości *fitness*, a *HMS* konfiguracji o najwyższej sprawności zapisywanych jest w głównej pamięci *HM* [205, 224].



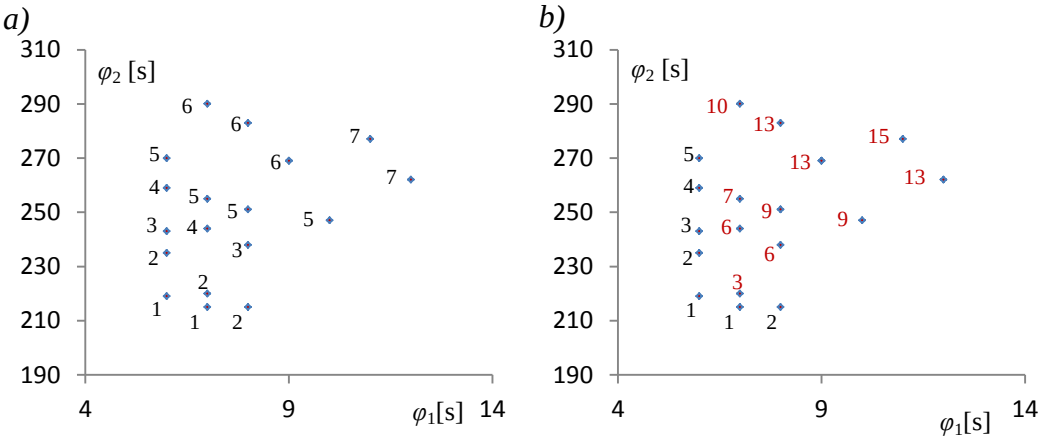
Rys. 38. Diagram procedury aktualizacji pamięci harmonicznej w wielokryterialnych algorytmach harmonicznych klasy MOHS [192]

Następnie za pomocą improwizacji konstruowanych jest *HMS2* konfiguracji, które zapisywane są do tablicy nowych improwizacji *HM2*. Tablice *HM* i *HM2* są łączone, a jej elementy podlegają weryfikacji pod kątem spełnienia ograniczeń. Ponadto konfiguracjom dopuszczalnym nadawane są rangi. Na rysunku 39 zamieszczono rangi dla przykładowych ocen konfiguracji. Dla konfiguracji niedopuszczalnych rangi nie są wyliczane, lecz wyznaczane są wartości funkcji kary.

Zazwyczaj stosuje się dwie procedury nadawania rang: sortowania ocen *Goldberga* i wyznaczania rang w zależności od liczby ocen dominujących *Fonseci-Fleminga*. W procedurze opartej na liczbie poziomów niezdominowanych przyjmuje się, że dwa osobniki mają równe rangi, jeśli są wzajemnie niezdominowane w sensie przyjętej relacji (rys. 39.a). Implikuje to wyznaczanie identycznych wartości *fitness* dla konfiguracji z równymi rangami.

Jeśli wybrana konfiguracja dominuje inną, to otrzymuje ona niższą rangę. Niezdominowane konfiguracje dopuszczalne w konkatenacji tablic otrzymują rangę 1. Następnie są one tymczasowo eliminowane z repozytorium i kolejne niezdominowane alternatywy otrzymują rangę 2. Procedura jest powtarzana aż do redukcji wszystkich rozwiązań dopuszczalnych w konkatenacji tablic [146].

Natomiast podejście *Fonseci-Fleminga* polega na tym, że ranga osobnika w populacji zależy od liczby osobników, które nad nim dominują (rys. 39.b) [104]. Rangi te zazwyczaj są wyższe niż rangi wyznaczone za pomocą liczby niezdominowanych poziomów (rys. 39).



Rys. 39. Rangi wyznaczone na podstawie:
a) liczby poziomów dominacji; b) liczby ocen dominujących.
Źródło: opracowanie własne.

Ocena konfiguracji może cechować się różnymi rangami w zależności od procedury nadawania rang. Zaletą procedury *Fonseci-Fleminga* jest większa różnorodność rang dla konkatenacji tablic, a wadą - fakt, że dwie wzajemnie niedominujące się konfiguracje mogą cechować się różnymi rangami. Identyczną wadę posiada procedura uwzględniająca liczbę niezdominowanych poziomów.

Po wyznaczeniu rang dla rozwiązań dopuszczalnych i kar dla konfiguracji niedopuszczalnych obliczana jest sprawność poszczególnych wariantów. Dla konfiguracji dopuszczalnych x o randze $\eta(x)$ sprawność rozwiązania jest następująca:

$$fitness(x) = \eta_{\max} - \eta(x) + \Phi_{\max} + 1, \tag{4.6.3}$$

gdzie:

$\eta_{\max}$ – maksymalna wartość rangi dla konkatenacji tablic pamięci harmonicznych;

$\Phi_{\max}$ – maksymalna wartość funkcji kary $P(x)$ za niespełnienie ograniczeń.

Dla rozwiązań niedopuszczalnych oblicza się funkcję *fitness*, jak niżej:

$$fitness(x) = \Phi_{\max} - P(x) + 1. \tag{4.6.4}$$

Uformowanie nowej głównej pamięci harmonicznej *HM* polega na tym, że kwalifikuje się do niej *HMS* najlepszych rozwiązań w sensie rangi $\eta(x)$. Weryfikacji podlega łącznie *HMS*+*HMS2* konfiguracji z obu tablic pamięci. Początkowo kwalifikowane są rozwiązania z rangą 1, następnie z rangą 2 aż do momentu rozpoczęcia wypełniania archiwum konfiguracjami cechującymi się rangą graniczną η_{gr}.

Jeśli liczba konfiguracji o randze granicznej η_{gr} jest równa liczbie niezapisanych jeszcze wierszy w nowej *HM*, to wszystkie konfiguracje z rangą graniczną są zapisywane do *HM*. Jeśli jednak konfiguracji tej klasy jest więcej niż wolnych wierszy w *HM*, to powstaje dylemat, które z nich należy zakwalifikować. Pierwszy sposób selekcji konfiguracji z rangą graniczną η_{gr} polega na losowym uzupełnieniu brakujących konfiguracji [33]. Można także wykorzystać dodatkowe kryterium, którego nie uwzględniono w kryterium wektorowym.

Alternatywny sposób selekcji konfiguracji z rangą graniczną polega na tym, że rozważa się zagęszczenie ocen tych konfiguracji między sobą. Konfiguracje o randze η_{gr} są numerowane $1,\ldots,m,\ldots M$ w taki sposób, że:

$$\varphi_1(x(1)) \le \ldots \varphi_1(x(m\text{-}1)) \le \varphi_1(x(m)) \le \varphi_1(x(m+1)) \le \ldots \le \varphi_1(x(M)). \tag{4.6.5}$$

Załóżmy, że należy dokonać wyboru dwóch konfiguracji spośród pięciu (M=5) alternatyw cechujących się rangą graniczną η_{gr} =5 (rys. 40). Konfiguracje

ponumerowano zgodnie z (4.6.5), przy czym oceny konfiguracji nr 3 i 4 są identyczne. W tabeli nr 5 przedstawiono współrzędne ocen konfiguracji z rysunku 40.

Tabela 5. Wartości miary zagęszczenia *DCD* dla wybranych ocen konfiguracji gridu cechujących się rangą graniczną η_{gr}=5

m	φ_1 [s]	φ_2 [s]	*CD*	*VV*	*log (1/VV)*	*DCD*
1	6	270	16	111,25	-2,04630002	-7,8189903
2	7	255	10,5	40,625	-1,60879337	-6,5266306
3	8	251	2,5	13,625	-1,13433651	-2,2039315
4	8	251	3	0,625	0,204119983	14,6972382
5	10	247	6	13	-1,11394335	-5,3862703

Źródło: opracowanie własne.

Kryterium selekcji alternatyw cechujących się identyczną rangą graniczną może polegać na wyborze brakujących rozwiązań z najniższymi wartościami *względnej miary zagęszczenia DCD*, którą wyznaczamy dla zadanej konfiguracji nr *m*, jak niżej [321]:

$$DCD(m) = \frac{CD(m)}{\log \frac{1}{VV(m)}}, \qquad m = \overline{1, M}; \tag{4.6.6}$$

gdzie:

$$CD(m) = \begin{cases} \frac{2}{N}\sum_{n=1}^{N}\left|\omega_n(m+1) - \omega_n(m)\right|, & m = 1; \\ \frac{1}{N}\sum_{n=1}^{N}\left|\omega_n(m+1) - \omega_n(m-1)\right|, & m = \overline{2, M-1}; \\ \frac{2}{N}\sum_{n=1}^{N}\left|\omega_n(m-1) - \omega_n(m)\right|, & m = M. \end{cases}$$

- miara zagęszczenia *m*-tej konfiguracji będąca średnią arytmetyczną odchyłek współrzędnych ocen sąsiednich;

$$VV(m) = \begin{cases} \frac{2}{N}\sum_{n=1}^{N}\left[\left|\omega_n(m+1) - \omega_n(m)\right| - CD(m)\right]^2, & m = 1; \\ \frac{1}{N}\sum_{n=1}^{N}\left[\left|\omega_n(m+1) - \omega_n(m-1)\right| - CD(m)\right]^2, & m = \overline{2, M-1}; \\ \frac{2}{N}\sum_{n=1}^{N}\left[\left|\omega_n(m-1) - \omega_n(m)\right| - CD(m)\right]^2, & m = M. \end{cases}$$

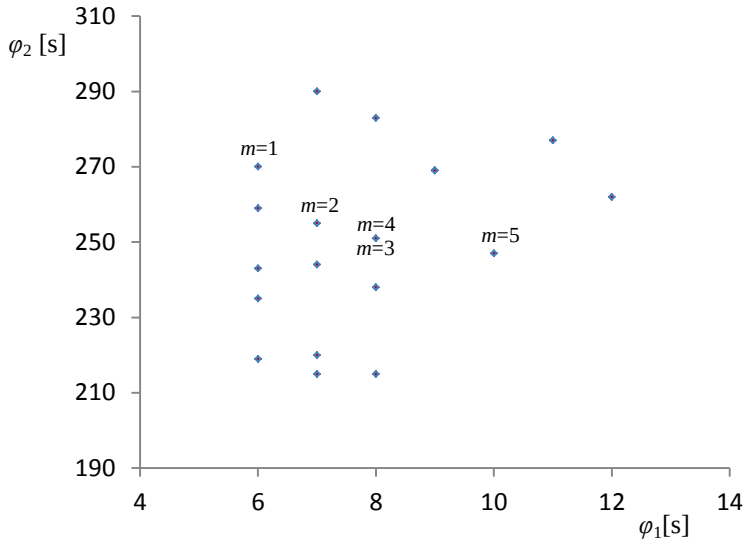
- wariancja odchyłek współrzędnych między ocenami sąsiednimi *m*-tej oceny konfiguracji.

Im mniejsza wartość względnej miary zagęszczenia *DCD*(*m*), tym większe szanse na zakwalifikowanie się *m*-tej oceny cechującej się rangą graniczną. Jeśli jednak miary *DCD* są jednakowe dla kilku rozwiązań, to ustala się kolejność w oparciu o losowy wybór konfiguracji. W wypadku rysunku 40 zakwalifikowano konfiguracje nr 1 i 2, które cechują się najniższym zagęszczeniem w sensie miary *DCD*.

Sąsiedztwo w wypadku miary względnego zagęszczenia wyznaczane jest w oparciu o nr konfiguracji *m*. W odniesieniu do formuły zaproponowanej przez

Sivasubramaniego i *Swarupa* w zależności (4.6.6) wprowadzono dwie konieczne modyfikacje. Ponieważ konfiguracje nr 1 oraz nr M=5 mają tylko jednego sąsiada, odpowiednio nr 2 lub nr M-1=4, a pozostałe konfiguracje mają dwóch sąsiadów, to odpowiednie wartości w funkcji zagęszczenia zwiększane są dwukrotnie dla m=1 oraz m=M.

Ponadto, jeśli dwie sąsiednie konfiguracje cechują się identyczną oceną, to rozwiązania te numerowane są kolejno, np. m oraz m+1, a następnie wyliczane są odchyłki współrzędnych $\left|\omega_n(m+1)-\omega_n(m-1)\right|$ uwzględniające pary sąsiadów.



Rys. 40. Przykład indeksowania konfiguracji cechujących się rangą graniczną
Źródło: opracowanie własne.

Wadą względnej miary zagęszczenia *DCD* jest wyznaczenie różnych wartości dla dwóch konfiguracji cechujących się identyczną oceną (tab. 5). Oceny 3 i 4 o identycznych współrzędnych (8; 251) otrzymały wartości zagęszczenia odpowiednio -2,2039315 i 14,6972382. Nieco zaskakujące jest pominięcie w wyniku selekcji oceny nr 5 (rys. 40).

Warto zauważyć, że w wypadku skrajnych indeksów m=1 lub m=M należy podwoić wartość miary zagęszczenia. Kolejną wadą jest fakt, że zagęszczenie wyznaczane jest w oparciu o trzy konfiguracje, a nie wszystkie z rangą graniczną. Wreszcie miara zagęszczenia *DCD* o ujemnych wartościach nie jest intuicyjnie prosta do interpretacji.

Z powyższych powodów w rozprawie zaproponowano *geometryczną miarę zagęszczenia GCD* (ang. *geometric crowding distance*) w znormalizowanej przestrzeni

wielokryterialnej. Współrzędne ocen konfiguracji cechujących się rangą graniczną podlegają normalizacji przedziałowej od 0 do 1, zgodnie z poniższą formułą:

$$\bar{\varphi}_n(m) = \frac{\varphi_n(m) - \varphi_n^{\min}}{\varphi_n^{\max} - \varphi_n^{\min}}, \quad m = \overline{1, M}; n = \overline{1, N}, \tag{4.6.7}$$

gdzie:

$\bar{\varphi}_n(m)$ - wartość znormalizowanej przedziałowo *n*-tej współrzędnej oceny konfiguracji z rangą graniczną;

$\varphi_n^{\max}$ - maksymalna wartość *n*-tej współrzędnej ocen cechujących się rangą graniczną;

$\varphi_n^{\min}$ - minimalna wartość *n*-tej współrzędnej ocen cechujących się rangą graniczną.

Geometryczną miarę zagęszczenia *GCD* definiuje się jako łączną odległość *Euklidesa* w znormalizowanej przestrzeni wielokryterialnej między *m*-tą oceną a pozostałymi ocenami cechującymi się rangą graniczną, co zapisuje się następująco:

$$GCD(m) = \sum_{\substack{k=1 \\ k \neq m}}^{M} \sqrt{\sum_{n=1}^{N} \left(\bar{\varphi}_n(m) - \bar{\varphi}_n(k)\right)^2}, \qquad m = \overline{1, M}. \tag{4.6.8}$$

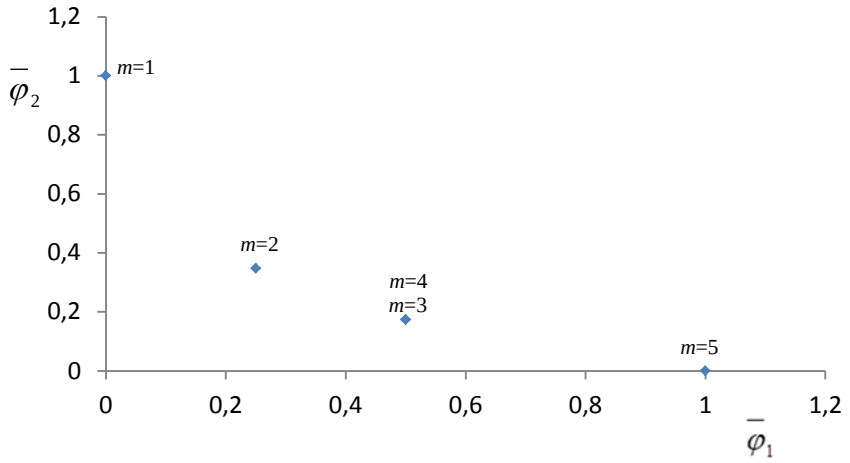
W tabeli nr 6 przedstawiono wartości geometrycznej miary zagęszczenia dla konfiguracji cechujących się η_{gr}=5 (rys. 40).

Tabela 6. Wartości geometrycznej miary zagęszczenia *GCD* dla wybranych ocen konfiguracji gridu cechujących się rangą graniczną η_{gr}=5

m	*Macierz odległości między ocenami*					$\bar{\varphi}_1$	$\bar{\varphi}_2$	*GCD*
	1	*2*	*3*	*4*	*5*			
1	0	0,698449	0,965619	0,965619	1,414214	0	1	4,043900
2	0,698449	0	0,304542	0,304542	0,82673	0,25	0,34783	2,134263
3	0,965619	0,304542	0	0	0,529382	0,5	0,17391	1,799543
4	0,965619	0,304542	0	0	0,529382	0,5	0,17391	1,799543
5	1,414214	0,82673	0,529382	0,529382	0	1	0	3,299709

Źródło: opracowanie własne.

Na podstawie wartości *GCD* należy zakwalifikować dwie oceny nr 1 i 5, które są najbardziej odległe od pozostałych (rys. 41). Warto podkreślić, że zgodnie z miarą *DCD* ocena nr 5 nie zostałaby zakwalifikowana do archiwum *HM*. Na rysunku 41 przedstawiono znormalizowane oceny konfiguracji.



Rys. 41. Znormalizowane oceny konfiguracji cechujące się rangą graniczną
Źródło: opracowanie własne.

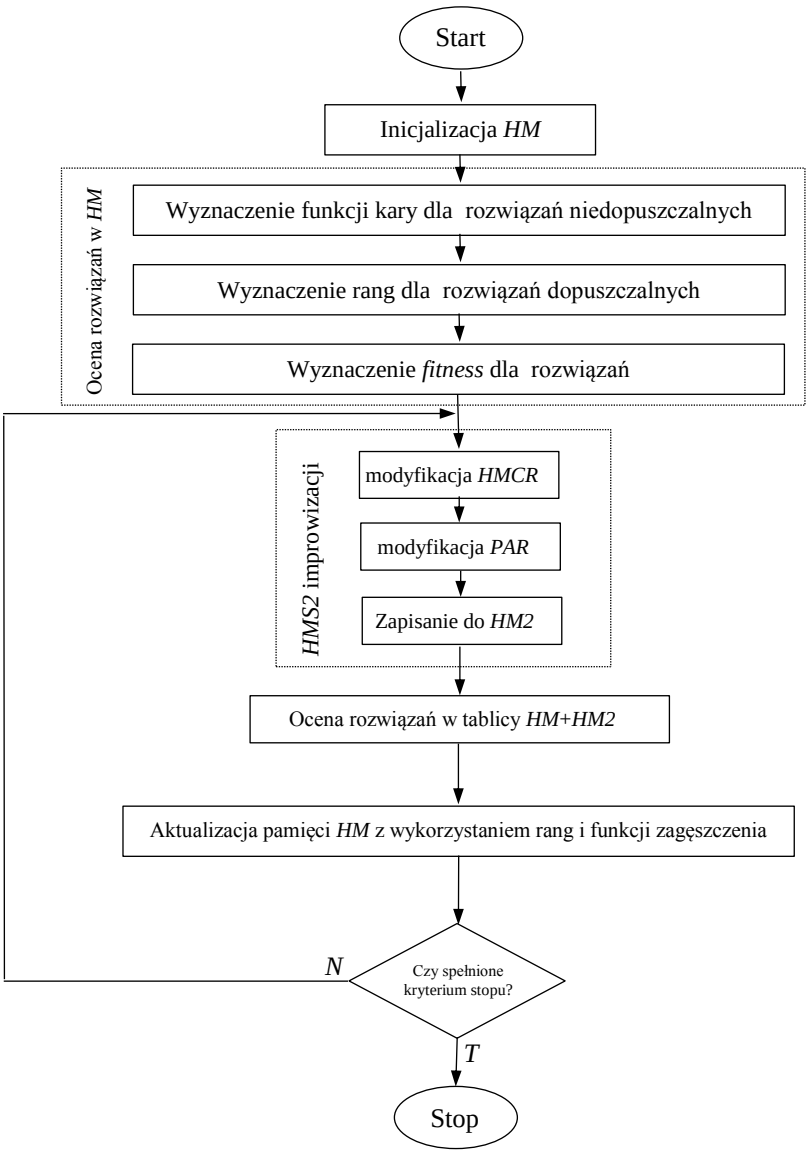
Na rysunku 42 zaprezentowano diagram zaproponowanej wielokryterialnej metaheurystyki przeszukiwania harmonicznego MOHS (ang. *Multi-objective Harmony Search*), w której uwzględnia się także rozwiązania niedopuszczalne. Warto podkreślić, że wielokryterialne algorytmy harmoniczne w literaturze przedmiotu różnią się przede wszystkim procedurami nadawania rang i miarami zagęszczenia. Ponadto nie uwzględnia się w nich rozwiązań niedopuszczalnych.

Metaheurystykę MOHS można zaimplementować w postaci różnych wariantów w zależności od procedury nadawania rang, funkcji zagęszczenia lub rozmiaru pamięci pomocniczej. Algorytm MOHS1 cechuje się wykorzystaniem procedury nadawania rang opartą na sortowaniu poziomów niezdominowania, a także losową selekcją konfiguracji z rangą graniczną. Algorytm MOHS2 różni się od MOHS1 tym, że selekcja konfiguracji cechujących się rangą graniczną zachodzi za pomocą funkcji *GCD*.

Algorytm MOHS3 cechuje się wykorzystaniem procedury nadawania rang z wykorzystaniem liczby dominujących ocen, a także losową selekcją konfiguracji z rangą graniczną. Natomiast algorytm MOHS4 cechuje się wykorzystaniem procedury nadawania rang w oparciu o liczbę dominujących ocen, a także selekcją konfiguracji cechujących się rangą graniczną za pomocą funkcji zagęszczenia *GCD*.

Ponadto każdy z ww. algorytmów może być rozpatrywany w zależności od rozmiaru pamięci pomocniczej *HMS2*. Zazwyczaj rozpatruje się algorytmy dla wariantu *HMS2*=1 lub *HMS2=HMS*. Warto zatem zbadać jakość konfiguracji wyznaczanych za pomocą algorytmów, w których 1<*HMS2*<*HMS* oraz *HMS2*>*HMS*. Z powyższych powodów w aplikacji ADAIORG'14 możliwe są wybory dotyczące procedury

nadawania rang, selekcji ocen z rangą graniczną, rozmiaru głównej pamięci harmonicznej oraz wielkości tymczasowej pamięci harmonicznej.



Rys. 42. Diagram wielokryterialnej metaheurystyki harmonicznej MOHS
Źródło: opracowanie własne

Złożoność obliczeniowa algorytmu MOHS1 z procedurą nadawania rang i losową selekcją ocen z rangą graniczną jest $O(n^7)$. Wyznaczenie wartości współrzędnej $\varphi_2(x) = \tilde{Z}(x)$ cechuje się największą złożonością obliczeniową $O(n^5)$ wśród funkcji skalarnych, przy czym $n=\max\{I, V, J\}$. Podczas procesu improwizacji współrzędne wyznaczane są $NG_{\max}(HMS+HMS2)$ razy, a zatem złożoność obliczeniowa procedury improwizacji jest $O(n^7)$, gdzie $n=\max\{I, V, J, NG_{\max}, HMS, HMS2\}$. Można pokazać, że pozostałe algorytmy klasy MOHS cechują się także złożonością $O(n^7)$.

4.7. Algorytmy harmoniczne do wyznaczania konfiguracji kompromisowych

Do wyznaczania kompromisowych konfiguracji gridu zaproponowano dwa algorytmy harmoniczne: CHS1 oraz CHS2. W pierwszym etapie działania algorytmu CHS1 wyznacza się reprezentację konfiguracji *Pareto*-optymalnych $\boldsymbol{X}^{\text{sel}}$ dla zagadnienia optymalizacji wektorowej (3.4.7) lub (3.4.8) za pomocą wielokryterialnej metaheurystyki harmonicznej MOHS. Oceny cząstkowe konfiguracji ze zbioru $\boldsymbol{X}^{\text{sel}}$ są zapisane w pamięci harmonicznej *HM*. Algorytm CHS1 operuje zatem na konfiguracjach dopuszczalnych. Jeśli w zbiorze $\boldsymbol{X}^{\text{sel}}$ nie ma konfiguracji dopuszczalnych, to stosowany jest algorytm CHS2.

W drugim etapie działania algorytmu CHS1 wykorzystuje się zbiór reprezentacji konfiguracji *Pareto*-optymalnych $\boldsymbol{X}^{\text{sel}}$ do wyznaczenia punktu idealnego ω^{inf}. Natomiast ocenę antyidealną ω^{sup} wyznacza się w oparciu o oszacowanie teoretyczne zweryfikowane za pomocą ocen otrzymanych podczas przeszukiwań w zbiorze $\boldsymbol{X}^{\text{search}}$. Powyższe zapisujemy zgodnie z poniższymi zależnościami:

$$\omega_n^{\text{inf}} = \min_{x \in \boldsymbol{X}^{sel}} \varphi_n(x), \quad n = \overline{1, N}, \tag{4.7.1}$$

$$\omega_n^{\text{sup}} = \max_{x \in \boldsymbol{X}^{sel} \cup \boldsymbol{X}^{search}} \varphi_n(x), \quad n = \overline{1, N}. \tag{4.7.2}$$

Kryteria skalarne $\varphi_n \; n = \overline{1, N}$ odpowiadają kryteriom w zagadnieniu (3.4.7) lub (3.4.8). Przykładowo dla zagadnienia (3.4.7) zachodzi $\varphi_1(x) = \hat{Z}_{\max}(x)$ oraz $\varphi_2(x) = \tilde{Z}_{\max}(x)$.

W trzecim etapie ze zbioru $\boldsymbol{X}^{\text{sel}}$ wybierana jest konfiguracja kompromisowa w oparciu o minimalną odległość oceny od punktu idealnego w znormalizowanej przestrzeni kryterialnej, co można zapisać jak niżej:

$$\zeta_p(x^{komp}) = \min_{x \in \boldsymbol{X}^{sel}} \zeta_p(x). \tag{4.7.3}$$

W szczególności dla p=2 można wyznaczyć konfigurację *Salukwadze* x^{Sal} za pomocą metody przeglądu zbioru $\boldsymbol{X}^{\text{sel}}$ i wyboru minimalnej odległości konfiguracji od znormalizowanego punktu idealnego, jak niżej:

$$\zeta_2(x^{Sal}) = \min_{x \in \boldsymbol{X}^{sel}} \sqrt{\sum_{n=1}^{N} \left(\frac{\varphi_n(x) - \omega_n^{\min}}{\omega_m^{\max} - \omega_n^{\min}} \right)^2}. \tag{4.7.4}$$

Warto podkreślić, że złożoność obliczeniowa algorytmu CHS1 zależy od złożoności obliczeniowej metaheurystyki MOHS wykorzystywanej w pierwszym etapie, która jest $O(n^7)$. Można pominąć zatem złożoność obliczeniową drugiego etapu CHS1 $O(n^2)$, gdzie n=max{*HMS, N*}. W tej fazie analizuje się *HMS* elementów ze zbioru $\boldsymbol{X}^{\text{sel}}$, a także wyznacza się *N* współrzędnych punktu idealnego i *N* współrzędnych punktu antyidealnego. Złożoność obliczeniową trzeciego etapu algorytmu CHS1 to także O(n^2), gdzie n=max{*HMS, N*}. W tej fazie przegląda się *HMS* elementów ze zbioru $\boldsymbol{X}^{\text{sel}}$ w celu wyznaczenia odległości ocen od punktu idealnego oraz wyznaczenia konfiguracji o najmniejszej odległości. Ponieważ oceny rozwiązań wyznaczono w pierwszym etapie, to wyznaczanie odległości między oceną a punktem idealnym polega na wykonaniu kilku operacji arytmetycznych.

Algorytm CHS1 można zmodyfikować w sytuacji decyzyjnej, w której ze zbioru $\boldsymbol{X}^{\text{sel}}$ wybiera się preferowaną konfigurację ze względu na dodatkowe kryterium, którego nie uwzględniono w zagadnieniu (3.4.7) lub (3.4.8) jako kryterium skalarne.

Natomiast algorytm CHS2 (rys. 43) umożliwia wyznaczanie konfiguracji *Salukwadze* dla zagadnienia (3.5.16). O ile w algorytmie CHS1 wykorzystuje się algorytm MOHS w jednej z czterech jego wersji, o tyle w CHS2 stosuje się procedury z algorytmu HS$\Delta_{\max}$. Po wprowadzeniu danych wejściowych dla zagadnienia (3.5.16) wyznacza się oszacowania punktu idealnego, jak niżej:

$$\omega^{\inf} = \left[\hat{Z}_{\max}^{\min}; 0\right], \tag{4.7.5}$$

gdzie $\omega_1^{\inf} = \hat{Z}_{\max}^{\min} = \max\limits_{v=1,V} \min\limits_{j=1,J} \left\{ t_{vj} \right\}$.

Ponadto na podstawie macierzy $\boldsymbol{T}$ oraz $\boldsymbol{\tau}$ wyznacza się następujące oszacowanie teoretyczne oceny antyidealnej:

$$\omega^{\sup} = \left[\hat{Z}_{\max}^{\max}, \tilde{Z}_{\max}^{\max}\right], \tag{4.7.6}$$

gdzie:

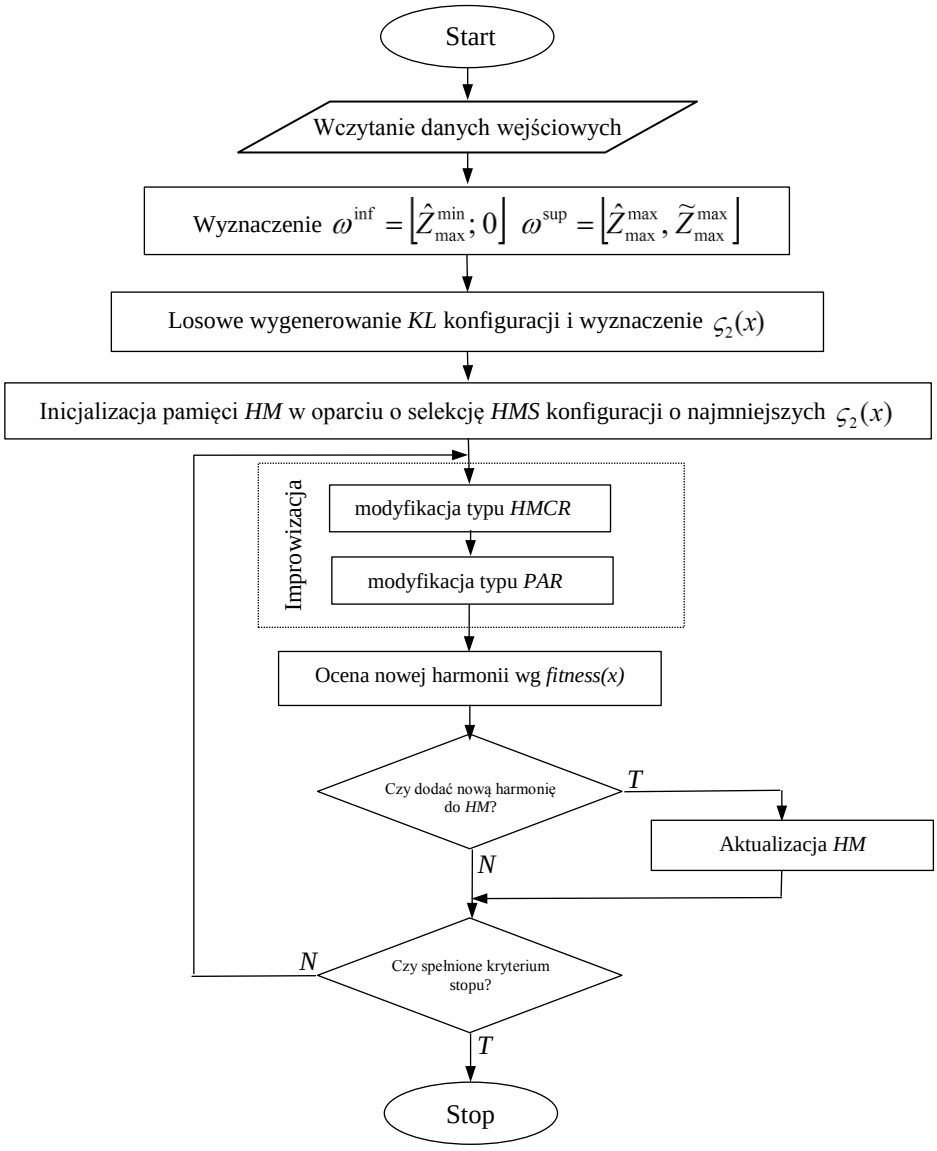
$$\omega_1^{\sup} = \hat{Z}_{\max}^{\max} = \sum_{v=1}^{V} \max_{j=1,J} \left\{ t_{vj} \right\},$$

$$\omega_2^{\sup} = \tilde{Z}_{\max}^{\max} = \sum_{v=1}^{V} \sum_{u=1}^{U} \max_{i=1,I;\, k=1,I} \left\{ \tau_{vuik} \right\}.$$

Następnie *KL* razy (*KL>HMS*) losowo generowane są konfiguracje, spośród których wybiera się *HMS* rozwiązań o największej wartości *fitness*. Jeśli konfiguracja

jest dopuszczalna, to wyznaczana jest odległość $\zeta_2(x)$ za pomocą następującej zależności:

$$\zeta_2(x) = \sqrt{\left(\frac{\hat{Z}_{\max}(x) - \hat{Z}_{\max}^{\min}}{\hat{Z}_{\max}^{\max} - \hat{Z}_{\max}^{\min}}\right)^2 + \left(\frac{\tilde{Z}_{\max}(x)}{\tilde{Z}_{\max}^{\max}}\right)^2}. \tag{4.7.7}$$



Rys. 43. Diagram algorytmu harmonicznego CHS2 do wyznaczania konfiguracji Salukwadze dla zagadnieniu polioptymalizacji (3.5.16)
Źródło: opracowanie własne

Wartość funkcji *fitness* wyznaczana jest następująco:

$$fitness(x) = \begin{cases} \zeta_2(x) + \Phi_{\max} + 1, & x \in \boldsymbol{X}; \\ \Phi_{\max} - P(x) + 1, & x \notin \boldsymbol{X}, \end{cases} \tag{4.7.8}$$

gdzie $\Phi_{\max}$ – maksymalna wartość funkcji kary $P(x)$ za niespełnienie ograniczeń.

W powyższy sposób inicjalizowana jest pierwsza wersja tablicy harmonicznej *HM*. Kolejne kroki algorytmu CHS2 są prawie identyczne, jak algorytmu HSΔmax, przy czym wartości funkcji sprawności obliczane są za pomocą zależności (4.7.8).

Złożoność obliczeniowa algorytmu CHS2 zależy od nakładów na wyznaczenie wartości funkcji skalarnych oraz od złożoności procedury weryfikacji dopuszczalności konfiguracji. Ponieważ wyznaczenie wartości współrzędnej $\varphi_1(x) = \hat{Z}_{\max}(x)$ cechuje się złożonością $O(n^3)$, a $\varphi_2(x) = \tilde{Z}_{\max}(x)$ - $O(n^5)$, gdzie n=max{*I, V, J*}, to nakłady na wyznaczenie odległości oceny konfiguracji od punktu idealnego są proporcjonalne do n^6 w wypadku *N* kryteriów cząstkowych.

Złożoność obliczeniowa procedury weryfikacji spełnienia ograniczeń zależy od złożoności procedur wyznaczania: łącznej wydajności, kosztów zakupu komputerów, stopnia rozproszenia, zajętości pamięci RAM i HD, a także poboru mocy. Można pokazać, że złożoność obliczeniowa procedury weryfikacji jest $O(n^2)$. Zatem inicjacja pamięci *HM*, w której *KL* razy losowo generuje się konfiguracje gridu cechuje się złożonością obliczeniową $O(n^7)$, gdzie n=max{*I, V, J, KL*}. Uwzględniając proces improwizacji i aktualizacji pamięci *HM*, złożoność obliczeniowa jest $O(n^8)$, przy czym n=max{*I, V, J,* $NG_{\max}$}.

Zmieniając parametr p z 2 na 1 lub $p \to \infty$, zmienia się p-normę, a w konsekwencji funkcję celu w zagadnieniu (3.5.16). Konfiguracje kompromisowe mogą zatem różnić się między sobą dla różnych parametrów p. Warto zauważyć, że każda konfiguracja kompromisowa dla $p \in [1,\infty)$ jest efektywna. Natomiast jeśli $p \to \infty$, to rozwiązanie kompromisowe może nie być konfiguracją *P*-optymalną [17].

4.8. Wnioski i uwagi

W rozdziale zaproponowano siedem nowych algorytmów optymalizacji konfiguracji gridu, które wywodzą się z metaheurystyki przeszukiwania harmonicznego zaproponowanej przez *Kima Geema* w 2000 roku. Algorytmy harmoniczne zaimplementowano w języku *Java* w ramach pakietu ADAIORG'14.

Algorytm HSΔmax umożliwia wyznaczanie co najwyżej *HMS* konfiguracji suboptymalnych, które mogą być optymalne lub też mogą cechować się większą wartością funkcji celu $\Delta_{\max}$ niż optimum. Algorytm HSΔmax umożliwia wyznaczanie rozwiązań suboptymalnych zagadnień programowania binarnego (3.1.1), (3,2,1) oraz

(3.3.1). Algorytm HSΔmax można przystosować do optymalizacji innych kryteriów oceny konfiguracji gridu, w tym wydajności, kosztu, dostępności czy poboru mocy.

W HSΔmax wprowadzono szereg istotnych modyfikacji w odniesieniu do jego podstawowej wersji, w tym kodowanie całkowitoliczbowe konfiguracji oraz procedurę dywersyfikacji. Opracowano nowe zależności na wyznaczanie w każdej improwizacji wartości zmiennych decyzyjnych zgodnie z trójwartościową regułą ruletki. W kryterium stopu bada się dodatkowo czy nie zaszła przedwczesna zbieżność algorytmu, co może się zdarzyć, kiedy stan obliczeń utknie w maksimum lokalnym funkcji sprawności. Przedwczesnej zbieżności algorytmu harmonicznego zapobiega się za pomocą dodatkowego warunku przy wprowadzeniu improwizacji do *HM*. Sprawdza się, czy w tablicy istnieje identyczna konfiguracja jak improwizacja kandydująca.

Istotnym rozszerzeniem w proponowanych algorytmach jest wprowadzenie dwóch parametrów *BW1* i *BW2*, za pomocą których z prawdopodobieństwem (1-*HMCR*)*PAR* modyfikowane są zmienne decyzyjne. W wypadku zmiennej określającej węzeł wykonywania modułu wykorzystuje się parametr *BW1*, a w wypadku zmiennej określającej rodzaj komputera w węźle – *BW2*.

Algorytmy harmoniczne stosowane są do wyznaczania rozwiązań optymalnych w sensie *Pareto* od 2011 roku. Nieznane są jednak ich zastosowania do rozwiązywania zagadnień polioptymalizacji kombinatorycznej, a tym bardziej do optymalizacji konfiguracji gridu. Dlatego też w rozprawie skonstruowano cztery algorytmy klasy MOHS, które wykorzystują procedurę nadawania rang *Goldberga* lub *Fonseci-Fleminga*. Ponadto algorytmy te różnią się ze względu na dobór konfiguracji cechujących się rangą graniczną do pamięci harmonicznej, która to selekcja może być losowa lub może wykorzystać oryginalną miarę *GCD*. Współrzędne ocen konfiguracji cechujących się rangą graniczną podlegają normalizacji przedziałowej. Ponadto wartość miary zagęszczenia wyznaczana jest w oparciu o rozwiązania z rangą graniczną.

Do wyznaczania konfiguracji kompromisowych zaproponowano dwa algorytmy CHS1 oraz CHS2, które różnią się tym, że algorytm CHS1 dokonuje selekcji konfiguracji kompromisowej z reprezentacji zbioru rozwiązań *Pareto*-suboptymalnych. Istotną rolę w wyznaczeniu tej reprezentacji odgrywa algorytm klasy MOHS. Wyznaczenie współrzędnych punktu idealnego w algorytmie CHS1 następuje na podstawie współrzędnych ocen ze zbioru otrzymanego za pomocą MOHS.

Natomiast w algorytmie CHS2 minimalizuje się odległość *Euklidesa* między punktem idealnym a oceną konfiguracji. Wyznaczenie współrzędnych punktu idealnego

następuje na podstawie oszacowania teoretycznego. Złożoność obliczeniowa algorytmów MOHS oraz CHS jest $O(n^7)$.

Warto podkreślić, że spektrum zastosowań metaheurystyki harmonicznej jest bardzo szerokie: zagadnienia inżynierskie, energetyka, zarządzanie zasobami wodnymi, medycyna czy robotyka. W literaturze przedmiotu studiowane są instancje, dla których za pomocą algorytmu harmonicznego uzyskano rozwiązania wyższej jakości niż za pomocą algorytmu genetycznego. Z tego powodu w pracy zaprezentowano nowatorskie zastosowania *harmony search* w rozproszonych systemach komputerowych klasy grid.

PODSUMOWANIE

Oryginalnym dorobkiem autora jest przedstawiony w niniejszej rozprawie system metodologiczny do wyznaczania konfiguracji gridów, który może wspomagać zarządzanie zasobami. Opracowano modele konfiguracji gridów, a także omówiono kryteria i ograniczenia, na podstawie których sformułowano adekwatne wektorowej zagadnienia optymalizacji.

Problemy optymalizacji wykorzystania zasobów w gridzie opierają się na minimalizacji ważonego obciążenia procesorów w newralgicznym komputerze i obciążenia komunikacyjnego w newralgicznym serwerze przy uwzględnieniu ograniczeń takich jak: zagwarantowanie planowanej wydajności gridu, nieprzekroczenie kosztów zakupu komputerów, wymuszenia odpowiedniego stopnia rozproszenia modułów czy też postulatu nieprzekroczenia wielkości dostępnej pamięci RAM czy pamięci dyskowej. Uwzględniane są także inne wymagania, takie jak żądanie nieprzekroczenia zadanego obciążenia procesorów newralgicznego hosta, czy też zapewnienie odpowiedniego poziomu dostępności gridu. Istotne jest rozważanie kwestii ekologicznej związanej z ograniczeniem poboru mocy energii elektrycznej.

Do oceny konfiguracji zaproponowano dwa inne kryteria: obciążenie procesorów w newralgicznym serwerze, a także obciążenie komunikacyjne newralgicznego hosta. Można także rozważać minimalizację łącznego obciążeniu gridu lub łączne obciążenia newralgicznego komputera. Ponadto administrator gridu może maksymalizować wydajność systemu.

Minimalizacji może podlegać koszt realizacji modułów w gridzie, koszt zakupu hostów, a także pobór mocy energii przez komputery gridu. Maksymalizacji podlega wielkość dostępnej pamięci RAM lub rezerwa pamięci dyskowej w newralgicznym komputerze. Ponadto można rozważać maksymalizację dostępności gridu lub stopnia rozproszenia modułów. Warto wspomnieć, że na wartości każdego z powyższych kryteriów mogą być nałożone ograniczenia.

Do rozwiązania sformułowanych problemów optymalizacji konfiguracji gridu, autor opracował siedem algorytmów harmonicznych, w tym: *HSΔmax*, cztery algorytmy klasy MOHS, oraz algorytmy CHS1 i CHS2. Aplikacje te napisane w języku programowania *Java* w ramach pakietu ADAIORG'14 umożliwiają powtórzenie uzyskanych wyników badań i mogą być zastosowane do wyznaczania konfiguracji

gridów na podstawie sformułowanych oryginalnych zagadnień optymalizacji wielokryterialnej (3.4.7), (3.4.8) i (3.5.16). W szczególności sformułowano dwa zagadnienia wyznaczania reprezentacji konfiguracji optymalnych w sensie *Pareto* z dwoma i pięcioma kryteriami. Postawiono także problem wyznaczania kompromisowych konfiguracji *Salukwadze* dla dwóch kryteriów.

Ważnym wynikiem teoretycznym jest twierdzenie 3.2 o ograniczoności i skończoności zbioru ocen dla zagadnienia polioptymalizacji (3.4.7). Wykazano, że dla niepustego zbioru dopuszczalnych konfiguracji gridu $\tilde{\boldsymbol{X}}$ oraz funkcji kryterium $\varphi(x) = [\hat{Z}_{\max}(x), \tilde{Z}_{\max}(x)]^T$, $\varphi(x) \in \boldsymbol{R}^2$ dla $x \in \tilde{\boldsymbol{X}}$, zbiór wyników $\boldsymbol{\Omega} = \varphi(\tilde{\boldsymbol{X}})$ jest zbiorem niepustym i ograniczonym. W rezultacie do rozwiązania problemu (3.4.7) można stosować algorytmy klasy MOHS.

Warto również podkreślić, że sformułowane zagadnienia optymalizacji wektorowej w zakresie konfiguracji *Pareto*-optymalnych i konfiguracji kompromisowych dla p=2, a także opracowane algorytmy harmoniczne MOHS, CHS1 i CHS2 stanowią oryginalny dorobek naukowy zamieszczony w pracy. Za pomocą algorytmów klasy MOHS możliwe jest wyznaczanie konfiguracji suboptymalnych w sensie *Pareto*, na podstawie których następuje selekcja konfiguracji hierarchicznych lub alternatyw kompromisowych. Natomiast za pomocą algorytmu CHS2 wyznaczane są bezpośrednio konfiguracje kompromisowe dla p=2. Algorytm CHS2 może być także zmodyfikowany pod kątem poszukiwania innej klasy kompromisowej konfiguracji.

Dokonano weryfikacji opracowanych algorytmów dla szeregu instancji testowych, z których niektóre omówiono w dodatku do rozprawy. Uzyskane wyniki eksperymentalne potwierdziły słuszność przeprowadzonych rozważań w odniesieniu do modelu, sformułowanych zagadnień polioptymalizacji i zaproponowanych algorytmów przeszukiwania harmonicznego.

Przesłankami do stwierdzenia, że sformułowany problem badawczy został poprawnie rozwiązany są przede wszystkim skonstruowane metody optymalizacji, które bazują na algorytmach *harmony search*. Natomiast postawioną hipotezę roboczą można uznać za tezę naukową, gdyż została naukowo zweryfikowana z pozytywnym skutkiem. Wobec powyższego cel rozprawy został osiągnięty.

Skonstruowane modele, sformułowane zadania optymalizacji oraz opracowane metody stanowią wkład autora do przetwarzania rozproszonego, sztucznej inteligencji i optymalizacji wektorowej, a także mają istotne znaczenie praktyczne w przetwarzaniu

gridowym. Rozprawa nie wyczerpuje bogatej tematyki dotyczącej optymalizacji wykorzystania zasobów w gridach za pomocą algorytmów *harmony search*, lecz stanowi fundament do kontynuacji badań. Jednym z mankamentów rozprawy wydaje się pominięcie interesującego problemu systematycznej rekonfiguracji gridu za pomocą reguł zarządzania zasobami w dynamicznym środowisku, co wynika z ograniczonych ram rozprawy.

Modele, opracowane metody i towarzyszące im implementacje algorytmów były dotychczas stosowane przez autora w ramach pracy naukowo-badawczej prowadzonej na Politechnice Gdańskiej przy wdrożeniu eksperymentalnego gridu *Comcute*, który implementuje paradygmat wolontariatu obliczeniowego. Wybrane zagadnienia autor zaprezentował także na konferencjach naukowych.

Kierunkiem dalszych badań w sztucznej inteligencji, optymalizacji wielokryterialnej i systemach rozproszonych będzie rozwijanie metod *harmony search* w celu rozwiązywania innych zagadnień polioptymalizacji, w tym dla ciągłych zmiennych decyzyjnych. Perspektywicznym kierunkiem badań stanowią prace nad implementacją opracowanych algorytmów *harmony search* jako algorytmy kwantowe.

Reasumując, warto podkreślić, że rozwój teorii i zastosowań algorytmów przeszukiwania harmonicznego w zagadnieniach wyznaczania konfiguracji gridu jest istotnym kierunkiem badawczym w zakresie sztucznej inteligencji, optymalizacji wektorowej, a także przetwarzania rozproszonego.

BIBLIOGRAFIA

1. Abelson H., Sussman G. J.: *Struktura i interpretacja programów komputerowych*. WNT, Warszawa 2002.
2. Afshari S., Aminshahidy B., Pishvaie M.R.: *Application of an improved harmony search algorithm in well placement optimization using streamline simulation*. J. Petrol. Sci. Eng., Vol. 78, No 3–4, 2011, pp. 664–678.
3. Ahmad I., Mohammad G.H., Salman A.A,. Hamdan S.A.: *Broadcast scheduling in packet radio networks using harmony search algorithm*. Expert Syst. Appl., Vol. 39, 2012, pp. 1526–1535.
4. Ahmed A.M., Bakar A.A., Hamdan A.R.: *Harmony search algorithm for optimal word size in symbolic time series representation.* Proc. Conf. on Data Mining and Optimization, Malaysia, 2011, pp. 57–62.
5. Ajith A.P., Murthy C.S.R.: *An improved algorithm for module allocation in distributed computing systems*. Journal of Parallel and Distributed Computing, Vol. 42, No. 1, November 1997, pp. 82-90.
6. Ajith A.P., Murthy C.S.R.: *Algorithms for reliability-oriented module allocation in distributed computing systems*. Journal of System Software, Vol. 40, No. 2, 1998, pp. 125-138.
7. Alatas B.: *Chaotic harmony search algorithms*. Appl. Math. Comput., Vol. 216, No. 9, 2010, pp. 2687–2699.
8. Alberto I., Azcarate C., Mateo P.M.: *Multiobjective evolutionary algorithms. Pareto rankings.* Monografias del Seminario Matematico Garcia de Galdeano, Vol. 27, 2003, pp. 27-35.
9. Al-Betar M.A., Doushc I.A., Khadera A.T., Awadallaha M.A.: *Novel selection schemes for harmony search*. Appl. Math. Comput., Vol. 218, No 10, 2012, pp. 6095–6117.
10. Al-Betar M.A., Khader A.T., Zaman M.: *University course timetabling using a hybrid harmony search metaheuristic algorithm*. IEEE Trans. Syst. Man Cybern: Part C: Appl. Rev., Vol. 42, 2012, pp. 66–681.
11. Al-Betar M.A., Khader A.T., Liao I.Y.: *A harmony search with multi-pitch adjusting rate for the university course timetabling*. In: Recent Advances In Harmony Search Algorithm, Studies in Computational Intelligence, Vol. 270, 2010, pp. 147–161.
12. Alexandre E., Cuadra L., Gil-Pita R.: *Sound classification in hearing aids by the harmony search algorithm.* Stud. Comput. Intell., Vol. 191, 2009, pp. 173–188.
13. Alia O.M., Mandava R., Aziz M.E.: *A hybrid harmony search algorithm to MRI brain segmentation*. In: IEEE Int. Conf. on Cognitive Informatics, Malaysia, 2010, pp. 712–721.
14. Alia O.M., Mandava R.: *The variants of the harmony search algorithm: an overview*. Artif. Intell. Rev., Vol. 36, No. 1, 2011, pp. 49–68.
15. Alsewari A.R.A., Zamli K.Z.: *Design and implementation of a harmony-search-based variable-strength T-way testing strategy with constraints support.* Inform. Software Tech., Vol. 54, No. 6, 2012, pp. 553–568.
16. Alsewari A.A., Zamli K.Z.: *Interaction test data generation using harmony search algorithm.* In: IEEE Symposium on Industrial Electronics and Applications, 2011, pp. 559–564.
17. Ameljańczyk A.: O*ptymalizacja wielokryterialna w problemach sterowania i zarządzania*. PAN, Warszawa 1984.
18. Ameljańczyk A.: *Teoria gier i optymalizacja wektorowa*. WAT, Warszawa 1980.
19. Andrews G. R.: *Concurrent programming. Principles and practice.* The Benjamin/Cummings Publishing Company, Redwood City 1991.
20. Andrieux A., Czajkowski K., Dan A., Keahey K., Ludwig H., Nakata T., Pruyne J., Rofrano J., Tuecke S., Xu M.: *Web Services Agreement Specification (WS-Agreement)*, GFD-R-P.107. Open Grid Forum, 2007.
21. Aneja Y.P., Nair K.P.K.: *Bicriteria transportation problem*. Management Science, Vol. 25, No. 1, 1979, pp. 73-78.
22. Arabas J.: *Wykłady z algorytmów ewolucyjnych*. WNT, Warszawa 2004.
23. Armand P.: *Finding all maximal efficient faces in multiobjective linear programming*. Mathematical Programming. Vol. 61, No. 1-3, 1993, pp. 357-375.
24. Askarzadeh A., Rezazadeh A.: *An innovative global harmony search algorithm for parameter identification of a PEM fuel cell model*. IEEE Trans. Ind. Electron., Vol. 59, No. 9, 2012, pp. 3473–3480.
25. Ayachi I., Kammarti R., Ksouri M., Borne P.: *Application of harmony search to container storage location*. IEEE Int. Conf. on Systems, Man, and Cybernetics, France, 2011, pp. 1556–1561.
26. Ayob M., Hadwan M., Ahmad M., Ahmad Z.: *Enhanced harmony search algorithm for nurse rostering problems.* Journal of Applied Sciences, Vol. 13, No. 6, June 2013, pp. 846-853.
27. Ayvaz M.T.: *Application of harmony search algorithm to the solution of groundwater management models.* Adv. Water Resour., Vol. 32, No. 6, 2008, pp. 916–924.

28. Ayvaz M.T.: *Identification of groundwater parameter structure using harmony search algorithm.* Stud. Comput. Intell., Vol. 191, 2009, pp. 129–140.
29. Ayvaz M.T.: *Solution of groundwater management problems using harmony search algorithm.* Stud. Comput. Intell., Vol. 270, 2010, pp. 111–122.
30. Babbar M., Lakshmikantha A., Goldberg D.: *A modified NSGA-II to solve noisy multiobjective problems.* In: Foster J. *et al.* (eds.): Genetic and Evolutionary Computation Conf. (GECCO'2003), Chicago, Illinois, USA, Lecture Notes in Computer Science, Vol. 2723, 2003, pp. 21–27.
31. Bacon J.: *Concurrent systems.* Addison-Wesley, Wokingham 1993.
32. Balicki J. i in.: *Comcute system utrzymania wielkiej mocy obliczeniowej*. Raport końcowy z projektu rozwojowego OR00010811, Politechnika Gdańska, Gdańsk 2012.
33. Balicki J.: *Algorytmy ewolucyjne oraz algorytmy przeszukiwania tabu do optymalizacji przydziałów modułów programów w rozproszonych systemach komputerowych*. Wyd. AMW, Gdynia 2000.
34. Balicki J.: *Multi-criterion decision making by artificial intelligence techniques*. Proc. on the 8th Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases, February 2009, Cambridge, pp. 319-324.
35. Balicki J., Krawczyk H., Nawarecki E. (Eds.): *Grid and volunteer computing*. Gdańsk University of Technology Press, Gdańsk 2012.
36. Ban V.T.: *A finite algorithm for minimizing a concave function under linear constraints and its applications.* Proc. of IFIP Working Conf. on Recent Advances on System Modelling and Optimization, 1983.
37. Barzegari M., Bathaee S.M.T., Mohsen A.: *Optimal coordination of directional overcurrent relays using harmony search algorithm*. In: Int. Conf. on Environment and Electrical Engineering, 2010, pp. 321–324.
38. Bekda G., Nigdeli S.M.: *Estimating optimum parameters of tuned mass dampers using harmony search.* Eng. Struct., Vol. 33, No. 9, 2011, pp. 2716–2723.
39. Benson H. P.: *An outer approximation algorithm for generating all efficient extreme points in the outcome set of a multiple objective linear programming problem*. Journal of Global Optimization**,** Vol. 13, No. 1, 1998, pp. 1–24.
40. Berman F., Fox G., Hey A.: *Grid computing: making the global infrastructure a reality*. Wiley, New York 2003.
41. Bernaschi M., Castiglione F., Succi S.: *A high performance simulator of the immune system*. Future Generation Computer System, Vol. 15, 2006, pp. 333-342.
42. Blazewicz J., Ecker K. H., Pesch E., Schmidt G., Weglarz J.: *Handbook on scheduling: From theory to applications*. Springer, Heidelberg-Berlin 2007.
43. Błażewicz J.: *Problemy optymalizacji kombinatorycznej - Złożoność obliczeniowa. Algorytmy aproksymacyjne.* PWN, Warszawa 1986.
44. Bo G., Huang M., Ip W.H., Wang X.: *The application of harmony search in fourth-party logistics routing problems.* Stud. Comput. Intell., Vol. 270, 2010, pp. 135–145.
45. Bokhari S. H.: *Partitioning problems in parallel, pipelined, and distributed computing.* IEEE Transactions on Computers, Vol. C-37, No. 1, January 1988, pp. 48-57.
46. Bokhari S. H.: *Assignment problems in parallel and distributed computing.* Kluwer Academic Publishers, Boston 1987.
47. Bonomi F., Kumar A.: *Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler*. IEEE Trans. Comput., Vol. 39, No. 10, 1990, pp. 1232–1250.
48. Brzeziński J., Sobaniec C.: *Systemy rozproszone*. Wyd. Politechniki Poznańskiej, Poznań 2006.
49. Buscemi M.G., Montanari U., Taneja S.: *A game-theoretic analysis of grid job scheduling*. J. Grid Computing, Vol. 10, No. 3, 2012, pp. 501–519.
50. Ceylan H., Ceylan H.: *Harmony search algorithm for transport energy demand modeling.* Stud. Comput. Intell., Vol. 191, 2009, pp. 163–172.
51. Ceylan O., Dag H., Ozdemir A.: *Harmony search method based parallel contingency analysis*. Int. Conf. on Power System Technology, 2010, pp. 1–6.
52. Chandran L.P., Nazeer K.A.A.: *An improved clustering algorithm based on K-means and harmony search optimization*. IEEE Recent Advances in Intelligent Computational Systems, 2011, pp. 447–450.
53. Chatterjee A., Ghoshal S.P., Mukherjee V.: *Solution of combined economic and emission dispatch problems of power systems by an apposition-based harmony search algorithm.* Int. J. Electr. Power Energy Syst., Vol. 39, No. 1, 2011, pp. 9–20.
54. Chaudhary V., Aggarwal J.K.: *A generalized scheme for mapping parallel algorithms*. IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 3, March 1993, pp. 328-346.

55. Chen C., Cherkassky V.: *Task allocation and reallocation for fault tolerance in multicomputer systems*, IEEE Transaction on Aerospace and Electronic Systems, Vol. 30, Issue 4, October 1994, pp. 1094 – 1104.
56. Cheng Y.M.: *Modified harmony methods for slope stability problems: music-inspired harmony search algorithm*. Stud. Comput. Intell., Vol. 191, 2009, pp. 141–162.
57. Cheng P., Yong W.: *A hybrid simplex harmony search algorithm and its application to model reduction of linear systems*. Chinese Control Conf., China, 2010, pp. 5272–5275.
58. Chien A., Calder B., Elbert S., Bhatia K.: *Entropia: architecture and performance of an enterprise desktop grid system.* J. Parallel Distrib. Comput., Vol. 63, 2003, pp. 597–610.
59. Choi S., Chisu W.: *Partitioning and allocation of objects in heterogeneous distributed environments using niched Pareto genetic-algorithm, Proc. of 1998 Asia Pacific Software Engineering Conf. (APSEC 98)*, Taipei, Taiwan, December 1998, pp. 322-329,
60. Chu W. W., Lan L. M. T.: *Task allocation and precedence relations for distributed real-time systems.* IEEE Transactions on Computers, Vol. C-36, No. 6, June 1987, pp. 667-679.
61. Cisty M.: *Application of the harmony search optimization in irrigation*. Stud. Comput. Intell., Vol. 270, 2010, pp. 123–134.
62. Cobos C., Estupiñan D., Perez J.: *GHS+LEM: global-best harmony search using learnable evolution models.* Appl. Math. Comput., Vol. 218, No. 6, 2011, pp. 2558–2578.
63. Cobos C., Andrade J., Constain W., Mendoza M., Leon E.: *Web document clustering based on global-best harmony search, K-means, frequent term sets and Bayesian information criterion*. IEEE Congress on Evolutionary Computation, 2010, pp. 1–8.
64. Coelho L.S., Bernert L.A.: *An improved harmony search algorithm for synchronization of discrete-time chaotic systems*. Chaos Soliton Fract., Vol. 41, 2009, pp. 2526–2532.
65. Coelho L.S., Diego L., Bernert A.: *A harmony search approach using exponential probability distribution applied to fuzzy logic control optimization*. Stud. Comput. Intell., Vol. 270, 2010, pp. 77–88.
66. Coelho L.S., Mariani V.C.: *An improved harmony search algorithm for power economic load dispatch.* Energy Convers. Manage., Vol. 50, No. 10, 2009, pp. 2522–2526.
67. Coelho L.S., Bernert L.A., Mariani V.C.: *Chaotic differential harmony search algorithm applied to power economic dispatch of generators with multiple fuel options*. IEEE Congress on Evolutionary Computation, 2010, pp. 1–5.
68. Coello Coello C.A., Jaimes A.L., Quintero L.V.S.: *Ranking methods in many-objective evolutionary algorithms.* In: Raymond Chiong (ed.), *Nature-inspired algorithms for optimisation*, Springer, New York 2009, pp. 413-434.
69. Coello Coello C.A., Lamont G., Veldhuizen D.: *Evolutionary algorithms for solving multi-objective problems*, Springer, New York 2007 (*the second edition*).
70. Coello Coello C.A., Schuetze O., Laumanns M., Tantar E.: *Computing gap-free Pareto front approximations with stochastic search algorithms.* Evolutionary Computation, Vol. 18, 2010, pp. 65-96.
71. Coello Coello C.A., Mukhopadhyay A., Maulik U., Bandyophadyay S.: *A Survey of multiobjective evolutionary algorithms for data mining: Part I*. IEEE Transactions on Evolutionary Computation, Vol. 18, No. 1, February 2014, pp. 4-19.
72. Corne D.W., Knowles J.D., Oates M.J.: *The Pareto envelope-based selection algorithm for multiobjective optimization.* In: M. Schoenauer, K. Deb, G. Rudolph, X. Yao, 14. E. Lutton, J. J. Merelo, and H.-P. Schwefel (editors), Proc. of the VI Conf. on Parallel Problem Solving from Nature, Paris, September 2000, Lecture Notes in Computer Science, Vol. 1917, 2000, pp. 839–848.
73. Coulouris G., Dollimore J., Kindberg T.: *Distributed systems: Concepts and design*. Addison-Wesley, Harlow 2005.
74. Coutinho F., de Carvalho L.A.V.: *Strategies based on green policies to the grid resource allocation.* Proceedings of the 21st Int. Conf. on Parallel Architectures and Compilation TechniquesNew York, 2012, pp. 487–488.
75. Da Costa G., Gelas J.-P., Georgiou Y., Lefevre L., Orgerie A.-C., Pierson J.-M., Richard O., Sharma K.: *The green-net framework: Energy efficiency in large scale distributed systems.* Proc. of the 2009 IEEE Int. Symposium on Parallel&Distributed Processing, IEEE Computer Society, Washington, 2009, pp. 1–8.
76. Das S., Mukhopadhyay A., Roy A., Abraham A., Panigrahi B.K.: *Exploratory power of the harmony search algorithm: analysis and improvements for global numerical optimization*. IEEE Trans. Systems Man Cybern. Part B: Cybern., Vol. 41, No. 1, 2011, pp. 89–106.
77. Das Sharma K., Chatterjee A., Rakshit A.: *Design of a hybrid stable adaptive fuzzy controller employing lyapunov theory and harmony search algorithm*. IEEE Trans. Contr. Syst. Tech., Vol. 18 2010, pp. 1440–1447.

78. De Jong K. A.: *Evolutionary computation. A unified approach*. A Bradford Book, The MIT Press Cambridge, Massachusetts, London 2006.
79. Deb K., Agrawal S., Pratap A., T. Meyarivan: *A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II*. In: M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, (Eds.): Proc. of the VI Conf. on Parallel Problem Solving from Nature, Paris, September 2000, Lecture Notes in Computer Science, Vol. 1917, 2000, pp. 849–858.
80. Deb K.: *Multi-objective optimization using evolutionary algorithms*, John Wiley & Sons, New York 2001.
81. Degertekin S.O.: *Improved harmony search algorithms for sizing optimization of truss structures.* Comput. Struct., Vol. 92–93, 2012, pp. 229–241.
82. Del Ser J., Matinmikko M., Gil-Lopez S., Mustonen M.: *Centralized and distributed spectrum channel assignment in cognitive wireless networks: a harmony search approach.* Appl. Soft Comput., Vol. 12, No. 2, 2012, pp. 921–930.
83. Del Ser J., Bilbao M.N., Gil-Lopez S., Matinmikko M., Salcedo-Sanz S.: *Iterative power and subcarrier allocation in rate-constrained OFDMA downlink systems based on harmony search heuristics*. Eng. Appl. Artif. Intell., Vol. 24, No. 5, 2011, pp. 748–756.
84. Dhaenens C, Lemesre, J, Talbi E.G.: *K-PPM: A new exact method to solve multi-objective combinatorial optimization problems.* European Journal of Operational Research, Vol. 200, no. 1, 2010, pp. 45-53.
85. Diao R., Shen Q.: *Feature selection with harmony search*. IEEE Trans. Syst. Man Cybern. Part B: Cybern., 2012, pp. 1–15.
86. Diao R.: *Two new approaches to feature selection with harmony search*. In: IEEE Int. Conf. on Fuzzy Systems, Aberystwyth, UK, 2010, pp. 1–7.
87. Donoso Y., Fabregat R.: *Multi-objective optimization in computer networks using metaheuristics*. Auerbach Publications, Boca Raton 2007.
88. Dorronsoro B., Bouvry P., Cañero J., Maciejewski A., Siegel H.: *Multi-objective robust static mapping of independent tasks on grids*. IEEE Congress on Evolutionary Computation (CEC). Part of the World Congress on Computational Intelligence (WCCI), 2010, pp. 3389–3396.
89. Doty K. W., McEntire P. L., O'Reilly J. G.: *Task allocation in a distributed computer system.* Proceedings of the IEEE Infocom, Las Vegas, Nevada, USA, 1982, pp. 33-38.
90. Duan Q., Liao T.W., Yi H.Z.: *A comparative study of different local search application strategies in hybrid metaheuristics.* Appl. Soft Comput., Vol. 13, No. 3, 2013, pp. 1464–1477.
91. Duch W., Korbicz J., Rutkowski L., Tadeusiewicz R.: *Sieci neuronowe.* AOW Exit, Warszawa, 2000.
92. Ehrgott, M, Gandibleux, X.: *A survey and annotated bibliography of multiobjective combinatorial optimization.* OR-Spektrum, Vol. 22, No. 4, 2000, pp. 425-460.
93. Erdal F., Dogan E., Saka M.P.: *Optimum design of cellular beams using harmony search and particle swarm optimizers.* J. Constr. Steel Res., Vol. 67, No. 2, 2011, pp. 237–247.
94. Ezhilarasi G.A., Swarup K.S.: *Network partitioning using harmony search and equivalencing for distributed computing*. J. Parallel Distrib. Comput., Vol. 72, No. 8, 2012, pp. 936–943.
95. Farina M., Amato P.: *Fuzzy optimality and evolutionary multiobjective optimization*. Proc. of the Second Int. Conf. on Evolutionary Multi-Criterion Optimization, Faro, Portugal, April 2003, Lecture Notes in Computer Science. Vol. 2632, pp. 58–72.
96. Fesanghary M., Damangir E., Soleimani I.: *Design optimization of shell and tube heat exchangers using global sensitivity analysis and harmony search algorithm.* Appl. Therm. Eng., Vol. 29, No. 5, 2009, pp. 1026–1031.
97. Fesanghary M., Damangir E., Soleimani I.: *Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems.* Comput. Methods Appl. Mech. Eng., Vol. 197, 2008, pp. 3080–3091.
98. Fesanghary M., Asadib S., Geem Z. W.: *Design of low-emission and energy-efficient residential buildings using a multi-objective optimization algorithm*. Build. Environ., Vol. 49, 2012, pp. 245–250.
99. Fesanghary M.: *An introduction to the hybrid HS-SQP method and its applications: Recent advances in harmony search algorithm*. Stud. Comput. Intell., Vol. 270, 2010, pp. 99–109.
100. Fesanghary M.: *Harmony search applications in mechanical, chemical and electrical engineering.* Stud. Comput. Intell., Vol. 191, 2009, pp. 243–254.
101. Finkelstein A., Gryce C., Lewis-Bowen J.: *Relating requirements and architectures: A study of data-grids*. Journal of Grid Computing, September 2004, Vol. 2, Issue 3, pp 207-222.
102. Fisher N., Anderson J., Baruah S.: *Task partitioning upon memory-constrained multiprocessors*, Proceedings of the 11th IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA'05), Hong Kong, August 2005, pp. 416-421.

103. Foley J.D., Van Dam A.: *Fundamentals of interactive computer graphics*. Addison-Wesley, Massachusetts 1982.
104. Fonseca C.M., Fleming P.J.: *Genetic algorithms for multiobjective optimization: formulation, discussion and generalization*, In: S. Forrest (Ed.), Proc. of the Fifth Int. Conf. on Genetic Algorithms, San Mateo, University of Illinois at Urbana-Champaign, California, June 1993, pp. 416–423.
105. Forsati R., Mahdavi M.: *Web text mining using harmony search*. Stud. Comput. Intell., Vol. 270, 2010, pp. 51–64.
106. Fourie J., Green R., Mills S., *An accurate harmony search based algorithm for the blind deconvolution of binary images.* Int. Conf. on Audio Language and Image Processing, New Zealand, 2010, pp. 1117–1122.
107. Fourie J., Mills S., Green R.: *Visual tracking using harmony search*. Int. Conf. on Image and Vision Computing, New Zealand, 2008, pp. 1–6.
108. Fu F., Zhang C.: *A modified harmony search for multi-mode resource constrained project scheduling problem*. Int. Symposium on Computational Intelligence and Design, China, Vol. 1, 2011, pp. 181–184.
109. Funabiki N., Kitamichi J., Nishikawa S.: *An evolutionary neural network algorithm for max cut problems.* Proc. of the 1997 Int. Conf. on Neural Networks, Vol. 2, Houston, USA, June 1997, pp. 1260-1265.
110. Gandhi T.K., Chakraborty P., Roy G.G., Panigrahi B.K.: *Discrete harmony search based expert model for epileptic seizure detection in electroencephalography*. Expert Syst. Appl., Vol. 39, No. 4, 2012, pp. 4055–4063.
111. Gao J., Wang J., Wang B., Song X.: *A PAPR reduction algorithm based on harmony research for OFDM systems*. Procedia Eng., Vol. 15, 2011, pp. 2665–2669.
112. Gao X.Z., Wang X., Ovaska S.J.: *Harmony search methods for multi-modal and constrained optimization: music-inspired harmony search algorithm.* Stud. Comput. Intell., Vol. 191, 2009, pp. 39–51.
113. Gao K.Z., Li H., Pan Q.K., Li J.Q., Liang J.J.: *Hybrid heuristics based on harmony search to minimize total flow time in no-wait flow shop*. Chinese Control and Decision Conf., China, 2010, pp. 1184–1188.
114. Gao L., Zou D., Ge Y., Jin W.: *Solving pressure vessel design problems by an effective global harmony search algorithm.* Chinese Control and Decision Conf., China, 2010, pp. 4031–4035.
115. Gao X.Z., Jokinen T., Wang X.L., Ovaska S.J., Arkkio A.: *A new harmony search method in optimal wind generator design.* XIX Int. Conf. on Electrical Machines, 2010, pp. 1–6.
116. Garey M.R., Johnson D.S, Stockmayer L.: *Some simplified NP-complete problems.* Theoretical Computer Science, No. 1, 1971, pp. 237-267.
117. Geem Z.W., Tseng C.L., Williams J.C.: *Harmony search algorithms for water and environmental systems.* Stud. Comput. Intell., Vol. 191, 2009, pp. 113–127.
118. Geem Z.W., Tseng C.-L., Park Y.: *Harmony search for generalized orienteering problem: best touring in China.* Lect. Notes Comput. Sci., Vol. 3612, 2005, pp. 741–750.
119. Geem Z.W., Kim J.H., Loganathan G.V.: *A new heuristic optimization algorithm: harmony search*. Simulation, Vol. 76, No. 2, 2001, pp. 60–68.
120. Geem Z.W.: *Discussion on combined heat and power economic dispatch by harmony search algorithm.* Int. J. Electr. Power Energy Syst., Vol. 33, No. 7, 2011, p. 1348-1353.
121. Geem Z.W.: *Effects of initial memory and identical harmony in global optimization using harmony search algorithm*. Appl. Math. Comput., Vol. 22, No. 22, 2012, pp. 11337–11343.
122. Geem Z.W.: *Harmony search optimisation to the pump-included water distribution network design.* Civ. Eng. Environ. Syst., Vol. 26, No. 3, 2009, pp. 211–221.
123. Geem Z.W.: *Multiobjective optimization of time-cost trade-off using harmony search.* J. Constr. Eng. Manage., Vol. 136, No. 6, 2010, pp. 711–716.
124. Geem Z.W.: *Novel derivative of harmony search algorithm for discrete design variables.* Appl. Math. Comput., Vol. 199, 2007, pp. 223–230.
125. Geem Z.W.: *Optimal cost design of water distribution networks using harmony search.* Eng. Optim., Vol. 38, 2006, pp. 259–280.
126. Geem Z., Cho Y.: *Optimal design of water distribution networks using parameter-setting-free harmony search for two major parameters.* Journal of Water Resources Planning and Management, Vol. 137, No. 4, July 2011, pp. 377-380.
127. Geem Z., Sim K.: *Parameter-setting-free harmony search algorithm.* Applied Mathematics and Computation, Vol. 217, No. 8, December 2010, pp. 3881-3889.
128. Geem Z.: *Parameter Estimation of the nonlinear muskingum model using parameter-setting-free harmony search.* Journal of Hydrologic Engineering, Vol. 16, No. 8, August 2011, pp. 684-688.

129. Geem Z.W.: *Optimal design of water distribution networks using harmony search*. Ph.D. Thesis, Korea University 2000.
130. Geem Z.W.: *School bus routing using harmony search*. Genetic and Evolutionary Computation Conf., Washington, 2005, pp. 277–282.
131. Geem Z.W.: *Improved harmony search from ensemble of music players*. Lecture Notes in Artificial Intelligence, Vol. 4251, 2006, pp. 86–93.
132. Geem Z.W.: *Harmony search algorithm for solving sudoku*. Knowledge-Based Intelligent Information and Engineering Systems, Vol. 4692, 2007, pp. 371–378.
133. Geem Z.W.: *Optimal scheduling of multiple dam system using harmony search algorithm.* Lecture Notes in Computer Science, Vol. 4507, 2007, pp. 316–323.
134. Geem Z.W., Choi J.Y.: *Music composition using harmony search algorithm*. Lecture Notes in Computer Science, Vol. 4448, 2007, pp. 593–600.
135. Geem Z.W., Kim J.H., Loganathan G.V.: *A new heuristic optimization algorithm: harmony search.* Simulation. Vol. 76, No. 2, 2001, pp. 60–68.
136. Geem Z.W., Fourie J., Green R.: *Generalised adaptive harmony search: A comparative analysis of modern harmony search.* Journal of Applied Mathematics. Vol. 2013, 2013, pp. 60–72.
137. Geem Z.W., Lee K.S., Park Y.: *Application of harmony search to vehicle routing*. American Journal of Applied Sciences, Vol. 2, No. 12, 2005, pp. 1552–1557.
138. Geem Z.W., Park Y.: *Harmony search for layout of rectilinear branched networks*. WSEAS Transactions on Systems, Vol. 5, No. 6, 2006, pp. 1349–1354.
139. Geem Z.W., Williams J.C.: *Ecological optimization using harmony search*. Proc. of American Conf. on Applied Mathematics, Cambridge, 2008.
140. Geem Z.W.: *Multiobjective optimization of time-cost trade-off using harmony search*. Journal of Construction Engineering And Management-ASCE, Vol. 136, Issue: 6, 2010, pp. 711-716.
141. Geem Z.W.: *Optimal cost design of water distribution networks using harmony search.* Engineering Optimization, Vol. 38, No. 3, 2006, pp. 259–280.
142. Geem Z.W.: *Population variance harmony search algorithm to solve optimal power flow with non-smooth cost function*. Recent Advances in Harmony Search Algorithm, 2010, pp. 65–75.
143. Gil-Lopez S., Del Ser J., Salcedo-Sanz S., Perez-Bellido A.M., Cabero J.M., Portilla-Figueras J.A.: *A hybrid harmony search algorithm for the spread spectrum radar polyphase codes design problem*. Expert Syst. Appl., Vol. 39, No. 12, 2012, pp. 11089–11093.
144. Gil-Lopez S., Landa-Torres I., Del Ser J., Salcedo-Sanz S., Manjarres D., Portilla-Figueras J.A.: *A novel grouping heuristic algorithm for the switch location problem based on a hybrid dual harmony search technique*. Int. Work Conf. on Artificial Neural Networks, Malaga, Spain, 2011, pp. 353–359.
145. Goczyła K., Ciarkowski J.: *Wydobywanie wiedzy i klasyfikacja danych w systemie typu Web Forming.* W: Electronic commerce: teoria i zastosowanie, *Wyd. Politechniki Gdańskiej, Gdańsk* 2005, ss. 69-79.
146. Goldberg D. E.: *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.
147. Gray L., Kumar A., Li H.: *Workload characterization of the SPECpower_ssj2008 Benchmark.* In: Kounev, S., Gorton, I., Sachs, K. (eds.) Performance Evaluation: Metrics, Models and Benchmarks. Lecture Notes in Computer Science, Vol. 5119, Springer Berlin/Heidelberg 2008, pp. 262–282.
148. Habib S., Rahmatia A., Hajipoura V., Niakib S.T.A.: *A soft-computing Pareto-based meta-heuristic algorithm for a multi-objective multi-server facility location problem*. Appl. Soft Comput., Vol. 13, No. 4, 2013, pp. 1728–1740.
149. Han Y.Y., Pan Q.K., Liang J.J., Li J.: *A hybrid discrete harmony search algorithm for blocking flow shop scheduling.* IEEE Int.l Conf. on Bio-Inspired Computing: Theories and Applications, 2010, pp. 435–438.
150. Harrou F., Zeblah A.: *An efficient harmony search optimization for maintenance planning to the telecommunication systems*. In: Nashat Mansour (Ed.), Search Algorithms and Applications, 2011.
151. He L., Jarvis S.A., Spooner D.P., Jiang H., Dillenberger D.N., Nudd G.R.: *Allocating non-real-time and soft real-time jobs in multiclusters*. IEEE Trans. Parallel Distrib. Syst., Vol. 17, No. 2, 2006, pp. 99–112.
152. Hermenier F., Loriant N., Menaud J.-M.: *Power management in grid computing with Xen.* In: Proc. of the 2006 Int. Conf. on Frontiers of High Performance Computing and Networking, ISPA'06, Springer, Berlin, Heidelberg, 2006, pp. 407–416.
153. Hirales-Carbajal A., Tchernykh A., Yahyapour R., González-García J.L., Röblitz T., Ramírez-Alcaraz J.M.: *Multiple workflow scheduling strategies with user run time estimates on a grid*. J. Grid Computing, Vol. 10, No. 2, 2012, pp. 325–346.

154. Hoang D.C., Yadav P., Kumar R., Panda S.K.: *A robust harmony search algorithm based clustering protocol for wireless sensor networks*. IEEE Int. Conf. on Communications Workshops, Singapore, 2010, pp. 1–5.
155. Horn J., Nafpliotis N.: *Multiobjective optimization using the niched Pareto genetic algorithm*, Technical Report IlliGAl Report 93005, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1993.
156. Hou C. J., Shin K. G.: *Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems.* IEEE Transactions on Computers, Vol. 45, No. 12, December 1997, pp. 1338-1356.
157. Hsu C., Chung Y., Yang D., Dow C.: *A generalized processor mapping technique for array redistribution.* IEEE Transactions on Parallel and Distributed Systems, Vol. 12, No. 7, July 2001, pp. 743-757.
158. Hua J., Yun B., Zheng L., Yanxiu L.: *A hybrid algorithm of harmony search and simulated annealing for multiprocessor task scheduling*. Int. Conf. on Systems and Informatics, 2012, pp. 718–720.
159. Huang Y., Tripathi S.K.*: Resource allocation for primary-site fault-tolerant systems.* IEEE Transactions on Software Engineering, Vol. 19, No. 2, February 1993, pp. 108-119.
160. Huang M., Bo G., Wang X., Ip W.H.: *The optimization of routing in fourth-party logistics with soft time windows using harmony sear*ch. Sixth Int. Conf. on Natural Computation, Vol. 8, 2010, pp. 4344–4348.
161. Ida M.: *Efficient solution generation for multiple objective linear programming based on extreme ray generation method*. European Journal of Operational Research, Vol. 160, No. 1, 2005, pp. 242-251.
162. Iqbal M.A.: *Approximate algorithms for partitioning and assignment problems*. ICASE Report 86-40 NASA Contractor Report 178130, June 1986.
163. Jaberipour M., Khorram E.: *Solving the sum-of-ratios problems by a harmony search algorithm.* J. Comput. Appl. Math., Vol. 234, No. 3, 2010, pp. 733–742.
164. Jafarpour N., Khayyambashi M.R.: *QoS-aware selection of web service composition based on harmony search algorithm*. The 12th Int. Conf. on Advanced Communication Technology, Vol. 2, 2010, pp. 1345–1350.
165. Jaimes A.L., Quintero L.V.S., Coello Coello C.A.: *Ranking methods in many-objective evolutionary algorithms*. In: R. Chiong (ed.), *Nature-Inspired Algorithms for Optimisation*, Springer, New York 2009, pp. 223-231.
166. Javaheri H., Goldoost-Soloot R.: *Locating and sizing of series facts devices using line outage sensitivity factors and harmony search algorithm*. The 2nd Int. Conf. on Advances in Energy Engineering, Vol. 14, 2012, pp. 1445–1450.
167. Jennings N.R., Wooldridge M.: *Applications of intelligent agents.* In: Jennings N.R., Wooldridge M.: *Intelligent agents*, Springer-Verlag, New York, 1998, pp. 3-28.
168. Jia W., Zhou W.: *Distributed network systems – from concepts to implementation*. Springer Science, New York 2005.
169. Jing C., Guang-Liang L., Ran L.: *Discrete harmony search algorithm for identical parallel machine scheduling problem.* Chinese Control and Decision Conf., 2011, pp. 5457–5461.
170. Jozefowicz N., Laporte G., Semet F.: *A generic branch-and-cut algorithm for multiobjective optimization problems: Application to the multilabel traveling salesman problem*. INFORMS Journal on Computing, Vol. 24, No. 4, 2012, pp. 554-564.
171. Kafil M., Ahmad I.: *Optimal task assignment in heterogeneous distributed computing systems.* IEEE Concurrency, Vol. 6, No. 3, 1998, pp. 42-51.
172. Kaizhou G., Quanke P., Junqing L., Yongzheng H.: *A novel grouping harmony search algorithm for the no-wait flow shop scheduling problems with total flow time criteria*. In: Int. Symposium on Computer Communication Control and Automation, Vol. 1, 2010, pp. 77–80.
173. Kalns E.T., Ni L.M.: *Processor mapping techniques toward efficient data redistribution*. IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No. 12, December 1995, pp. 1234-1247.
174. Kameda H., Li J., Kim C., Zhang Y.: *Optimal load balancing in distributed computer systems*. Springer, London 1997.
175. Kartik S., Murthy C.S.R.: *Task allocation algorithms for maximizing reliability of distributed computing systems.* IEEE Transactions on Computers, Vol. 46, No. 6, June 1997, pp. 719-724.
176. Kattan A., Abdullah R.: *An enhanced parallel and distributed implementation of the harmony search based supervised training of artificial neural networks*. Int. Conf. on Computational Intelligence, Communication Systems and Networks, 2011, pp. 275–280.

177. Kattan A., Abdullah R., Salam R.A.: *Harmony search based supervised training of artificial neural networks.* Int. Conf. on Intelligent Systems, Modelling and Simulation, Malaysia, 2010, pp. 105–110.
178. Kaveh A., Shakouri A.: *Cost optimization of a composite floor system using an improved harmony search algorithm.* J. Constr. Steel Res., Vol. 66, No. 5, 2010, pp. 664–669.
179. Kaveh A., Ahangaran M.: *Discrete cost optimization of composite floor system using social harmony search model.* Appl. Soft Comput., Vol. 12, No. 1, 2012, pp. 372–381.
180. Kaveh A., Talatahari S.: *Particle swarm optimizer, ant colony strategy and harmony search scheme hybridized for optimization of truss structures.* Comput. Struct., Vol. 87, 2009, pp. 267–283.
181. Kayhan A.H., Korkmaz K.A., Irfanoglu A.: *Selecting and scaling real ground motion records using harmony search algorithm.* Soil Dyn. Earthq. Eng., Vol. 31, No. 7, 2011, pp. 941–953.
182. Kermani E.M., Salehinejad H., Talebi S.: *PAPR reduction of OFDM signals using harmony search algorithm.* Int. Conf. on Telecommunications, 2011, pp. 90–94.
183. Kertész A., Kacsuk P.: *Gmbs: A new middleware service for making grids interoperable.* Future Gener. Comput. Syst., Vol. 26, No. 4, 2010, pp. 542–553.
184. Khazali A.H., Parizad A., Kalantar M.: *Optimal reactive/voltage control by an improved harmony search algorithm.* In: Canadian Conf. on Electrical and Computer Engineering, 2010, pp. 1–6.
185. Khorram E., Jaberipour M.: *Harmony search algorithm for solving combined heat and power economic dispatch problems*. Energ. Convers. Manage., Vol. 52, No. 2, 2011, pp. 1550–1554.
186. Kim J.H. *et al.*: *Optimal planning model for rehabilitation of water networks*. In: Cubillo F. (Ed): Proc. on the 2nd Int. Conf. on Efficient Use and Management of Water for Urban Supply, Tenerife, Spain, April 02-04, 2003, Vol. 4, Issue: 3, pp. 133-147.
187. Kim S.H., Yoo W.S., Oh K.J., Hwang I.S., Oh J.E.: *Transient analysis and leakage detection algorithm using GA and HS Algorithm for a pipeline system.* Journal of Mechanical Science and Technology, Vol. 20, No. 3, 2006, pp. 426–434.
188. Kirlik G., Sayın S.: *A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems*. European Journal of Operational Research, Vol. 232, No. 3, 2014, pp. 479-488.
189. Klein D., Hannan E.: *An algorithm for the multiple objective integer linear programming problem*. European Journal of Operational Research, Vol. 9, No. 4, 1982, pp. 378-385.
190. Ko K.E., Sim K.B.: *An EEG signals classification system using optimized adaptive neuro-fuzzy inference model based on harmony search algorithm*. Int. Conf. on Control, Automation and Systems, Korea, 2011, pp. 1457–1461.
191. Konakia H.R., Tobagi F.A.: *On distributed computations with limited resources.* IEEE Transactions on Computers, Vol. 48, No. 6, June 1997, pp. 719-724.
192. Kougias I., Theodossiou N.: *Multiobjective pump scheduling optimization using harmony search algorithm (HSA) and polyphonic HSA.* Water Resources Management, Vol. 27, No. 5, March 2013, pp. 1249-1261.
193. Koza J.R. *et al.*: *Genetic programming IV. Routine human-competitive machine intelligence*. Kluwer Academic Publishers, New York 2003.
194. Krawczyk H., Rek A.: *Methodology for developing Web-based applications from reusable components using open-source tools.* Proc. of the 2nd Int. Conf. on Information Technology, ICIT 2010, June 2010, Gdansk, pp. 117-120.
195. Krichen S., Masri H., Guitouni A.: *Adjacency based method for generating maximal efficient faces in multiobjective linear programming.* Applied Mathematical Modelling, Vol. 36, No. 12, 2012, pp. 6301-6311.
196. Krishnaveni V., Arumugam G.: *A novel enhanced bio-inspired harmony search algorithm for clustering.* Int. Conf. on Recent Advances in Computing and Software Systems, 2012, pp. 7–12.
197. Kudikala S., Sabat S.L., Udgata S.K.: *Performance study of harmony search algorithm for analog circuit sizing.* Int. Symposium on Electronic System Design, 2011, pp. 12–17.
198. Kuhn H. W.: *The Hungarian method for the assignment problem*, Naval Research Logistic Quarterly, Vol. 2, 1955, pp. 83-97.
199. Kulluk S., Ozbakir L., Baykasoglu A.: *Training neural networks with harmony search algorithms for classification problems.* Eng. Appl. Artif. Intell., Vol. 25, No. 1, 2012, pp. 11–19.
200. Kulluk S., Ozbakir L., Baykasoglu A.: *Self-adaptive global best harmony search algorithm for training neural networks*. In: World Conf. on Information Technology, Vol. 3, 2011, pp. 282–286.
201. Kurowski K., Nabrzyski J., Oleksiak A., Weglarz J.: *A multicriteria approach to two-level hierarchy scheduling in grids.* J. Scheduling, Vol. 11, No. 5, 2008, pp. 371–379.
202. Kurowski K., Oleksiak A., Witkowski M., Nabrzyski J.: *Distributed power management and control system for sustainable computing environments*. Int. Green Computing Conf., 2010, pp. 365–372.

203. Lammie M., Brenner P., Thain D.: *Scheduling grid workloads on multicore clusters to minimize energy and maximize performance*. GRID, 2009, pp. 145–152.
204. Lamont G. B.: *Multi-objective evolutionary algorithms: what, why and where. A tutorial,* Proceedings of the Third Int. Conf. on Evolutionary Multi-Criterion Optimization, Guanajuato, Mexico, March 2005, pp. 23-35.
205. Landa-Torres I. *et al*.: *A multi-objective grouping harmony search algorithm for the optimal distribution of 24-hour medical emergency units.* Expert Syst. Appl.,Vol. 40, No. 6, 2012, pp. 2343–2349.
206. Landa-Torres I. *et al*.: *Evaluating the Initzation success of companies using a hybrid grouping harmony search-extreme learning machine approach.* IEEE J. Sel. Top. Signal Process., Vol. 6, No. 4, 2012, pp. 388–398.
207. Landa-Torres I. *et al*.: *Efficient citywide planning of openwifi access networks using novel grouping harmony search heuristics*. Eng. Appl. Artif. Intell., Vol. 26, No. 3, 2012, pp. 1124–1130.
208. Landa-Torres I. *et al*.: *A novel grouping harmony search algorithm for the multiple-type access node location problem.* Expert Syst. Appl., Vol. 39, No. 5, 2012, pp. 5262–5270.
209. Laumanns M., Thiele L., Zitzler E.: *An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method*. European Journal of Operational Research, Vol. 169, No. 3, 2006, pp. 932-942.
210. Lee K.S., Geem Z.W., Lee S.H.: *The harmony search heuristic for discrete structural optimization*. Eng. Optim., Vol. 37, 2005, pp. 663–684.
211. Lee K.S., Geem Z.W.: *A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice*. Comput. Methods Appl. Mech. Eng., Vol. 194, No. 36, 2005, pp. 3902–3933.
212. Lee K.S., Geem Z.W.: *A new structural optimization method based on the harmony search algorithm*. Comput. Struct., Vol. 82, No. 9–10, 2004, pp. 781–798.
213. Lee K.S. *et al*.: *The Harmony Search Heuristic Algorithm for Discrete Structural Optimization.* Engineering Optimization. Vol. 37, No. 7, 2005, pp. 663–684.
214. Lee K.S., Geem Z.W.: *A new structural optimization method based on the harmony search algorithm.* Computers and Structures, Vol. 82, No. 9-10, 2004, pp. 781–798.
215. Legrand A., Renard R., Robert Y., Vivien V.: *Mapping and load-balancing iterative computations.* IEEE Transactions on Parallel and Distributed Systems, Vol. 15, No. 6, June 2004, pp. 546-558.
216. Lemesre J., Dhaenens C., Talbi E.G.: *Parallel partitioning method (PPM): A new exact method to solve bi-objective problems.* Computers & operations research, Vol. 34, No. 8, 2007, pp. 2450-2462.
217. Li L., Chi S.C., Lin G.: *Genetic algorithm incorporated with harmony procedure and its application to searching of non-circular critical slip surface in soil slopes*. J. Hydraul. Eng., Vol. 36, 2005, pp. 910–918.
218. Li H.Q., Li L.: *A novel hybrid particle swarm optimization algorithm combined with harmony search for high dimensional optimization problems*. Int. Conf. on Intelligent Pervasive Computing Korea, 2007.
219. Li J. *et al*.: *An effective discrete harmony search for solving bi-criteria FJSP*. Chinese Control and Decision Conf., China, 2011, pp. 3625–3629.
220. Li K.: *Minimizing mean response time in heterogeneous multiple computer systems with a central stochastic job dispatcher*. Int. J. Comput. Appl., Vol. 20, No. 1, 1996, pp. 32–39.
221. Li K.: *Minimizing the probability of load imbalance in heterogeneous distributed computer systems*. Math. Comput. Model. Vol. 36, No. 9–10, 2002, pp. 1075–1084.
222. Li K.: *Optimal load distribution in nondedicated heterogeneous cluster and Grid computing environments*. J. Syst. Architect., Vol. 54, No. 1–2, 2008, pp. 111–123.
223. Li K.: *Optimizing average job response time via decentralized probabilistic job dispatching in heterogeneous multiple computer systems*. Comput. J., Vol. 41, No. 4, 1998, pp. 223–230.
224. Li Y. *et al*.: *A spectral clustering-based adaptive hybrid multi-objective harmony search algorithm for community detection*. IEEE Congress on Evolutionary Computation, Brisbane, 2012, pp. 1–8.
225. Lin Z., DaoLi Z., ZhongPing S.: *Finding a minimal efficient solution of a convex multiobjective program*. Journal of Optimization Theory and Applications, Vol. 118, No. 3, 2003, pp. 587-600.
226. Liu S.F, Soffa M.L.: *Parallel task assignment by graph partitioning*. Lecture Notes in Computer Science, Vol. 605, 1992, pp. 965-966.
227. Lo V.M.: *Heuristic algorithms for task assignment in distributed systems*. IEEE Transactions on Computers, Vol. C-37, No. 11, November 1988, pp. 1384-1397.
228. Lobo F.G., Lima C.F., Michalewicz Z.: *Parameter setting in evolutionary algorithms*, Studies in Computational Intelligence, Vol. 54, 2007, pp. 1-18.

229. Lokman B., Köksalan M.: *Finding all nondominated points of multi-objective integer programs*. Journal of Global Optimization, Vol. 57, No. 2, 2013, pp. 347-365.
230. Ludwig S.A., Moallem A.: *Swarm intelligence approaches for grid load balancing.* J. Grid Computing, Vol. 9, No. 3, 2011, pp. 279–301.
231. Luo P., Lü K., Shi Z.: *A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems*. J. Parallel Distrib. Comput. Vol. 67, No. 6, 2007, pp. 695–714.
232. Luo Q. *et al.*: *Optimal design of groundwater remediation systems using a multi-objective fast harmony search algorithm.* Hydrogeology Journal, Vol. 20, No. 8, December 2012, pp. 1497-1510.
233. Luo Q.K., Wu J.F., Yang Y.: *An improved fast harmony search algorithm for identification of hydrogeological parameters*. Int. Conf. on Natural Computation, Vol. 4, 2011, pp. 1960–1963.
234. Ma J., Liu J., Ren Z.: *Parameter estimation of poisson mixture with automated model selection through BYY harmony learning*. Pattern Recognition, Vol. 42, No. 11, 2009, pp. 2659–2670.
235. Mahdavi M., Fesanghary M., Damangir E.: *An improved harmony search algorithm for solving optimization problems.* Appl. Math. Comput., Vol. 188, No. 2, 2007, pp. 1567–1579.
236. Majid J., Esmaile K.: *Two improved harmony search algorithms for solving engineering optimization problems.* Commun. Nonlinear Sci. Numer. Simul., Vol. 15, No. 11, 2010, pp. 3316–3331.
237. Mandava R. *et al.*: *Osteosarcoma segmentation in MRI using dynamic harmony search based clustering*. Int. Conf. of Soft Computing and Pattern Recognition, Malaysia, 2011, pp. 423–429.
238. Manjarres D. *et al.*: *On the design of a novel two-objective harmony search approach for distance- and connectivity-based localization in wireless sensor networ*ks. Eng. Appl. Artif. Intell., Vol. 26, No. 2, 2013, pp. 669–676.
239. Manjarres D. *et al.*: *A novel heuristic approach for distance- and connectivity-based multihop node localization in wireless sensor networks*. Appl. Soft Comput., 2012, pp. 1–12.
240. Manjarres D. *et al.*: *A survey on applications of the harmony search algorithm.* Engineering Applications of Artificial Intelligence, Vol. 26, No. 8, September 2013, pp. 1818-1831.
241. Manjarres D. *et al.*: *A heuristically-driven multi-criteria tool for the design of efficient open WiFi access networks*. IEEE Int. Workshop on Computer-Aided Modeling Analysis and Design of Communication Links and Networks (CAMAD), 2012, pp. 218-231.
242. Marchand A., Silly-Chetto M.: *Dynamic scheduling of skippable periodic tasks: issues and proposals*, Journal of Software, Vol. 2, No. 5, November 2007, pp. 44-51.
243. Mashinchi M.H., Orgun M.A., Mashinchi M., Pedrycz W.: *A tabu-harmony search-based approach to fuzzy linear regression*. IEEE Trans. Fuzzy Syst., Vol. 19, No. 3, 2011, pp. 432–448.
244. Masiejczyk J.: *Wielokryterialne algorytmy przeszukiwania tabu do optymalizacji trajektorii autonomicznego pojazdu podwodnego*. Rozprawa doktorska, Wydział Elektroniki, Telekomunikacji i Informatyki, Politechnika Gdańska, Gdańsk 2013.
245. Mavrotas G., Diakoulaki D.: *A branch and bound algorithm for mixed zero-one multiple objective linear programming.* European Journal of Operational Research, Vol. 107, No. 3, 1998, pp. 530-541.
246. Mavrotas G., Florios K.: *An improved version of the augmented ε-constraint method (AUGMECON2) for finding the exact Pareto set in multi-objective integer programming problems*. Applied Mathematics and Computation, Vol. 219, No. 18, 2013, pp. 9652-9669.
247. Mavrotas G.: *Effective implementation of the ε-constraint method in multi-objective mathematical programming problems*. Applied Mathematics and Computation, Vol. 213, No. 2, 2009, pp. 455-465.
248. Mehdizadeh A. *et al.*: *An efficient 60 GHz resonator using harmony search*. IEEE Recent Advances in Intelligent Computational Systems, 2011, pp. 369–372.
249. Melih Ö., Azizoğlu M.: *Multi-objective integer programming: a general approach for generating all non-dominated solutions.* European Journal of Operational Research, Vol. 199, No. 1, 2009, pp. 25-35.
250. Michalewicz Z.: *Algorytmy genetyczne + struktury danych = programy ewolucyjne.* WNT, Warszawa 1996.
251. Mohsen A.M., Khader A.T., Ramachandram D.: *An optimization algorithm based on harmony search for RNA secondary structure prediction.* Stud. Comput. Intell., Vol. 270, 2010, pp. 163–174.
252. Montusiewicz J.: *Ewolucyjna analiza wielokryterialna w zagadnieniach technicznych*. Wyd. Instytut Podstawowych Problemów Techniki PAN, Warszawa 2004.
253. Mun S., Geem Z.W.: *Determination of individual sound power levels of noise sources using a harmony search algorithm*. Int. J. Ind. Ergon., Vol. 39, No. 2, 2009, pp. 366–370.

254. Mun S., Geem Z.W.: *Determination of viscoelastic and damage properties of hot mix asphalt concrete using a harmony search algorithm*. Mech. Mater., Vol. 41, No. 3, 2009, pp. 339–353.
255. Murthy L., Seo P.K.: *A dual-descent procedure for the file allocation and join site selection problem on a telecommunications network*. An Int. Journal Networks, Vol. 33, No. 2, 1999, pp. 109-123.
256. Navi S.P. *et al*.: *Using harmony search for solving a typical bioinformatics problem*. Int. Conf. on Informatics and Computational Intelligence, 2011, pp. 18–20.
257. Nesmachnow S. *et al*.: *Energy-aware scheduling on multicore heterogeneous grid computing systems.* Journal of Grid Computing, May 2013, pp. 223-235.
258. Nezhad S.E., Kamali H.J., Moghaddam M.E.: *Solving K-coverage problem in wireless sensor networks using improved harmony search*. Int. Conf. on Broadband, Wireless Computing, Communication and Applications, 2010, pp. 49–55.
259. Omran M.G.H., Mahdavi M.: *Global-best harmony search*. Appl. Math. Comput., Vol. 198, No 2, 2008, pp. 643–656.
260. Paluszak J. *et al*.: *Superkomputery do wspomagania procesów gospodarczych ze szczególnym uwzględnieniem sektora bankowego.* Współczesna Gospodarka, Vol. 5, Issue 4, 2014, pp. 1-16.
261. Paluszak J. *et al*.: *Metody sztucznej inteligencji do wspomagania bankowych systemów informatycznych.* W: K. Kreft, D. Wach, J. Winiarski (red.): Systemy informatyczne w gospodarce. *Wyd. Uniwersytetu Gdańskiego,* Gdańsk 2013, ss. 125-138.
262. Paluszak J., Balicki J., Zacniewski A.: *Wybrane aspekty architektur rozproszonych systemów komputerowych*. W: J. Balicki, J. Kuchta (red.): *Obliczenia rozproszone w systemach komputerowych o architekturze klasy grid.* Wyd. Politechniki Gdańskiej, Gdańsk 2012, ss. 7-23.
263. Paluszak J. *et al*.: *Rozpraszanie obliczeń za pomocą serwerów dystrybucyjnych.* W: J. Balicki, J. Kuchta (red.): *Obliczenia rozproszone w systemach komputerowych o architekturze klasy grid.* Wyd. Politechniki Gdańskiej, Gdańsk 2012, ss. 113-126.
264. Paluszak J. *et al*.: *Optymalizacja równoważenia obciążeń w systemach klasy grid.* W: J. Balicki, J. Kuchta (red.): *Obliczenia rozproszone w systemach komputerowych o architekturze klasy grid.* Wyd. Politechniki Gdańskiej, Gdańsk 2012, ss. 145-160.
265. Paluszak J. *et al*.: *Empiryczna weryfikacja hipotezy Collatza za pomocą obliczeń gridowych* ICT YOUNG 2012, II Konferencja Studentów i Doktorantów Elektroniki, Telekomunikacji, Informatyki, Automatyki i Robotyki, Gdańsk 2012.
266. Paluszak J. *et al*.: *Genetic programming for workload balancing in the Comcute grid system.* In: J. Balicki, H. Krawczyk, E. Nawarecki (Eds.): *Grid and Volunteer Computing*. Gdańsk University of Technology Press, Gdańsk 2012, pp. 97-115.
267. Paluszak J. *et al*.: *Genetic programming with negative selection for volunteer computing system optimization*. Proceedings of the 6th Int. Conf. on *Human System Interaction*, Sopot, Poland, June 06-08, 2013.
268. Paluszak J. *et al*.: *Mersenne number finding and Collatz hypothesis verification in the Comcute grid system.* In: J. Balicki, H. Krawczyk, E. Nawarecki (Eds.): *Grid and Volunteer Computing*. Gdańsk University of Technology Press, Gdańsk 2012, pp. 148-162.
269. Paluszak J. *et al*.: *Techniki audytowania zabezpieczeń sieci bezprzewodowej z wykorzystaniem systemów klasy grid.* W: J. Balicki, J. Kuchta (red.): *Obliczenia rozproszone w systemach komputerowych o architekturze klasy grid.* Wyd. Politechniki Gdańskiej, Gdańsk 2012, ss. 161-174.
270. Paluszak J., Balicki J., Korłub W.: *Efektywne zrównoleglanie obliczeń w systemie klasy grid na przykładzie hipotezy Collatza*. W: J. Balicki, J. Kuchta (red.): *Obliczenia rozproszone w systemach komputerowych o architekturze klasy grid.* Wyd. Politechniki Gdańskiej, Gdańsk 2012, ss. 243-254.
271. Paluszak J., Balicki J.: *Wybrane rozwiązania występujące w architekturach systemów rozproszonych klasy grid.* W: J. Balicki, J. Kuchta (red.): *Obliczenia rozproszone w systemach komputerowych o architekturze klasy grid.* Wyd. Politechniki Gdańskiej, Gdańsk 2012, ss. 24-38.
272. Paluszak J.: *Bezpieczeństwo transferu zestrukturalizowanych plików XML w sieci grid w oparciu o usługi Web Service poprzez protokół SOAP*. W: J. Balicki, J. Kuchta (red.): Obliczenia rozproszone w systemach komputerowych o architekturze klasy grid, Wyd. Politechniki Gdańskiej WETI, Gdańsk 2012, ss. 68-79.
273. Paluszak J.: *Inteligentne wspomaganie podejmowania decyzji z wykorzystaniem metod sztucznej inteligencji w środowisku obliczeniowym typu grid*. W: J. Balicki, J. Kuchta (red.): Obliczenia rozproszone w systemach komputerowych o architekturze klasy grid, Wyd. Politechniki Gdańskiej WETI, Gdańsk 2012, ss. 231-242.
274. Pan Q.K., Wang L., Gao L.: *A chaotic harmony search algorithm for the flow shop scheduling problem with limited buffers*. Appl. Soft Comput., Vol. 11, No. 8, 2011, pp. 5270–5280.

275. Pan Q.K., Suganthan P.N., Liang J.J., Tasgetiren M.F.: *A local-best harmony search algorithm with dynamic sub-harmony memories for lot-streaming flow shop scheduling problem.* Expert Syst. Appl., Vol. 38, No. 4, 2011, pp. 3252–3259.
276. Pan Q.K. *et al*.: *A self-adaptive global best harmony search algorithm for continuous optimization problems*. Appl. Math. Comput., Vol. 216, No. 3, 2010, pp. 830–848.
277. Panchal A.: *Harmony search in therapeutic medical physics*. Stud. Comput. Intell., Vol. 191, 2009, pp. 189–203.
278. Panchal A.: *Harmony search optimization for HDR prostate brachytherapy*. Rosalind Franklin University of Medicine and Science, 2008.
279. Pandi V.R., Panigrahi B.K.: *Dynamic economic load dispatch using hybrid swarm intelligence based harmony search algorithm.* Expert Syst. Appl., Vol. 38, No. 7, 2011, pp. 8509–8514.
280. Papadimitriou C.D., Steiglitz K.: *Combinatorial optimization: algorithms and complexity.* Prentice Hall, Englewood Cliffs, New York 1982.
281. Parikshit Y. *et al*.: *An intelligent tuned harmony search algorithm for optimization.* Inf. Sci., Vol. 196, 2012, pp. 47–72.
282. Parizad A., Khazali A.H., Kalantar M.: *Sitting and sizing of distributed generation through harmony search algorithm for improving voltage profile and reduction of THD and losses*. Canadian Conf. on Electrical and Computer Engineering, 2010, pp. 1–7.
283. Peiying H., Dandan W., Xiaoping L.: *An improved harmony search algorithm for blocking job shop to minimize makespan.* Int. Conf. on Computer Supported Cooperative Work in Design, 2012, pp. 763–768.
284. Price C. C., Pooch U.W.: *Search techniques for a nonlinear multiprocessor scheduling problem.* Naval Research Logistics Quarterly, Vol. 29, No. 2, June 1982, pp. 213-233.
285. Prodan R., Wieczorek M.: *Negotiation-based scheduling of scientific grid workflows through advance reservations*. J. Grid Comput., Vol. 8, No. 4, 2010, pp. 493–510.
286. Przybylski A., Gandibleux X. Ehrgott M.: *A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives*. Discrete Optimization, Vol. 7, No. 3, 2010, pp. 149-165.
287. Przybylski A., Gandibleux X., Ehrgott M.: *A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme*. INFORMS Journal on Computing, Vol. 22, No. 3, 2010, pp. 371-386.
288. Pugliese A., Talia D., Yahyapour R.: *Modeling and supporting grid scheduling*. J. Grid Computing Vol. 6, 2008, pp. 195–213.
289. Pyne S. J., Andrews P. L., Laven R. D.: *Introduction to wildland fire*. John Wiley and Sons, New York 1996.
290. Raidl G.: *An improved genetic algorithm for the multiconstrained 0-1 Knapsack Problem.* Proc. of the 1998 IEEE Int. Conf. on Evolutionary Computation, Anchorage, Alaska, USA, May 1998, pp. I-207-I-211.
291. Ramos C.C.O. *et al*.: *A novel algorithm for feature selection using harmony search and its application for non-technical losses detection.* Comput. Electr. Eng., Vol. 37, No. 6, 2011, pp. 886–894.
292. Rao G.S., Stone H.S., Hu T.C.: *Assignment of tasks in a distributed processor system with limited memory.* IEEE Transactions on Computers, Vol. C-28, No. 4, April 1979, pp. 291-299.
293. Rasiowa H.: *Wstęp do matematyki współczesnej*. PWN, Warszawa 2007 (wyd. XIV).
294. Ren W.J. *et al*.: *Harmony search algorithms for bi-criteria no-idle permutation flow shop scheduling problem*. In: Chinese Control and Decision Conf., China, 2011, pp. 2513-2517.
295. Ricart J., Hüttemann G., Lima J., Barán B.: *Multiobjective harmony search algorithm proposals*. Electronic Notes in Theoretical Computer Science, Vol. 281, December 2011, pp. 51-67.
296. Riedel M., Laure E., *et al*.: *Interoperation of worldwide production e-science infrastructures.* Journal of Concurrency and Computation: Practice and Experience, Vol. 21, 2009, pp. 961–990.
297. Riedel M. *et al*.: *Classification of different approaches for e-science applications in next generation computing infrastructures.* Proc. of the 4th IEEE Conf. on e-Science, Indianapolis, USA, 2008, pp. 198–205.
298. Rings T., Neukirchen H., Grabowski J.: *Testing grid application workflows using TTCN-3*. Int. Conf. on Software Testing Verification and Validation (ICST), IEEE Computer Society, Piscataway, 2008, pp. 210–219.
299. Rommel C.G.: *The probability of load balancing success in a homogeneous network.* IEEE Trans. Softw. Eng., Vol. 17, No. 9, 1991, pp. 922–933.
300. Ross K.W., Yao D.D.: *Optimal load balancing and scheduling in a distributed computer system.* J. ACM, Vol. 38, No. 3, 1991, pp. 676–690.
301. Russell S. J., Norvig P.: *Artificial intelligence: A modern approach*. Prentice Hall, New Jersey 2002.

302. Ruzika S., Wiecek M.: *Approximation methods in multiobjective programming*. Journal of Optimization Theory and Applications, Vol. 126, No. 3, 2005, pp. 473-501.
303. Saka M.P.: *Optimum design of steel sway frames to BS5950 using harmony search algorithm.* J. Constr. Steel Res., Vol. 65, No. 1, 2009, pp. 36–43.
304. Salcedo-Sanz S., Manjarres D., Pastor-Sanchez A., Del Ser J., Portilla-Figueras J.A., Gil-Lopez S.: *One-way urban traffic reconfiguration using a multi-objective harmony search approach.* Expert Syst. Appl., Vol. 40, No. 9, 2013, pp. 3341–3350.
305. Samuel A.L.: *Programming computers to play games.* Advances in Computers, Vol. 1, 1960, pp. 165-192.
306. Sarvari H., Khairdoost N., Fetanat A.: *Harmony search algorithm for simultaneous clustering and feature selection.* Int. Conf. of Soft Computing and Pattern Recognition, 2010, pp. 202–207.
307. Sarvari H., Zamanifar K.: *A self-adaptive harmony search algorithm for engineering and reliability problems*. Second Int. Conf. on Computational Intelligence, Modelling and Simulation, 2010, pp. 59–64.
308. Schneidewind N.: *Allocation and analysis of reliability: multiple levels: system, subsystem, and module*, Innovations in System and Software Engineering, Springer London, Vol. 2, No. 3-4, December 2006, pp. 121-136.
309. Schuetze O., Laumanns M., Tantar E., Coello Coello C.A.: *Computing gap-free Pareto front approximations with stochastic search algorithms.* Evolutionary Computation, Vol. 18, No. 1, 2010, pp. 65-96.
310. Shariatkhah M.H. *et al*.: *Duration based reconfiguration of electric distribution networks using dynamic programming and harmony search algorithm*. Int. J. Electr. Power Energy Syst., Vol. 41, No. 1, 2012, pp. 1–10.
311. Shatz S.M., Wang J.P.: *Models & algorithms for reliability-oriented task-allocation in redundant distributed-computer systems*. IEEE Transactions On Reliability, Vol. 38, No. 1, April 1989, pp. 16-27.
312. Shibata T., Choi S., Taura K.: *File-access patterns of data-intensive workflow applications.* Proc. of the 10th IEEE/ACM Int. Conf. on Cluster, Cloud and Grid Computing, 2010, pp. 522–525.
313. Shirazi B.A., Hurson A.R., Kavi K.M.: *Scheduling and load balancing in parallel and distributed systems.* IEEE Computer Society Press, Los Alamitos, California 1995.
314. Sinclair J.B.: *Optimal assignments in broadcast networks.* IEEE Transactions on Computers, No. 5, Vol. C-37, May 1988, pp. 521-531.
315. Singhal M., Shivaratri N.G.: *Advanced concepts in operating systems – distributed, database, and multiprocessor operating systems.* The McGraw-Hill Companies, Columbus 1994.
316. Sinha P.K.: *Distributed operating systems – concepts and design*. IEEE Press, San Francisco 1997.
317. Sinsupan N., Leeton U., Kulworawanichpong T.: *Application of harmony search to optimal power flow problems*. Int. Conf. on Advances in Energy Engineering, 2010, pp. 219–222.
318. Sirjani R., Mohamed A., Shareef H.: *Optimal allocation of shunt var compensators in power systems using a novel global harmony search algorithm*. Int. J. Electr. Power Energy Syst., Vol. 43, No. 1, 2012, pp. 562–572.
319. Sirjani R., Mohamed A.: *Improved harmony search algorithm for optimal placement and sizing of static var compensators in power systems*. First Int. Conf. on Informatics and Computational Intelligence, 2011. pp. 295–300.
320. Sirjani R., Mohamed A., Shareef H.: *An improved harmony search algorithm for optimal capacitor placement in radial distribution systems*. The 5th Int. Conf. on Power Engineering and Optimization, 2011, pp. 323–328.
321. Sivasubramani S., Swarup K.S.: *Multi-objective harmony search algorithm for optimal power flow problem*. Int. J. Electr. Power Energy Syst., Vol. 33, 2011, pp. 745–752.
322. Sivasubramani S.:, Swarup K.S.: *Environmental/economic dispatch using multi-objective harmony search algorithm*. Electr. Power Syst. Res., Vol. 81, No. 9, 2011, pp. 1778–1785.
323. Srinivas N., Deb K.: *Multiobjective optimization using nondominated sorting in genetic algorithms*, Evolutionary Computation, No. 2, Vol. 3, 1994, pp. 221–248.
324. Srinivasa R.R. *et al*.: *Optimal network reconfiguration of large-scale distribution system using harmony search algorithm*. IEEE Trans. Power Syst., Vol. 26, No. 3, 2011, pp. 1080–1088.
325. Steuer R.E. *Multiple criteria optimization: theory, computation, and application*. Willey, New York 1986,
326. Stone H.S.: *Critical load factors in two-processor distributed systems*. IEEE Transactions on Software Engineering, Vol. SE-4, No. 3, May 1978, pp. 254-258.
327. Sui J. *et al*.: *Mine ventilation optimization design based on improved harmony search*. In: Int. Conf. on Information Engineering, Vol. 1, 2010, pp. 67–70.

328. Sylva J., Crema A.: *A method for finding the set of non-dominated vectors for multiple objective integer linear programs*. European Journal of Operational Research, Vol. 158, No. 1, 2004, pp. 46-55.
329. Szabo C. *et al*.: *Science in the cloud: allocation and execution of data-intensive scientific workflows*. Journal of Grid Computing, October 2013, pp. 223-233.
330. Szabo C., Kroeger T.: *Evolving multi-objective strategies for task allocation of scientific workflows on public clouds*. IEEE Congress on Evolutionary Computation, 2012, pp. 1–8.
331. Taleizadeh A.A., Niaki S.T.A., Seyedjavadi S.M.H.: *Multi-product multi-chance-constraint stochastic inventory control problem with dynamic demand and partial back-ordering: a harmony search algorithm.* J. Manuf. Syst., Vol. 31, No. 2, 2012, pp. 204–213.
332. Tanenbaum A. S., van Steen M.: *Distributed systems: Principles and paradigms*. Prentice-Hall, New Jersey 2006.
333. Tanenbaum A. S.: *Computer networks*. Prentice-Hall, New Jersey 2003.
334. Tang X., Chanson S.T.: *Optimizing static job scheduling in a network of heterogeneous computers.* Proc. of Int. Conf. on Parallel Processing, Toronto, Canada, 2000, pp. 373–382.
335. Tangpattanakul P., Meesomboon A., Artrit P.: *Optimal trajectory of robot manipulator using harmony search algorithms*. Stud. Comput. Intell., Vol. 270, 2010, pp. 23–36.
336. Tantawi A.N., Towsley D.: *Optimal static load balancing in distributed computer systems.* J. ACM, Vol. 32, No. 2, 1985, pp. 445–465.
337. Thains D., Tannenbaum T., Livny M.: *Distributed computing in practice: the condor experience*. Concurr. Comput. Pract. Exp., Vol. 17, No. 2-4, 2005, pp. 323–356.
338. Thanikesavan S., Killat U.: *Global scheduling of periodic tasks in a decentralized real-time control system*, Proc. of the IEEE Int. Workshop on Factory Communication Systems, September 2004, pp. 307-310.
339. Towsley D.P.: *Allocating programs containing branches and loops within a multiple processor system*. IEEE Transactions on Software Engineering, Vol. SE-12, October 1986, pp. 1018-1024.
340. Varvarigou T., Trotter J.: *Module replication for fault-tolerant real-time distributed systems*. IEEE Transaction on Reliability, Vol. 47, No. 1, 1998, pp. 8-18.
341. Vasebi A., Fesanghary M., Bathaeea S.M.T.: *Combined heat and power economic dispatch by harmony search algorithm*. Int. J. Electr. Power Energy Syst., Vol. 29, No. 10, 2007, pp. 713–719.
342. Verma A., Panigrahi B.K., Bijwe P.R.: *Harmony search algorithm for transmission network expansion planning*. IEEE Proc. Generat. Transm. Distrib., Vol. 4, No. 6, 2010, pp. 663–673.
343. Wang C.M., Huang Y.F.: *Self-adaptive harmony search algorithm for optimization*, Expert Syst. Appl., Vol. 37, pp. 4, 2010, pp. 2826–2837.
344. Wang L., Li L.P.: *A coevolutionary differential evolution with harmony search for reliability-redundancy optimization*. Expert Syst. Appl., Vol. 39, No. 5, 2012, pp. 5271–5278.
345. Wang L., Pan Q.K., Tasgetiren M.F.: *A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem.* Comput. Ind. Eng., Vol. 61, No. 1, 2011, pp. 76–83.
346. Wang L., Pan Q.K., Tasgetiren M.F.: *Harmony filter: a robust visual tracking system using the improved harmony search algorithm.* Image Vis. Comput., Vol. 28, No. 12, 2010, pp. 1702–1716.
347. Wang L., Pan Q.K., Tasgetiren M.F.: *Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms*. Expert Syst. Appl., Vol. 37, No. 12, 2010, pp. 7929–7936.
348. Wang F.R. *et al*.: *A novel binary harmony search algorithm for intelligent test-sheet composition*. Int. Conf. on Electrical and Control Engineering, 2011, pp. 6213–6216.
349. Wang L. *et al*.: *A hybrid binary harmony search algorithm inspired by ant system.* IEEE Int. Conf. on Cybernetics and Intelligent Systems, 2011, pp. 153–158.
350. Wang, Y. *et al*.: *Interconnect EGEE and CNGrid e-Infrastructures through interoperability between gLite and GOS middlewares.* IEEE Computer Society, Piscataway, 2007, pp. 553–560.
351. Wei L. *et al*.: W*aveform matching approach for fault diagnosis of a high-voltage transmission line employing harmony search algorithm.* IEEE Proc. Generat. Transm. Distrib., Vol. 4, No. 7, 2010, pp. 801–809.
352. Węglarz J., Błażewicz J., Kovalyov M.: *Preemptable malleable task scheduling problem*. IEEE Transactions on Computers, Vol. 55, No. 4, April 2006, pp. 486-490.
353. Widell N., Nyberg C.: *Cross entropy based module allocation for distributed systems.* Proc. of the 16th IASTED Int. Conf. on Parallel and Distributed Computing and Systems, Cambridge, Massachusetts, USA, November 2004, pp. 223-231.
354. Wierzbicki A. P.: *A mathematical basis for satisficing decision making*. *Mathematical Modelling*, Vol. 3, No. 5, 1982, pp. 391-399.
355. Wolpert D.H., MacReady W.G.: *No free lunch theorems for optimization*. IEEE Trans. Evol. Comput., Vol. 1, No 1, 1997, pp. 67–82.

356. Wong W.K., Guo Z.X.: *A hybrid intelligent model for medium-term sales forecasting in fashion retail supply chains using extreme learning machine and harmony search algorithm*. Int. J. Prod. Econ., Vol. 128, No. 2, 2010, pp. 614–624.
357. Xie T., Qin X.,: *A new allocation scheme for parallel applications with deadline and security constraints on clusters*. Proc. of the IEEE Int. Conf. on Cluster Computing, Boston, Massachusetts, USA, September 2005, pp. 1-10.
358. Xu H. *et al*.: *Harmony search optimization algorithm: application to a reconfigurable mobile robot prototype*. Stud. Comput. Intell., Vol. 270, 2011, pp. 11–22.
359. Yadav P. *et al*: *An improved harmony search algorithm for optimal scheduling of the diesel generators in oil rig platforms*. Energ. Convers. Manag., Vol. 52, No. 2, 2011, pp. 893–902.
360. Yadav P. *et al*.: *Improved harmony search algorithm based optimal design of the brushless DC wheel motor*. IEEE Int. Conf. on Sustainable Energy Technologies, 2010, pp. 1–6.
361. Yazdi E., Azizi V., Haghighat A.T.: *A new biped locomotion involving arms swing based on neural network with harmony search optimizer*. IEEE Int. Conf. on Automation and Logistics, 2011, pp. 18–23.
362. Yigitbasi N., Iosup A., Epema D.: *C-meter: a framework for performance analysis of computing clouds.* Cluster Computing and the Grid, 2009, pp. 472–477.
363. Yildiz A.R., Öztürk F.: *Hybrid Taguchi-harmony search approach for shape optimization: recent advances in harmony search algorithm*. Stud. Comput. Intell., Vol. 270, 2010, pp. 89–98.
364. Yu J., Buyya R.: *A taxonomy of workflow management systems for Grid computing.* J. Grid Comput. Vol. 3, No. 3–4, 2005, pp. 171–200.
365. Yusof U.K., Budiarto R., Deris S.: *Harmony search algorithm for flexible manufacturing system (FMS) machine loading problem*. In: Conf. on Data Mining and Optimization, Malaysia, 2011, pp. 26–31.
366. Zacniewski A.: *Optymalizacja alokacji modułów programistycznych w rozproszonym systemie szkolenia wojskowego*. Rozprawa doktorska, Wydział Elektroniki, Telekomunikacji i Informatyki, Politechnika Gdańska, Gdańsk 2011.
367. Zha L. *et al*.: *System software for China national Grid.* In: Jin, H., Reed, D.A., Jiang, W. (eds.) NPC. Lecture Notes in Computer Science, Vol. 3779, pp. 14–21. Springer, New York 2005, pp. 223-229.
368. Zhang R., Hanzo L.: *Iterative multiuser detection and channel decoding for DS-CDMA using harmony search*. IEEE Signal Process. Lett., Vol. 16, No. 10, 2010, pp. 917–920.
369. Zhang W., Reimann M.: *A simple augmented ε-constraint method for multi-objective mathematical integer programming problems*. European Journal of Operational Research, Vol. 234, No. 1, 2014, pp. 15-24.
370. Zhao S.Z., Suganthan P.N., Das S.: *Dynamic multi-swarm particle swarm optimizer with sub-regional harmony search*. IEEE Congress on Evolutionary Computation, 2010, pp. 1–8.
371. Zitzler E., Thiele L.: *Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach.* IEEE Transactions on Evolutionary Computation, No. 3, Vol. 4, November 1999, pp. 257–271.
372. Zomaya A., Teh Y.: *Observations on using genetic algorithms for dynamic load-balancing.* IEEE Trans. Parallel Distrib. Syst., Vol. 12, 2001, pp. 899–911.
373. Zou D. *et al*.: *A novel global harmony search algorithm for reliability problems.* Comput. Ind. Eng., Vol. 58, No. 2, 2010, pp. 307–316.
374. Zou D. *et al*.: *Novel global harmony search algorithm for unconstrained problems.* Neurocomputing, Vol. 73, No. 16–18, 2010, pp. 3308–3318.
375. Zou D. L. *et al*.: *An effective global harmony search algorithm for reliability problems.* Expert Syst. Appl., Vol. 38, No. 4, 2011, pp. 4642–4648.
376. Zou D. *et al*.: *Solving 0–1 Knapsack problem by a novel global harmony search algorithm.* Appl. Soft Comput., Vol. 11, No. 2, 2011, pp. 1556–1564.
377. Zou, D. *et al*.: *A novel global harmony search algorithm for chemical equation balancing.* Int. Conf. on Computer Design and Applications, China, Vol. 2, 2011, pp. 1–5.

378. *Benchmark report for SAS Grid Manager on HP ProLiant BL460c G7 Blades and 3PAR T800 Storage System.* Hewlett-Packard Technical white paper, http://h20195.www2.hp.com/v2/getpdf.aspx/4AA3-8251ENW.pdf?ver=1.0, dostęp: 10.11.2014.
379. *BOINC*, http://www.boinc.com, Accessed 20 November 2014.
380. *BOINC Poland*, *http://www.boincatpoland.org/*, Accessed 6 December 2014.
381. *Comcute*, http://comcute.eti.pg.gda.pl/, Accessed 20 March 2014.

382. *Computer System AMD Sempron 145 & 3 Asus GeForce GTX 760*, http://www.freezepage.com/1384601420XCIGYKCBKJ, Accessed 20 November 2014.
383. *Computer System Pentium G550 & 2 Radeon R9 290*, http://www.freezepage.com/1387480124PSLSILVCMJ, Accessed 20 November 2014.
384. *Einstein@Home*, http://einstein.phys.uwm.edu/, Accessed 6 December 2014.
385. *Folding@home*, http://folding.stanford.edu/home/, Accessed 6 December 2014.
386. *GISELA*: *Grid Infrastructures for e-Science virtual communities in Europe and Latin-America (GISELA) project*. http://www.gisela-grid.eu/, Accessed 20 March 2014.
387. *Globus*. http://www.globus.org/. Accessed 20 March 2014.
388. *Great Internet Mersenne Prime Search – GIMPS*. http://www.mersenne.org/, Accessed 20 March 2014.
389. *Grid and Cloud Computing: Opportunities for Integration with the Next Generation Network.* http://Gridsam.sourceforge.net/2.1.3/index.html, Accessed 20 November 2014.
390. *MilkyWay@Home*, http://milkyway.cs.rpi.edu/milkyway/, Accessed 20 November 2014.
391. *Networked European Software and Services Initiative: Business Grids Vision and Strategic Research Agenda v3.0*. http://www.nessi-europe.com/, Accessed 20 March 2014.
392. *NextGrid.* http://www.nextGrid.org/, Accessed 20 March 2014.
393. *Open Science Grid*. http://www.openscienceGrid.org/, Accessed 20 March 2014.
394. *Oprogramowanie warstwy pośredniczącej.* http://www.middleware.org, dostęp 12.11.2014.
395. *Organization for the Advancement of Structured Information Standards (OASIS).* http://www.oasis-open.org, Accessed 20 March 2014.
396. Paluszak J.: *Harmony search application in Java*. http://www.harmonysearch.info, Accessed 20 November 2014.
397. *PlGrid*. http://www.plgrid.pl/, dostęp 12.11.2014.
398. *SETI@Home. http://setiathome.berkeley.edu/*, Accessed 20 March 2014.
399. *Sony Sparks Price War With PS4 Priced at $399*. *CNBC*. http://www.cnbc.com/id/100805004, Accessed 20 March 2014.
400. *Technologia Java EE*. http://java.sun.com/javaee, dostęp 12.11.2014.
401. *TeraGrid*. http://www.teraGrid.org/, Accessed 20 March 2014.
402. *The Bitcoin Network. Some statistics.* Staffordshire University. http://bitcoinwatch.com/, Accessed 6 December 2014.
403. *The Frontline solvers*. http://www.solver.com/, Accessed 6 December 2014.
404. *The Green500 List - November 2014.* http://www.green500.org/, Accessed 20 November 2014.
405. *The Linpack Benchmark*. http://www.top500.org/project/linpack/, Accessed 20 November 2014.
406. *The Top 500*. http://www.top500.org/, Accessed 30 November 2014.
407. *TOP5 – benchmarki dla smartfonów z Androidem*. http://smartfony.play.pl/, dostęp: 10.11.2014.
408. *Whetstone benchmarks*. http://www.roylongbottom.org.uk/whetstone.htm, Accessed 20 November 2014.
409. *Zestaw komputerów ATX Rack 19 Infotronik*. *http://www.infotronik.pl/komputery_rack.php*, dostęp: 10.12.2014.
410. *Zestaw serwerów PowerServer*. *http://www.powerserver.pl/*, dostęp: 10.12.2014.
411. *Zestaw systemów serwerowych. http://www.bizserver.eu/pl/sklep/systemy_serwerowe/*, dostęp: 10.12.2014.
412. Zieliński P.: *Testy wydajnościowe aplikacji webowych*. http://msdn.microsoft.com.pl/, dostęp: 10.11.2014.

WYKAZ RYSUNKÓW

WYKAZ TABEL

DODATEK. Wybrane eksperymenty numeryczne

Na stronie WWW *http://www.harmonysearch.pl* autor udostępnił pakiet ADAIORG'14 z opracowanymi algorytmami harmonicznymi do wyznaczania rozwiązań dla sześciu sformułowanych w rozprawie zagadnień optymalizacji konfiguracji gridu [396]. Poniżej zaprezentowano trzy reprezentatywne eksperymenty numeryczne wykonane za pomocą powyższego pakietu.

Eksperyment numeryczny nr 1

Rozważa się optymalizację konfiguracji gridu dla danych wejściowych zagadnienia (3.3.1), które zapisano w pliku *benchmarkNr1.txt*, jak niżej.

```
[V – liczba modułów]
12
[I – liczba węzłów]
6
[J – liczba rodzajów komputerów]
3
[xm – aktualna konfiguracja gridu]
2 5 1 5 6 1 4 4 3 6 2 2
[xpi]
2 3 2 1 2 3
[D – macierz dostępnych wielkości pamięci RAM i HD]
8.256 20.0 38.0
0.5 3 6
[C - Macierz zapotrzebowań na wiekość pamięci RAM i HD]
0.5 0.3 0.4 0.3 0.5 1.7 2.7 0.3 0.4 0.5 0.3  0.5
0.01 0.3 0.02 0.02 0.03 0.03 0.02 0.03 0.04 0.02 0.03 0.02
[T - Macierz czasów wykonania modułów]
1 3 2 4 3 1 2 1 2 3 1 3
2 2 2 3 2 2 3 2 1 4 2 2
3 1 3 3 3 1 2 1 2 2 2 1
[Tau - Macierz czasów komunikacji między modułami]
0 1 2 1 4 3 2 3 2 4 1 2
3 0 1 2 3 4 0 4 4 4 1 5
1 4 0 1 1 0 1 2 2 1 5 5
7 2 8 0 7 1 7 4 7 7 2 5
8 0 8 8 0 8 1 5 2 5 6 8
4 8 8 2 8 0 3 2 7 4 8 2
8 4 6 0 5 5 0 7 2 7 8 1
7 1 2 1 4 7 1 0 9 9 4 1
2 1 0 5 1 1 4 9 0 9 1 4
7 0 1 9 0 2 9 1 7 0 9 1
4 7 4 1 5 9 0 9 0 9 0 1
8 1 7 0 1 0 9 1 9 4 1 0
[theta min – wymagana minimalna łączna wydajność gridu]
20
[theta grid – wektor wydajności poszczególnych rodzajów komputerów]
1.84 4.848 6.771
[xi max – maksymalny koszt]
7000
[xi grid – wektor kosztów zakupu poszczególnych rodzajów komputerów]
400 896 1504
[upsilon min – wymagany minimalny stopień rozproszenia modułów]
80
[epsilon max – wymagane maksymalne zużycie energii elektrycznej przez grid]
3800
[epsilon grid – wektor poboru mocy przez poszczególne rodzaje komputerów]
250 600 650
```

Rys. 44. Dane wejściowe w pliku *benchmarkNr1.txt*
Źródło: opracowanie własne

Liczba kodowanych całkowitoliczbowo rozwiązań dopuszczalnych wynosi co najwyżej $1{,}59*10^{12}$, a liczba wszystkich rozwiązań kodowanych binarnie to $1{,}24*10^{27}$. Metoda przeglądu wariantów całkowitoliczbowych na komputerze klasy *PC/Windows 7/Intel Core i7-2670 QM 2,2 GHz* wymaga ponad 569 dni obliczeń.

Celem eksperymentu jest dostrojenie parametrów *HMCR, PAR, BW1, BW2* oraz *HMS* w algorytmie harmonicznym HSΔmax dla instancji nr 1. Zakłada się następujące wartości wag ρ_1=0,5, ρ_2=0,5.

Dla *HMCR*=0,7 *PAR*=0,7, *BW1*=1, *BW2*=1, *HMS*=10 wyznaczono w jednym z przebiegów algorytmu *HSΔmax* wartość $\Delta_{\max}$=82,5 [s] po 464 077 improwizacjach, co zajęło 19,72 [s] na komputerze klasy *PC/Windows 7/Intel Core i7-2670 QM 2,2 GHz*. Dla 30 przebiegów algorytmu wyznaczono konfigurację suboptymalną cechującą się $\Delta_{\max}$=82,5 [s]. Liczba iteracji do wyznaczenia optimum wynosiła: od 232 737 do 2 205 163, przy czym średnia wartość to 1 226 145.

Jednym ze sposobów zwiększenia efektywności algorytmu harmonicznego jest dopasowanie wartości parametrów *HMCR* i *PAR*. Zakładając, że parametry te nie zmieniają się podczas obliczeń, możliwe ich podstawowe kombinacje przedstawiono w tabeli nr 7. Przyjęto, że wartości pozostałych parametrów są następujące: *BW1*=1, *BW2*=1 oraz *HMS*=10.

Tabela 7. Średnia liczba improwizacji niezbędna do wyznaczenia konfiguracji suboptymalnej cechującej się $\Delta_{\max}$=82,5 [s] dla wybranych wartości parametrów *HMCR, PAR*, przy czym *BW1*=1, *BW2*=1, *HMS*=10

	HMCR				
PAR	*0,1*	*0,3*	*0,5*	*0,7*	*0,9*
0,1	3 927 038	3 845 374	3 112 857	1 226 145	618 750
0,3	3 837 746	3 837 647	3 937 465	1 226 145	685 273
0,5	3 847 937	3 736 487	3 745 836	1 534 736	**534 377**
0,7	3 547 362	3 437 773	3 232 221	1 226 145	592 353
0,9	3 464 675	4 347 443	3 343 847	2 545 534	1 458 282

Źródło: opracowanie własne.

Średnia liczba improwizacji niezbędna do wyznaczenia konfiguracji suboptymalnej cechującej się $\Delta_{\max}$=82,5 [s] jest najmniejsza dla *HMCR*=0,7, *PAR*=0,5, przy czym *BW1*=1, *BW2*=1 oraz *HMS*=10. Najszybszą zbieżność uzyskano dla *HMCR*=0,9 oraz *PAR*<0,9. Jeśli *HMCR*=1, to zachodzi przedwczesna zbieżność do jednego z lokalnych minimów, gdyż losowane są jedynie współrzędne z pamięci HM. Wprawdzie z prawdopodobieństwem *PAR* zmienna decyzyjna może być modyfikowana, to dla niewielkich wartości parametrów *BW1* i *BW2* zmiany są niewielkie. Z kolei wariant *HMCR*=0 prowadzi do przeszukiwania losowego, co skutkuje „przeskakiwaniem" nad optimum globalnym.

Zakładając, że parametry *BW1, BW2* nie zmieniają się podczas przebiegu algorytmu, możliwe ich kombinacje przedstawiono w tabeli nr 8. Najkorzystniejsze wyniki uzyskano dla *BW1*=1 oraz *BW2*=1.

Tabela 8. Średnia liczba improwizacji niezbędna do wyznaczenia konfiguracji suboptymalnej cechującej się Δ_{max}=82,5 [s] dla wybranych wartości parametrów *BW1, BW2,* przy czym *HMCR*=0.9, *PAR=0.5, HMS*=10

	BW2	
BW1	*1*	*2*
1	**534 377**	997 547
2	578 374	1 123 758
3	618 889	1 258 359

Źródło: opracowanie własne.

Istotnym parametrem jest *HMS* - liczba rozwiązań w tablicy harmonicznej. Zwiększanie wielkości tablicy *HMS* nie musi powodować poprawienia jakości wyznaczonych konfiguracji oraz szybszą zbieżność. Zakładając, że parametr *HMS* nie zmienia się podczas obliczeń, średnią liczbę improwizacji niezbędną do wyznaczenia konfiguracji suboptymalnej dla wybranych wartości parametru *HMS* przedstawiono w tabeli nr 9. Zwraca uwagę relatywnie szybka zbieżność algorytmu dla małych wartości *HMS*.

Tabela 9. Średnia liczba improwizacji niezbędna do wyznaczenia konfiguracji suboptymalnej cechującej się Δ_{max}=82,5 [s] dla wybranych wartości parametru *HMS*, przy czym *HMCR*=0,9, *PAR=0,5, BW1=1, BW2=1*

HMS	NG_{max}
1	892 500
5	586 429
10	**534 377**
20	753 826
50	973 948
100	1 228 392

Źródło: opracowanie własne.

Reasumując, średnia liczba improwizacji niezbędna do wyznaczenia konfiguracji suboptymalnej cechującej się Δ_{max}=82,5 [s] jest najmniejsza dla następujących parametrów algorytmu harmonicznego: *HMCR*=0.9, *PAR=0.5, BW1=1, BW2=1, HMS*=10. Wyznaczoną konfigurację przedstawiono na rysunku 36, a przykładowy przebieg algorytmu harmonicznego zilustrowano na rysunku 37. Warto podkreślić, że jeśli zbieżność algorytmu harmonicznego lub jakość wyznaczonych konfiguracji nie są zadowalające dla innych instancji, to można poprawić tą sytuację za pomocą dostrojenia parametrów *HMCR*, *PAR*, *BW1*, *BW2* oraz *HMS*.

Eksperyment numeryczny nr 2

Rozważa się instancję problemu optymalizacji dwukryterialnej (3.4.7), w którym poszukuje się reprezentacji zbioru konfiguracji *Pareto*-optymalnych przy jednoczesnej minimalizacji $\hat{Z}_{max}$ oraz $\tilde{Z}_{max}$. Dane wejściowe zapisano w pliku *benchamrkNr2.txt* (rys. 45).

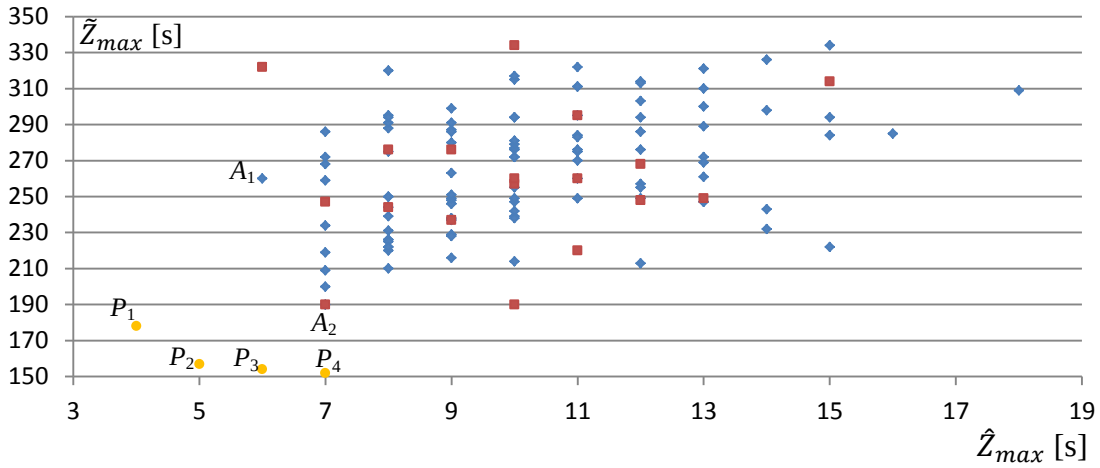
```
[V]
24
[I]
9
[J]
10
[xm]
2 5 1 5 6 1 4 4 3 6 2 8 2 5 8 5 9 1 9 4 7 6 7 2
[xpi]
2 3 2 1 2 3 4 5 7
[D]
8.256 20.0 38.0 48.256 70.0 138.0 184.256 200.0 380.0 500.0
0.5 3 6 10.5 12 17 20.5 23 30 42
[C]
0.5 0.3 0.4 0.3 0.5 1.7 2.7 0.3 0.4 0.5 0.3  0.5 0.5 0.3 0.4 0.3 0.5 1.7 2.7 0.3 0.4 0.5 0.3  0.5
0.01 0.3 0.02 0.02 0.03 0.03 0.02 0.03 0.04 0.02 0.03 0.02 0.01 0.3 0.02 0.02 0.03 0.03 0.02 0.03 0.04 0.02 0.03 0.02
[T]
1 3 2 4 3 1 2 1 2 3 1 3 1 3 2 4 3 1 2 1 2 3 1 3
2 2 2 3 2 2 3 2 1 4 2 2 2 2 2 3 2 2 3 2 1 4 2 2
3 1 3 3 3 1 2 1 2 2 2 1 3 1 3 3 3 1 2 1 2 2 2 1
1 2 1 3 2 1 2 1 2 2 2 1 3 1 2 2 3 3 1 2 1 2 3 1 2
1 2 1 1 2 1 3 2 1 4 2 1 2 2 1 3 2 2 3 1 1 4 2 1
2 1 2 3 2 1 2 1 2 2 2 2 1 2 1 2 2 2 2 1 2 1 2 1
1 1 2 2 1 1 2 1 2 1 1 3 1 3 2 1 3 1 2 1 2 1 1 3
1 2 1 1 1 1 2 2 1 1 2 2 1 2 2 1 2 2 2 3 2 1 2 2
1 1 1 3 1 1 2 1 2 2 2 1 1 1 1 3 3 1 1 2 1 2 1 2 1
1 1 1 1 1 1 2 1 2 2 2 2 1 1 1 1 1 1 1 2 1 2 1 2 1
[Tau]
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 1 0
[theta min]
200
[theta grid]
1.84 4.848 6.771 12.84 14.848 16.771 21.84 24.848 26.771 40.2
[xi max]
70000
[xi grid]
400 896 1504 1900 2896 3504 4000 4896 5504 7012
[upsilon min]
180
[epsilon max]
38000
[epsilon grid]
250 600 650 750 800 950 1250 1600 1650 2100
```

Rys. 45. Dane wejściowe zapisane w pliku *benchmarkNr2.txt*
Źródło: opracowanie własne

Uwzględnia się następujące ograniczenia: wydajnościowe, kosztu zakupu komputerów, stopnia rozproszenia modułów, wielkości pamięci RAM, wielkości pamięci dyskowej oraz energetyczne.

W porównaniu do instancji nr 1 liczba modułów jest zwiększona z 12 do 24, liczba węzłów z 6 do 9, a liczba rodzajów komputerów z 3 do 10. Instancja cechuje się 306 zero-jedynkowymi zmiennymi decyzyjnymi, a binarna przestrzeń przeszukiwań zawiera $1{,}3*10^{92}$ rozwiązań. Liczba konfiguracji, które spełniają warunki formalne, wynosi $7{,}9*10^{31}$, przy czym nie wszystkie są rozwiązaniami dopuszczalnymi. Metoda przeglądu i oceny wszystkich konfiguracji na superkomputerze *Tianhe-2* wymagałaby 74 393 451 lat obliczeń.

Na rysunku 46 przedstawiono oceny konfiguracji dopuszczalnych, które wyznaczono za pomocą metody losowej. Wygenerowano 100 konfiguracji, z których do pamięci harmonicznej zakwalifikowano *HMS*=20 rozwiązań.



Rys. 46. Oceny konfiguracji wygenerowane za pomocą metody losowej na tle ocen suboptymalnych w sensie *Pareto* wyznaczonych za pomocą algorytmu harmonicznego MOHS3
Źródło: opracowanie własne

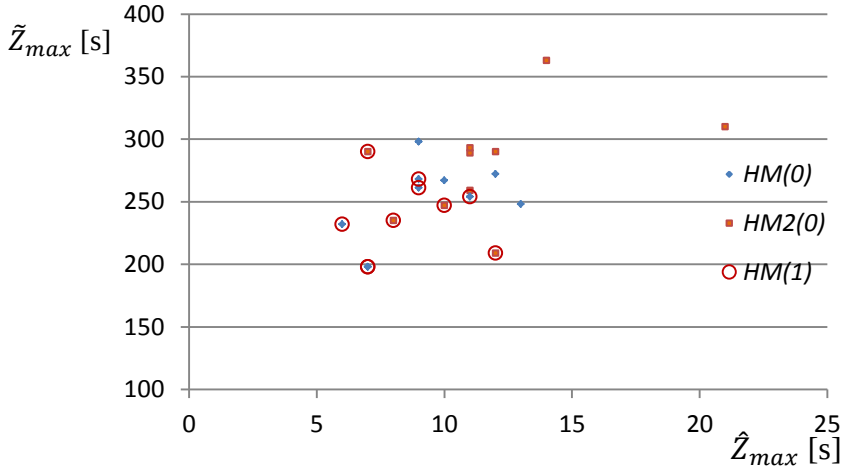
Za pomocą metody losowej wygenerowano konfigurację gridu cechującą się oceną A_1=(6; 260) [s]. Konfiguracja X^{α}=[7,4,4,7,5,6,9,1,1,3,3,9,1,2,8,1,5,8,7,9,5,7,6,8]^T, X^{β}=[8,1,4,6,8,7,7,2,9]^T ma następujące parametry: Z_{suma}=1 376 [s], $\Delta_{max} = 133\ [s]$ dla $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$, Θ=156,446 [TFLOPS], $\Xi(x)$=29 996 [USD], υ=269.

Drugą niezdominowaną oceną w zbiorze konfiguracji początkowych, wyznaczonego za pomocą metody losowej jest A_2=(7; 190) [s]. Konfiguracja cechuje się następującymi parametrami: Z_{suma}=1388 [s], $\Delta_{max} = 98{,}5\ [s]$ dla $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$,

Θ=138,454 [TFLOPS], $\Xi(x)$=26 992 [USD], υ=271. Specyfikacja konfiguracji to: X^{α}=[1,5,8,1,8,5,8,6,2,3,7,9,4,4,9,9,7,7,1,9,5,4,6,3]^T oraz X^{β}=[5,7,6,6,2,5,1,7,8]^T.

Do pierwszej tablicy harmonicznej zakwalifikowano oceny zaznaczone kwadratami (rys. 46). Ustalono następujące wartości parametrów: *HMCR*=0.9, *PAR=0.5*, *BW1=2*, *BW2=3*, *HMS*=10. Ponadto pomocnicza pamięć harmoniczna cechowała się rozmiarem *HMS2*=10.

Na rysunku 47 zaprezentowano zasadę wyznaczania nowej tablicy harmonicznej *HM*(1) na podstawie jej zawartości z poprzedniej iteracji *HM*(0) oraz pomocniczej tablicy *HM2* w pierwszej iteracji jednego z przebiegów algorytmu MOHS3, w którym procedura nadawania rang opiera się na algorytmie *Fonseci-Fleminga*. Ponadto rozwiązania z rangą graniczną dobierane są w sposób losowy. Cztery konfiguracje z pamięci początkowej *HM* nie zakwalifikowano do nowej pamięci *HM*.



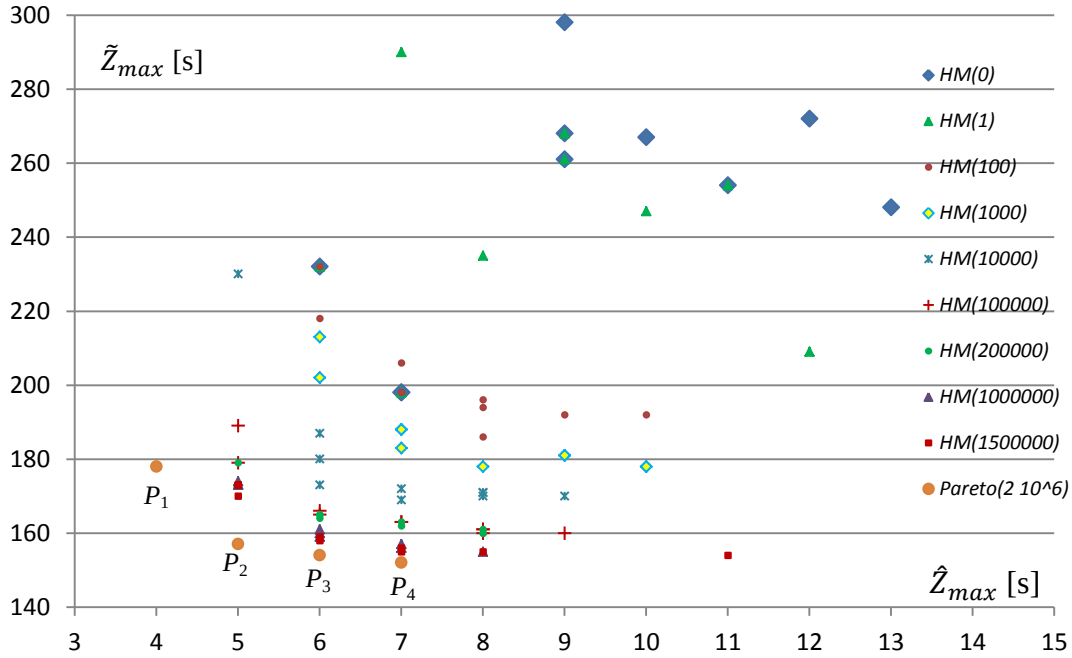
Rys. 47. Wyznaczenie nowej tablicy harmonicznej na podstawie jej zawartości z poprzedniej iteracji oraz pomocniczej tablicy harmonicznej
Źródło: opracowanie własne

Na rysunku 48 zobrazowano migrację rozwiązań z tablicy harmonicznej *HM* wraz ze wzrostem numeru jej aktualizacji *NHM* do wartości 2 000 000. Do wyznaczenia nowej tablicy harmonicznej niezbędnych jest *HMS2* improwizacji. Otrzymano cztery oceny niezdominowane: P_1=(4; 178), P_2=(5; 157), P_3=(6; 154) i P_4=(7; 152). Na komputerze klasy *PC Intel Core i7-2670QM 2,20 GHz* czas wyznaczenia reprezentacji konfiguracji *Pareto*-suboptymalnych wyniósł 5 735 [s].

Konfiguracja o ocenie P_1=(4; 178) cechuje się parametrami: Z_{suma}=1344 [s], $\Delta_{max} = 91\ [s]$ dla $\rho_1 = 0{,}5, \rho_2 = 0{,}5$, Θ=201,464 [TFLOPS], $\Xi(x)$=39 488 [USD], υ=265.

Specyfikacja konfiguracji to: X^{α}=[5,2,8,5,8,1,4,6,7,3,5,9,4,7,3,4,9,3,8,2,6,1,1,6]^T, X^{α}=[7,8,7,8,8,6,5,9,8]^T.

Natomiast konfiguracja o ocenie P_2=(5; 157) cechuje się parametrami: Z_{suma}=1348 [s], $\Delta_{max} = 81$ [s] dla $\rho_1 = 0{,}5$, $\rho_2 = 0{,}5$, Θ=212,5 [TFLOPS], $\Xi(x)$=41 776 [USD], υ=269. Specyfikacja to: X^{α}=[3,6,5,5,6,7,8,9,2,1,2,5,4,3,6,1,9,3,8,4,4,2,7,1]^T, X^{β}=[8,8,8,8,8,5,8,7,9] ^{T}.



Rys. 48. Migracja ocen z *HM* w kierunku reprezentacji ocen *Pareto*-suboptymalnych dla benchmarku nr 2 za pomocą algorytmu harmonicznego *MOHS3*
Źródło: opracowanie własne

Konfiguracja o ocenie P_3=(6, 152) cechuje się parametrami: Z_{suma}=1318 [s], $\Delta_{max} = 79$ [s] dla $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$, Θ=44,223 [TFLOPS], $\Xi(x)$=9 616 [USD], υ=263. Specyfikacja konfiguracji to: X^{α}=[5,7,5,6,3,7,4,1,2,9,3,2,6,9,8,4,9,8,1,3,5,2,5,8]^T, X^{β}=[1,3,3,3,1,2,3,1,3]^T.

Konfiguracja o ocenie P_4=(9, 151) cechuje się parametrami: Z_{suma}=1322 [s], $\Delta_{max} = 80$ [s] dla $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$, Θ=54,223 [TFLOPS], $\Xi(x)$=11 616 [USD], υ=264. Specyfikacja konfiguracji to: X^{α}=[3,3,3,9,5,7,6,1,2,9,5,2,6,7,8,4,7,8,1,5,4,4,3,8]^T, X^{β}=[1,3,3,6,1,2,3,1,3]^T.

Algorytm MOHS3 porównano z innymi wielokryterialnymi wersjami algorytmów harmonicznych. MOHS1 cechuje się tym, że procedura nadawania rang opiera się na algorytmie *Goldberga*. Ponadto rozwiązania z rangą graniczną dobierane są w sposób losowy. Natomiast w MOHS2 oprócz procedury *Goldberga* konfiguracje z rangą graniczną dobierane są wg najmniejszej wartości funkcji zagęszczenia *GCD*. Z kolei w MOHS4 zastosowano procedurę nadawania rang *Fonseci-Fleminga* oraz selekcję rozwiązań z rangą graniczną wg najmniejszej wartości funkcji zagęszczenia *GCD*.

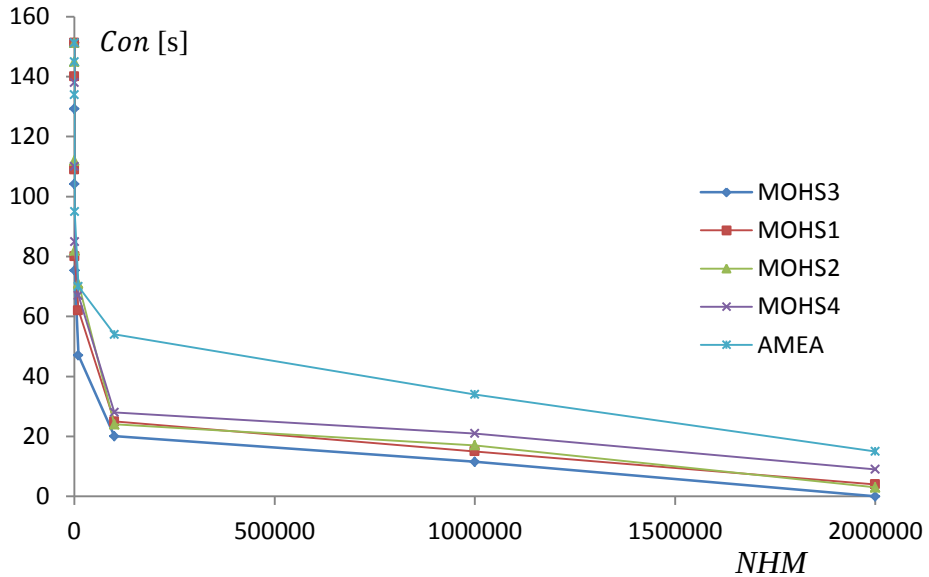
Niech algorytm $MOHS_m$ wyznacza reprezentację $\boldsymbol{\Omega}_m^*$ ocen *Pareto*-suboptymalnych. Aby porównać jakość wyznaczonych rozwiązań za pomocą różnych algorytmów w odniesieniu do znanej reprezentacji ocen *Pareto*-suboptymalnych $\{P_1, P_2, P_3, P_4\}$, wprowadza się Con_m - stopień osiągnięcia frontu *Pareto* przez *m*-ty algorytm, jak niżej:

$$Con_m(NHM) = \sum_{l=1}^{4} \min_{\omega \in \boldsymbol{\Omega}_m^*} d_l(P_l, \omega),\ m = \overline{1,4}, \tag{D.1}$$

gdzie:

$d_l(P_l, \omega)$ - odległość *Euklidesa* między *l*-tą oceną *Pareto*-suboptymalną a oceną niezdominowaną wyznaczoną przez *m*-ty algorytm w iteracji nr *NHM*.

Na rysunku 49 przedstawiono zbieżność czterech algorytmów klasy MOHS. Najszybszą zbieżność uzyskano dla algorytmu MOHS3, za pomocą którego po 2×10^6 aktualizacjach tablicy harmonicznej osiągnięto brzeg *Pareto*.



Rys. 49. Zbieżność wybranych wersji wielokryterialnych algorytmów harmonicznych
Źródło: opracowanie własne

Instancja nr 2 umożliwia porównanie jakości konfiguracji gridu wyznaczonych za pomocą algorytmu MOHS3 z wynikami uzyskanymi za pomocą innych metaheurystyk. Wprawdzie do najpopularniejszych metaheurystyk wielokryterialnych należy algorytm ewolucyjny NSGA-II, to jednak adaptacyjny algorytm ewolucyjne AMEA umożliwia wyznaczanie rozwiązań wyższej jakości dla zagadnień optymalizacji konfiguracji gridu [33]. W algorytmie AMEA można stosunkowo prosto zrealizować kodowanie całkowitoliczbowe konfiguracji, co znacząco zawęża przestrzeń przeszukiwań. Takiej opcji nie ma algorytm NSGA-II, co powoduje, że w zadanym czasie umożliwia wyznaczenie wyników niższej jakości niż AMEA dla tej klasy problemów polioptymalizacji.

Z drugiej strony dla wielu instancji analizowanych w literaturze przedmiotu algorytm NSGA-II wyznaczył wyniki wyższej jakości niż algorytm przeszukiwania tabu MOTS [366] czy wielokryterialna wersja algorytmu mrówkowego MOACA [79].

Na rysunku 49 zaprezentowano zbieżność wyników uzyskanych za pomocą algorytmu ewolucyjnego AMEA na tle zbieżności algorytmów harmonicznych. Rozmiar populacji L wynosił 20, tak jak wielkość pamięci harmonicznej, a *NHM* odpowiada numerowi generowanej populacji. Tempo mutacji rosło wraz ze wzrostem *NHM* w następujący sposób:

$$p_m = \frac{NHM}{L \times M \times T_{\max}}, \tag{D.2}$$

gdzie:

M – liczba zmiennych decyzyjnych przy kodowaniu całkowitoliczbowym,

$T_{\max}$ – maksymalna liczba populacji.

Natomiast tempo krzyżowania malało wraz ze wzrostem *NHM*, jak niżej:

$$p_c = e^{-NHM/T_{\max}} - 0{,}1 \tag{D.3}$$

Algorytm ewolucyjny AMEA dla instancji nr 2 cechuje się wolniejszą zbieżnością do frontu *Pareto* niż algorytm harmoniczny MOHS3.

Eksperyment numeryczny nr 3

Celem eksperymentu jest wyznaczenie konfiguracji kompromisowej dla zagadnienia (3.5.16) za pomocą algorytmu harmonicznego CHS1. Rozważa się możliwość rozbudowy gridu *Comcute* w laboratorium Politechniki Gdańskiej przy niewielkich nakładach finansowych. Warstwa pośrednicząca gridu *Comcute* składa się 6 modułów, w tym 2 klasy *W* i 4 klasy *S*. Moduły rozmieszczono w 6 węzłach, w których

zainstalowano dwuprocesorowe serwery DELL E5640 z pamięcią dyskową 3 [TB]. Ponadto trzy serwery w szafie typu *rack* 19" 1U dedykowane są do obsługi: aplikacji, bazy danych, DNS, *XenServer*, prototypu *Comcute* w środowisku .NET, a także do modyfikacji oprogramowania warstwy pośredniczącej.

Liczbę węzłów można zwiększyć z 9 do co najwyżej 15 ze względu na ograniczoną infrastrukturę gridową w laboratorium. Znaczące zwiększenie liczby węzłów jest już trudniejsze bez dodatkowych nakładów związanych z pozyskaniem i zabezpieczeniem nowych pomieszczeń lub też wykorzystaniem zasobów w chmurze obliczeniowej. Ponadto należy uwzględnić większe zużycie energii elektrycznej.

Zakłada się, że $\xi_{\max}$ - limit środków finansowych przeznaczonych na rozbudowę gridu wynosi 350 000 [zł], a $\varepsilon_{\max}$ - ograniczenie na pobór mocy energii elektrycznej 25 [kW]. Przy takich założeniach oraz na podstawie kosztów zakupu serwerów (tab. 10) nie można zakupić 15 bardzo wydajnych serwerów klasy HP *ProLiant*.

Celem optymalnego wykorzystania zasobów w gridzie jest minimalizacja obciążeń w warstwie pośredniczącej podczas weryfikacji hipotezy *Collatza*. Można rozpatrywać inne scenariusze obliczeń, uwzględniając różnorodne kombinacje 9 zadań użytkowników lub minimalizując obciążenia związane z wykonywaniem aplikacji. Jednakże zagadnienie *Collatza* jest najbardziej intensywnie użytkowanym zadaniem.

Aby zwiększyć wydajność systemu, planuje się wzrost liczby modułów z 6 do 45, w tym 15 modułów klasy *W*, a 30 - klasy *S*. Dokonano pomiaru skumulowanego czasu wykonania modułu *W* na serwerze klasy β_{11} podczas weryfikacji hipotezy *Collatza*. Weryfikacja hipotezy trwała 27 minut, podczas których 64 klienty wykonywane były na 16 komputerach. Skumulowany czas wykonania modułu *W* na β_{11} wyniósł 320,288 [s]. Natomiast skumulowany czas wykonania modułu *S* na β_{11} to 306,830 [s]. Na podstawie wyników testu *CPU Mark* oszacowano pozostałe elementy macierzy $\boldsymbol{T}$ (tab. 10) [409-411]. Skumulowany czas komunikacji z *W* do *S* wyniósł 27,602 [s], a skumulowany czas komunikacji z *S* do *W* – 337,926 [s]. Na tej podstawie wyznaczono macierz skumulowanych czasów komunikacji $\boldsymbol{\tau}$.

Moduł typu *W* wymaga 4 [GB] pamięci RAM i 5 [GB] HD, a moduł typu *S* - 1,5 [GB] RAM i 0,5 [GB] HD. Zapotrzebowanie *W* na pamięć RAM dla 100 000 paczek to 4 [GB]. Maszyna wirtualna *Javy* preferuje rezerwowanie dodatkowej pamięci niż użycie *garbage collectora*. Natomiast wymaganie na pamięć dyskową zależy od rozmiaru bazy danych, w której zapisane są paczki danych dla zadania.

Dla zadania *S* maszyna *Javy* rezerwuje 512 [MB] pamięci RAM, a górny limit ustalony jest na poziomie 1,5 [GB], gdyż niemal wszystkie dane przechowywane są w pamięci RAM. W przypadku HD wymagania dla *S* można oszacować na poziomie 0,5 GB, co obejmuje: maszynę *Javy*, serwer aplikacji, jego *cache* i logi.

Tabela 10. Parametry wybranych rodzajów serwerów dostępnych na rynku

Nr j	Nazwa serwera βj	Specyfikacja procesorów	$\tilde{d}_{j1}$ - wielkość RAM [GB]	$\tilde{d}_{j2}$ - wielkość dysku [TB]	ε_j - moc zasilania [W]	ξ_j - koszt zakupu [zł]	ϑ_j -wydajność**	t_{vj} $v=\overline{1,15}$	t_{vj} $v=\overline{16,45}$
1.	Infotronik ATX i5-4430	Intel 4-Core i5-4430 3,0GHz	8	2	400	4628	6284	577,171	552,919
2.	Infotronik ATX i7-4790	Intel 4-Core i7-4790 3,6GHz	16	2	550	4919	10110	358,748	343,674
3.	BizServer E5-2660v2	2xIntel 6-core Xeon E5-2660v2 2,20GHz	48	24	1280	11751	23922	151,615	145,245
4.	DELL R520	2xIntel 6-core Xeon E5-2420v2 2,20GHz	32	4	1500	12596	17194	210,942	202,079
5.	Fujitsu Primergy RX300S8	2xIntel 6-core Xeon E5-2620v2 2,10GHz	32	1,6	900	20218	17384	208,637	199,870
6.	IBM x3650 M4	2xIntel 6-core Xeon E5-2620v2 2,10GHz	48	4,6	675	21350	17384	208,637	199,870
7.	HP ProLiant DL380p gen8	2xIntel 10-core Xeon E5-2690v2 2.90 GHz	32	2	1500	33324	28720	126,286	120,980
8.	HP ProLiant E5-2695	2xIntel 14-core Xeon E5-2695v3 2.3 GHz	32	2	1500	37338	42246	85,853	82,245
9.	HP ProLiant E5-2697	2xIntel 14-core Xeon E5-2697v3 2.6 GHz	32	2	1500	39951	45112	80,399	77,020
10.	HP ProLiant E5-2699	2xIntel 18-core Xeon E5-2699v3 2,3 GHz	32	2	1500	47075	50384	71,986	68,961
11.	DELL E5640 v1	2xIntel 4-core Xeon E-5640 2,66GHz	24	3	900	0*	11324	320,288	306,830
12.	DELL E5640 v2	2xIntel 4-core Xeon E-5640 2,66GHz	48	3	900	0*	11324	320,288	306,830

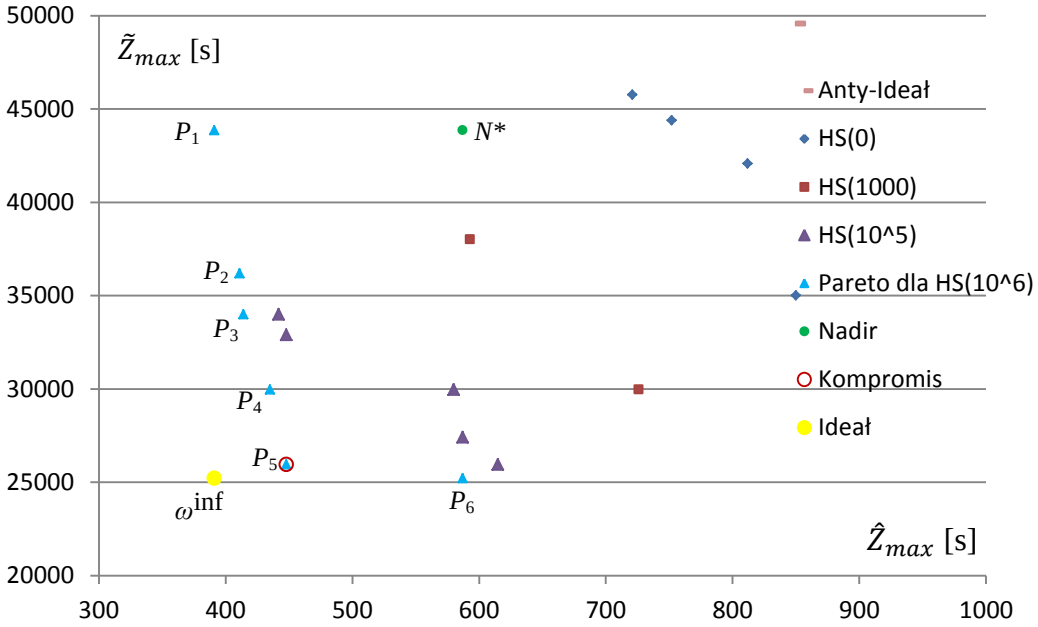
* serwery zainstalowane w gridzie *Comcute* (4 szt. DELL R520 v1 oraz 5 szt. R520 v2)
** wg testu *CPU Mark*, http://www.passmark.com/, dostęp: 19.01.2015
Źródło: opracowanie własne na podstawie [409-411]

W instancji zagadnienia optymalizacji dwukryterialnej (3.5.16) preferuje się jednoczesną minimalizację $\hat{Z}_{max}$ oraz $\tilde{Z}_{max}$. Dane wejściowe zapisano w pliku *benchmarkNr3.txt* [396]. W porównaniu do instancji nr 2 liczba modułów jest zwiększona z 24 do 45, liczba węzłów z 9 do 15, a liczba rodzajów komputerów z 10 do 12. Instancja cechuje się 855 zero-jedynkowymi zmiennymi decyzyjnymi, a binarna przestrzeń przeszukiwań zawiera $2{,}4*10^{257}$ elementów. Liczba zmiennych kodowanych całkowitoliczbowo wynosi 60, a liczba tak zakodowanych konfiguracji - $1{,}3*10^{69}$.

Obliczenia za pomocą metody zmodyfikowanego przeglądu na superkomputerze *Tianhe-2* wymagają ok. 10^{45} lat.

Do rozwiązania problemu zastosowano algorytm CHS1 z następującymi parametrami: *HMCR*=0.9, *PAR=0.5*, *BW1=3*, *BW2=4*, *HMS*=30 oraz *HMS2*=20. Czas obliczeń dla 1 000 000 aktualizacji pamięci harmonicznej HM wyniósł ok. 5 godzin na komputerze klasy *PC/Windows 7/Intel Core i7-2670 QM 2,2 GHz.*

Na rysunku 50 przedstawiono migrację ocen do brzegu *Pareto*. W pierwszej fazie obliczeń za pomocą CHS1 wykorzystano algorytm MOHS3 do wyznaczenia reprezentacji konfiguracji *Pareto*-suboptymalnych $\{P_1,P_2,P_3,P_4,P_5,P_6\}$.



Rys. 50. Wyznaczenie oceny kompromisowej dla instancji nr 3 za pomocą algorytmu harmonicznego CHS1
Źródło: opracowanie własne

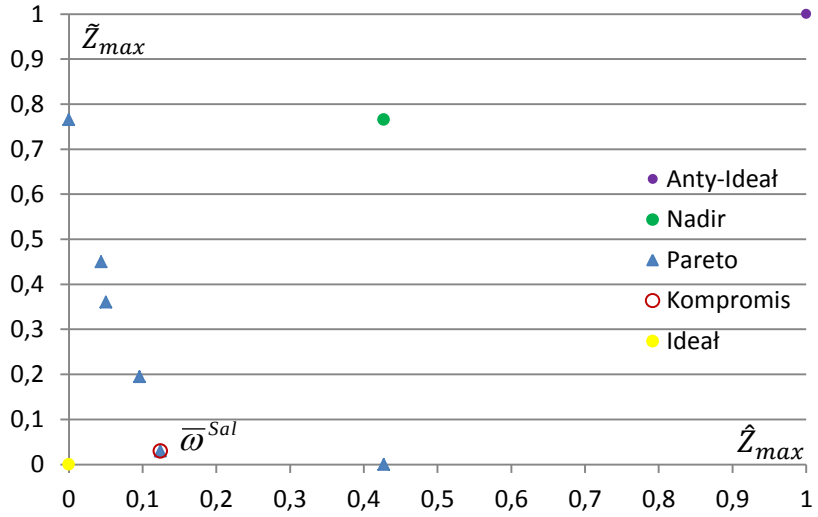
W drugiej fazie obliczeń za pomocą algorytmu CHS1 wyznaczono współrzędne: punktu idealnego ω^{inf}=(411; 25 221) [s], punktu nadir N^*=(587; 43 863) [s] i punktu antyidealnego ω^{sup}=(850; 49 577) [s]. Ponadto dokonano normalizacji przedziałowej współrzędnych reprezentacji konfiguracji niezdominowanych (rys. 51).

Następnie wyznaczono konfigurację *Salukwadze* (kompromisową dla parametru *p*=2) cechującą się najmniejszą odległością od punktu idealnego w znormalizowanej przestrzeni kryterialnej. Odległość $\varsigma_2(\overline{\omega}^{Sal}) = \inf_{\omega \in \boldsymbol{\Omega}} \xi_2(\overline{\omega})$ wynosi 0,127. W przestrzeni znormalizowanej $\overline{\boldsymbol{\Omega}}$ zachodzi $\overline{\omega}^{Sal}$=(0,124; 0,030), a w przestrzeni wielokryterialnej $\boldsymbol{\Omega}$ ocena kompromisowa ω^{Sal} to (448; 25 952) [s].

Konfiguracja x^{Sal} cechuje się parametrami: Z_{suma}=318 001 [s], Δ_{max} = 13 200 [s] dla $\rho_1 = 0{,}5$ $\rho_2 = 0{,}5$, Θ=453 450 w jednostkach CPU *Mark*, Ξ =325 875 [zł], υ=856 oraz E=19 240 [W]. Spełnione są także wymagania na pamięć RAM i HD.

Zakodowana całkowitoliczbowo konfiguracja jest następująca:

X^{α}=[10,7,12,4,1,11,12,10,5,8,3,14,7,13,11,9,8,5,6,6,1,3,14,10,11,6,13,2,4,5,9,13,9,7,9,4,6,2,2,5,3,1,4,12,2,]^T, X^{β}=[3,8,3,9,8,9,8,3,9,3,3,3,3,3,0]^T.



Rys. 51. Wyznaczenie oceny kompromisowej w znormalizowanej przestrzeni kompromisowej za pomocą algorytmu CHS1
Źródło: opracowanie własne

W tabeli 11 przedstawiono specyfikację konfiguracji *Salukwadze* dla rozważanej sytuacji decyzyjnej. W skład węzłów wchodzi: 8 serwerów *BizServer* E5-2660v2 oraz po 3 serwery HP *ProLiant* E5-2695 i HP *ProLiant* E5-2697. Natomiast w węźle 15 nie ma potrzeby instalowania komputera.

Tabela 11. Specyfikacja konfiguracji *Salukwadze* dla instancji nr 3

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
j	3	8	3	9	8	9	8	3	9	3	3	3	3	3
Nr W	5		11	4	9		2, 13	10		1,8	6, 15	3.7	14	12
Nr S	6,27	13,23,24,30	7,26	14,21,28	3,15,25	4,5,11,22	19	2	1,16,18,20	9	10	29	12,17	8

Źródło: opracowanie własne

Warto porównać zaproponowaną konfigurację kompromisową z konfiguracją obecnie eksploatowanego gridu *Comcute*. W wyniku zwiększenia liczby modułów do 45 w warstwie pośredniczącej, należy oczekiwać, że 480 zadań klienckich w rozbudowanej konfiguracji będzie obsługiwanych z taką samą intensywnością, jak 64 zadania w obecnym gridzie. Ponadto wydajność wg testu CPU *Mark* będzie zwiększona co najmniej 4,5-krotnie przy spełnieniu istotnych wymagań, w tym nieprzekroczenia limitów finansowych i energetycznych.