Załącznik 3

Dr inż. Mariusz Uchroński Wrocław, 7.12.2021 r.
Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji
Katedra Automatyki, Mechatroniki i Systemów Sterowania
mgr inż. 2008, dr inż. 2014

Efektywne metody rozwiązywania problemów optymalizacji dyskretnej

Autoreferat

Spis treści

Spis treści **1**

1 Imię i nazwisko **2**

2 Posiadane dyplomy, stopnie naukowe **2**

3 Informacja o dotychczasowym zatrudnieniu w jednostkach naukowych **3**

4 Omówienie osiągnięć, o których mowa w art. 219 ust. 1 pkt. 2 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2020 r. poz. 85 z późn. zm.) **3**
4.1 Tytuł osiągnięcia naukowego 3
4.2 Publikacje wchodzące w skład osiągnięcia naukowego 3

5 Informacja o wykazywaniu się istotną aktywnością naukową albo artystyczną realizowaną w więcej niż jednej uczelni, instytucji naukowej lub instytucji kultury, w szczególności zagranicznej **5**

6 Informacja o osiągnięciach dydaktycznych, organizacyjnych oraz popularyzujących naukę **7**
6.1 Informacja o osiągnięciach dydaktycznych 7
6.2 Informacja o osiągnięciach organizacyjnych 9
6.3 Informacja o osiągnięciach popularyzujących naukę 10

7 Inne informacje dotyczące kariery zawodowej **11**
7.1 Autorstwo lub współautorstwo publikacji naukowych w czasopismach znajdujących się w bazie Journal Citation Reports (JCR) opublikowane po uzyskaniu stopnia doktora 14

7.2 Prace w czasopismach spoza listy JCR opublikowane po uzyskaniem stopnia doktora 14
7.3 Otrzymane nagrody i wyróżnienia 15
7.4 Recenzje artykułów 15
7.5 Projekty badawcze 16
7.6 Współpraca z przemysłem 16

8 Opis działalności naukowo badawczej 16
8.1 Obliczenia równoległe w optymalizacji dyskretnej 19
8.2 Metody przyspieszania przeszukiwania przestrzeni rozwiązań 23
8.2.1 Kryteria sumokosztowe 25
8.2.2 Problemy z minimalizacją czasu cyklu 32
8.3 Równoległe implementacje metod przyspieszania obliczeń 41
8.3.1 Zrównoleglenie algorytmu przeszukiwania z tabu 41
8.3.2 Równoległe wyznaczanie wzorców w rozwiązywaniu cyklicznego problemu przepływowego z przezbrojeniami 45
8.4 Badanie rozkładu minimów lokalnych w przestrzeni rozwiązań 51
8.5 Optymalizacja z niepewnymi parametrami 57
8.6 Chemia obliczeniowa – dynamika molekularna 63
8.7 Cyberbezpieczeństwo – wykrywanie anomalii w ruchu sieciowym 63
8.8 Inżynieria środowiska – badanie jakości powietrza wewnętrznego 64

9 Analiza cytowań 65

Bibliografia 65

---

1 Imię i nazwisko

Mariusz Uchroński

2 Posiadane dyplomy, stopnie naukowe

— z podaniem podmiotu nadającego stopień, roku ich uzyskania oraz tytułu rozprawy doktorskiej

- **Doktor nauk technicznych**, Wydział Elektroniki Politechniki Wrocławskiej, dyscyplina: Automatyka i Robotyka, tytuł rozprawy doktorskiej: „Równoległe i rozproszone algorytmy optymalizacji w elastycznych systemach wytwarzania", promotor: prof. dr hab. inż. Czesław Smutnicki, (2014),
- **Magister inżynier**, Wydział Elektroniki Politechniki Wrocławskiej, kierunek: Elektronika i Telekomunikacja, specjalność: Zastosowania Inżynierii Komputerowej w Technice, tytuł pracy magisterskiej: „Harmonogramowanie w systemach przepływowych z uwzględnieniem możliwości równoczesnego wykonywania wielu zadań", promotor: prof. dr hab. Wojciech Bożejko, (2008).

3 Informacja o dotychczasowym zatrudnieniu w jednostkach naukowych

- 2020 – obecnie, Adiunkt, Katedra Automatyki, Mechatroniki i Systemów Sterowania, Politechnika Wrocławska,
- 2016 – 2020, Asystent, Katedra Automatyki, Mechatroniki i Systemów Sterowania, Politechnika Wrocławska,
- 2015 – obecnie, Specjalista, Wrocławskie Centrum Sieciowo-Superkomputerowe, Politechnika Wrocławska,
- 2009 – 2015, Starszy asystent informatyczny, Wrocławskie Centrum Sieciowo-Superkomputerowe, Politechnika Wrocławska,
- 2019 – 2021, Poznańskie Centrum Superkomputerowo-Sieciowe, Instytut Chemii Bioorganicznej Polskiej Akademii Nauk w Poznaniu.

4 Omówienie osiągnięć, o których mowa w art. 219 ust. 1 pkt. 2 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2020 r. poz. 85 z późn. zm.)

Omówienie to winno dotyczyć merytorycznego ujęcia przedmiotowych osiągnięć, jak i w sposób precyzyjny określać indywidualny wkład w ich powstanie, w przypadku, gdy dane osiągnięcie jest dziełem współautorskim, z uwzględnieniem możliwości wskazywania dorobku z okresu całej kariery zawodowej.

4.1 Tytuł osiągnięcia naukowego

Efektywne metody rozwiązywania trudnych problemów optymalizacji dyskretnej

4.2 Publikacje wchodzące w skład osiągnięcia naukowego

Cykl powiązanych tematycznie artykułów naukowych opublikowanych w czasopismach naukowych lub w recenzowanych materiałach z konferencji międzynarodowych, które w roku opublikowania artykułu w ostatecznej formie były ujęte w wykazie sporządzonym zgodnie z przepisami wydanymi na podstawie art. 267 ust. 2 pkt 2 lit. b, pt.

[C1] **Uchroński, M.** Parallel algorithm with blocks for a single-machine total weighted tardiness scheduling problem. Applied Sciences. 2021, vol. 11, nr 5, art. 2069, s. 1-17, https://doi.org/10.3390/app11052069
IF 02.679 (2021), MNiSW 2019-2021: 70

[C2] Bożejko W., **Uchroński M.**[1], Wodecki M., Blocks for two-machines total weighted tardiness flow shop scheduling problem. Bulletin of the Polish Academy of Sciences. Technical Sciences. 2020, vol. 68, nr 1, s. 31-41.
IF 01.662 (2020), MNiSW 2019-2021: 100

[C3] Bożejko W., **Uchroński M.**, Wodecki M., Block approach to the cyclic flow shop scheduling. Computers and Industrial Engineering. 2015, vol. 81, s. 158-166.
IF 02.086 (2015), MNiSW 2013-2018: 40

[C4] Bożejko W., **Uchroński M.**, Wodecki M., Parallel metaheuristics for the cyclic flow shop scheduling problem. Computers and Industrial Engineering. 2016, vol. 95, s. 156-163.
IF 02.623 (2016); MNiSW 2013-2018: 40

[C5] Bożejko, W., Smutnicki, C., **Uchroński, M.**, Wodecki, M., Big valley in scheduling problems landscape - Metaheuristics with reduced searching area. 2017 22nd International Conference on Methods and Models in Automation and Robotics, MMAR 2017, 2017, pp. 458-462, 8046871
MNiSW 2017: 15 – Web of Science

[C6] Bożejko, W., **Uchroński, M.**, Wodecki, M., Parallel tabu search algorithm with uncertain data for the flexible job shop problem. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2016, 9693, pp. 419-428
MNiSW 2016: 15 – Web of Science

Dorobek naukowy

1. Jestem autorem lub współautorem 52 recenzowanych publikacji, z czego 12 posiadających wskaźnik Impact Factor, w tym 1 samodzielna oraz 11 wieloautorskich. Po uzyskaniu stopnia doktora (w roku 2014) 8, w tym jedna samodzielna.

2. Liczba cytowań wg. Clarivate Web of Knowledge Core Collection wynosi 140, w tym 128 bez autocytowań, Według Scopus 226 (202 bez autocytowań), a Google Scholar 321.

3. Indeks Hirscha (według Web of Knowledge): 6 – stan na 29.11.2021 r. Scopus 7, a Google Scholar 8.

4. Jestem recenzentem w czasopismach naukowych z dziedziny optymalizacji, algorytmiki i zastosowań matematyki: *Open Computer Science, International Journal of Mathematics and Mathematical Sciences, Mathematics, Algorithms.*

[1] We wszystkich pracach autorzy występują w porządku alfabetycznym

5 Informacja o wykazywaniu się istotną aktywnością naukową albo artystyczną realizowaną w więcej niż jednej uczelni, instytucji naukowej lub instytucji kultury, w szczególności zagranicznej

Od 2014 roku jestem zaangażowany w realizację projektu PRACE (*ang. Partnership for Advanced Computing in Europe*), którego celem jest umożliwienie interdyscypinarnych badań naukowych i inżynieryjnych z korzyścią dla społeczeństwa Europy. Cel ten realizowany jest przez udostępnienie środowisku naukowemu oraz inżynierom w Europie światowej klasy infrastruktury i usług HPC (*ang. High Performance Computing*) o wydajności obliczeniowej rzędu PetaFlops i większej. W ramach udziału w piątej fazie implementacji projektu PRACE (tzw. PRACE-5IP) nawiązałem współpracę z Europejskim Centrum Doskonałości E-CAM.

E-CAM jest partnerstwem 16 węzłów CECAM (*fr. Centre Européen de Calcul Atomique et Moléculaire*), 3 centrów PRACE, 16 partnerów z przemysłu oraz centrum HPC dla przemysłu. Partnerzy E-CAM to m.in.: University College Dublin, Free University of Berlin, Sapienza University of Rome, Technical University of Vienna, University of Cambridge, University of Amsterdam, Jülich Supercomputing Centre, CSC-IT Centre for Science Ltd, Irish Centre for High-End Computing.

E-CAM skupia swoją działalność wokół czterech obszarów naukowych związanych z obliczeniami HPC: klasyczna dynamika molekularna, struktura elektroniczna, dynamika kwantowa oraz modelowanie wielkoskalowe. Moja współpraca z E-CAM w ramach projektu PRACE-5IP polegała na udziale w realizacji następujących zadań badawczo–rozwojowych:

- Task Scheduling Library for Optimising Time-Scale Molecular Dynamics Simulations. Celem realizowanych prac badawczo–rozwojowych było utworzenie narzędzia badawczego (w formie algorytmicznej biblioteki programistycznej) pozwalającego na efektywne wykonywanie obliczeń klasy HTC (*High Throughput Computing*) – paradygmat obliczeniowy, który koncentruje się na wydajnej realizacji dużej liczby luźno powiązanych zadań. Do zarządzania tego typu obliczeniami wykonane zostało narzędzie do harmonogramowania zadań. Główne funkcje powstałego w ramach prac narzędzia obejmują: definicje zadania obliczeniowego, szeregowanie zadań obliczeniowych (obsługiwane w Python) oraz wykonywanie obliczeń (realizowane w warstwie MPI). Powstała biblioteka jest elastyczna, skalowalna, wydajna i adaptacyjna. Umożliwia jednoczesne wykorzystanie CPU, KNL i GPU oraz dynamiczne dostosowanie wykorzystania tych zasobów w oparciu o wymagania dotyczące żądanych zasobów niezbędnych do wykonywania zadań. Ostateczna skalowalność i możliwości sprzętowe rozwiązania są podyktowane przez charakterystykę skalowalności samych zadań (jeśli na zadanie z procesorami graficznymi przypada 100 węzłów, wówczas biblioteka może wykonać n z nich jednocześnie, w zależności od dostępności zasobów).

- Intelligent HTC for Committor Analysis. Analiza Committor jest potężnym, ale kosztownym obliczeniowo narzędziem do badania mechanizmów reakcji w złożonych systemach w obszarze dynamiki molekularnej. Committor może być również

używany do generowania początkowych trajektorii dla próbkowania ścieżki przejścia, co jest mniej kosztowną techniką do badania mechanizmów reakcji. Głównym celem prowadzonych prac badawczo-rozwojowych było ułatwienie wdrożenia analizy committor w aplikacji OpenPathSampling[2], która umożliwia przenoszenie wydajności w całym zakres sprzętu HPC i witryn hostingowych. Zrealizowane to zostało za pomocą sprzętowych silników dynamiki molekularnej w OpenPathSampling w połączeniu z niestandardowym rozszerzeniem biblioteki do platformy analizy danych Dask[3], która pozwala na wykonywanie zadań z obsługą MPI w sterowalnym przepływie wykonywania obliczeń o wysokiej przepustowości. Opracowane oprogramowanie może zostać wykorzystywane do generowania wstępnych trajektorii do badania zmian konformacyjnych w głównej proteazie wirusa SARS-CoV-2, który powoduje COVID-19. Ta zmiana konformacyjna może regulować dostępność miejsca aktywnego głównej proteazy, a lepsze zrozumienie jego mechanizmu może pomóc w projektowaniu leków.

Wynikiem współpracy z E-CAM w ramach projektu PRACE-5IP są, między innymi, publikacje:

1. Alan O'Cais, David Swenson, **Mariusz Uchroński**, Adam Włodarczyk, Task Scheduling Library for Optimising Time-Scale Molecular Dynamics Simulations, 2019, https://doi.org/10.5281/zenodo.3527642

2. Miłosz Białczak, Alan O'Cais, David Swenson, **Mariusz Uchroński**, Adam Włodarczyk, Intelligent HTC for Committor Analysis, 2020, http://doi.org/10.5281/zenodo.4382017

Ponadto, współpraca z E-CAM w ramach projektu PRACE-5IP zaowcowała udziałem jako ekspert w międzynarodowym spotkaniu „Intelligent HTC for scientific applications" organizowanym przez Politecnico di Torino w dniach 1-4 lipca 2019.

W latach 2019–2020 współpracowałem z Poznańskim Centrum Superkomputerowo-Sieciowym przy Instytucie Chemii Bioorganicznej Polskiej Akademii Nauk w Poznaniu. W ramach współpracy zajmowałem się automatyzacją i optymalizacją wykonywania obliczeń na Komputerach Dużej Mocy (KDM) w zagadnieniach z dziedziny chemii – parametryzacji reaktywnego pola siłowego poprzez dopasowanie sił – realizowanych w ramach międzynarodowego projektu PRACE2.0. Moja współpraca obejmuje także optymalizację narzędzi do parametryzacji reaktywnego pola siłowego w obliczeniach z dziedziny chemii (prace w ramach projektu PRACE AISBL 2017) oraz automatyzację analizy danych z pomiarów jakości powietrza wewnętrznego z wykorzystaniem metod obliczeniowych.

Ponadto, realizowałem aktywność naukową w formie wspólnych publikacji i projektów we współpracy z następującymi naukowcami reprezentującymi jednostki badawcze z kraju i z zagranicy:

1. Dr Zenon Chaczko, University of Technology Sydney, Australia[4],

[2]http://openpathsampling.org/

[3]https://dask.org/

[4]Bożejko, Chaczko Z., Uchroński M., Wodecki M., Parallel patterns determination in solving cyclic flow shop problem with setups, in: Special issue on Discrete Systems: Theory and Applications (Bozejko W., Wodecki M., Eds.), Archives of Control Sciences, Vol. 27, No. 2, 2017, 183-196.

2. Dr inż. Paweł Rajba, Instytut Informatyki Uniwersytetu Wrocławskiego[5],
3. Dr inż. Piotr Nadybski, Wyższa Szkoła Zawodowa w Legnicy[6],
4. Prof. dr hab. inż. Adam Zofka, Instytut Badawczy Dróg i Mostów, Warszawa[7].

6 Informacja o osiągnięciach dydaktycznych, organizacyjnych oraz popularyzujących naukę

6.1 Informacja o osiągnięciach dydaktycznych

- Prowadzone zajęcia dydaktyczne:
  1. Podstawy programowania (wykład),
  2. Platformy programistyczne NET i Java (wykład i laboratorium),
  3. Wykład monograficzny na specjalności Systemy informatyczne w automatyce (ASI),
  4. Administracja sieciowymi systemami operacyjnymi (laboratorium),
  5. Systemy operacyjne 2 (laboratorium),
  6. SCR - sieci komputerowe (laboratorium),
  7. SCR - systemy operacyjne (laboratorium),
  8. Technologie informacyjne (laboratorium).
- Przygotowanie, dla nowo tworzonego kierunku studiów *Informatyczne Systemy Automatyki*, kart przedmiotów:
  1. Platformy programistyczne .Net i Java,
  2. Paradygmaty programowania obiektowego,
  3. Bazy danych,
  4. Technologie WWW,
  5. Obliczenia superkomputerowe,
  6. Zaawansowane technologie programowania,
  7. Obliczenia wysokiej wydajności.
- Przygotowanie materiałów dydaktycznych dla specjalności Przemysł 4.0 w ramach realizacji projektu "ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej".

[5]Bożejko W., Rajba P., Uchroński M., Wodecki M., A Job Shop Scheduling Problem with Due Dates Under Conditions of Uncertainty, Proceedings of the International Conference on Computational Science, in: M. Paszynski et al. (Eds.): ICCS 2021, LNCS 12742, Springer, pp. 198-205 (2021).

[6]Bożejko W., Nadybski P., Uchroński M., Wodecki M., Cykliczny system produkcyjny z dwumaszynowymi gniazdami, w: Innowacje w zarządzaniu i inżynierii produkcji (red. R. Knosala), Oficyna Wydawnicza Polskiego Towarzystwa Zarządzania Produkcją, Opole 2017, ISBN 978-83-941281-1-1, 512-523.

[7]Udział w projekcie GOSPOSTRATEG – NCBR, 2020–2021 (Wykonawca)

- **Promotor pomocniczy** obronionych doktoratów:

  1. Andrzej Gnatowski, Właściwości i algorytmy dla systemów wytwarzania z maszynami równoległymi, promotor: prof. dr hab. Wojciech Bożejko, dyscyplina przewodu: automatyka i robotyka, 2019.
  2. Piotr Nadybski, Algorytmy szeregowania zadań i rozdziału zasobów w systemach wieloprocesorowych, promotor: prof. dr hab. Mieczysław Wodecki, dyscyplina przewodu: informatyka, 2021.

- Promotor prac magisterskich:

  1. Algorytm dostosowywania trasy przejazdu robota mobilnego w środowisku przemysłowym, 2021.
  2. Sterowanie platformą mobilną za pomocą systemu wizyjnego, 2020.
  3. Comparative analysis of tools for building web applications backend, 2020.
  4. Rozpoznawanie wzorców w szeregach czasowych temperatury oraz stężenia dwutlenku węgla w pomieszczeniach biurowych, 2018.

- Promotor prac inżynierskich:

  1. Wieloosobowa gra strategiczna z elementami sztucznej inteligencji, 2020.
  2. Aplikacja mobilna - planowanie jadłospisu na podstawie przemyślanych zakupów, 2020.
  3. Aplikacja webowa do zarządzania ligą sportową, 2020.
  4. Wykorzystanie bazy danych do przechowywania, analizowania oraz archiwizowania awarii form odlewniczych oraz rdzennic, 2020.
  5. Aplikacja webowa - katalog gier planszowych z rankingami graczy, 2020.
  6. System kontroli dostępu do szafy z narzędziami, 2019.
  7. Interpreter Lisp, 2019.
  8. Aplikacja mobilna - trener personalny, 2019.
  9. Aplikacja mobilna - krokomierz, 2019.
  10. Mobilny komunikator internetowy, 2019.
  11. Edytor graficznego interfejsu użytkownika, 2019.
  12. Aplikacja webowa do zarządzania zasobami w chmurze, 2019.
  13. Przeszukiwalne repozytorium dokumentów, 2019.
  14. Gra strategiczna z elementami sztucznej inteligencji, 2019
  15. Aplikacja internetowa - serwis ogłoszeniowy, 2019.
  16. Aplikacja webowa do zarządzania projektem w metodologii Scrum, 2019.
  17. Aplikacja webowa obsługująca kino, 2018.
  18. Deskorolka elektryczna ze sterowaniem opartym na Arduino, 2018.
  19. Monitoring czasu pracy kierowcy, 2018.

20. Gra wieloosobowa z elementami sztucznej inteligencji, 2018.
21. Aplikacja mobilna wspomagająca planowanie podróży rowerem miejskim, 2018.
22. Aplikacja mobilna - detektor okolicznych atrakcji turystycznych, 2018.
23. Symulacja procesu produkcyjnego w środowisku AutoMod, 2018
24. Aplikacja mobilna dedykowana przyszłym mamom, 2018.
25. Aplikacja wspomagająca układanie planu zajęć, 2017.
26. Aplikacja do zarządzania zadaniami w projekcie na platformę Android, 2017.
27. Akwizycja danych pomiarowych o jakości powietrza wewnętrznego, 2017.
28. Równoległe algorytmy przetwarzania danych o jakości powietrza wewnętrznego na potrzeby systemów automatyki budynkowej, 2016.

6.2 Informacja o osiągnięciach organizacyjnych

- Członek komisji egzaminacyjnych specjalności:
  - ART, Technologie informacyjne w systemach automatyki, kierunek Automatyka i Robotyka, Wydział Elektroniki,
  - ARP, Przemysł 4.0, kierunek Automatyka i Robotyka, Wydział Elektroniki,
  - CBD, Bezpieczeństwo danych, kierunek. Cyberbezpieczeństwo, Wydział Elektroniki
- Pełnomocnik ds. Polskich Ram Kwalifikacji na kierunku Automatyka i Robotyka na Wydziale Elektroniki,
- Sekretarz Komisji Programowej Kierunku Automatyka i Robotyka na Wydziale Elektroniki,
- Pełnomocnik ds. Polskich Ram Kwalifikacji na kierunku Informatyczne Systemy Automatyki na Wydziale Informatyki i Telekomunikacji
- Sekretarz Komisji Programowej Kierunku Informatyczne Systemy Automatyki na Wydziale Informatyki i Telekomunikacji,
- członek Zespołu ds. Ewaluacji w Politechnice Wrocławskiej, zakres prac:
  - prace koncepcyjne, programistyczne w Systemie Deponowania Politechniki Wrocławskiej (https://sd.pwr.edu.pl),
  - integracja źródeł danych - dorobek naukowy, dane o zatrudnieniu, dane o oświadczeniach do dyscyplin,
  - projektowanie i implementacja algorytmów optymalizacji w kontekście kryterium I-go (optymalizacja plecaków dyscyplin),
  - wsparcie programistyczne i informatyczne w procesie gromadzenia informacji o projektach i patentach raportowanych do POLON,

  - przeprowadzenie procesu integracji integracja Systemie Deponowania Politechniki Wrocławskiej z ORCID,
  - udostępnienie REST API na potrzeby przekazywania danych do POLON i PBN,
  - przeprowadzanie symulacji tworzenia nowych dyscyplin w Politechnice Wrocławskiej.

- Wsparcie procesów na Politechnice Wrocławskiej w ramach Systemie Deponowania Politechniki Wrocławskiej[8]:
  - wsparcie techniczne/informatyczne dla obsługi Zwiększonego Wynagrodzenia,
  - wsparcie techniczne/informatyczne dla obsługi Wyjazdów Zagranicznych,
  - wsparcie techniczne/informatyczne dla obsługi Przyjazdów Zagranicznych,
  - wsparcie techniczne/informatyczne dla obsługi Repozytorium Zamówień Publicznych,
  - wsparcie techniczne/informatyczne dla obsługi Postępowań Awansowych,
  - wsparcie techniczne/informatyczne dla obsługi baz danych Ośrodka Współpracy Nauki z Gospodarką.

- Pełnienie funkcji lidera zespołu programistów w Zespole Utrzymania Projektów Wrocławskiego Centrum Sieciowo-Superkomputerowego (WCSS):
  - organizacja, utrzymywanie oraz doskonalenia procesu wytwarzania oprogramowania w WCSS,
  - wsparcie procesu wdrażania, udostępniania i utrzymywania usług (systemów informatycznych) w ramach realizowanych projektów,
  - organizacja pracy zespołu programistycznego,
  - rekrutacja nowych pracowników.

- Wsparciu społeczności akademickiej Politechniki Wrocławskiej oraz Akademii Wychowania Fizycznego we Wrocławiu podczas pandemii COVID-19 przez zaangażowanie w obsługę procesu dydaktycznego prowadzonego w formie zdalnej. Realizowane zadania obejmowały obsługę systemu informatycznego do planowania zdalnych zajęć dydaktycznych na platformie ZOOM.

6.3 Informacja o osiągnięciach popularyzujących naukę

1. Popularyzacja obliczeń superkomputerowych wśród studentów i doktorantów PWr – wykład monograficzny, zapoznanie doktorantów z infrastrukturą Wrocławskiego Centrum Sieciowo-Superkomputerowego (WCSS).

2. „Innowacyjna Uczelnia – Innowacyjny Nauczyciel", W ramach projektu odpowiedzialny byłem za przeprowadzenie szkoleń. Odbyły się 2 edycje szkoleń:

[8] https://sd.pwr.edu.pl

a) Szkolenie "Dobre praktyki w programowaniu" obejmowało swoim zakresem zagadnienia związane z tematyką wytwarzania oprogramowania takie jak: system kontroli wersji, organizacja kodu źródłowego, testowanie oprogramowania oraz utrzymywanie i konserwacja kodu źródłowego (2019).

b) Zapoznanie uczestników szkolenia "Nowe techniki programowania równoległego" z tematyką obliczeń równoległych i rozproszonych, a w szczególności z architekturą i wykonywaniem obliczeń w systemie superkomputerowym (klaster) oraz technikami programowanie równoległego z wykorzystaniem MPI, OpenMP czy Python. Szkolenia w dużej części miały charakter warsztatów co pozwoliło uczestnikom na zdobycie umiejętności praktycznych (2019).

3. Cykliczne prezentacja, dla grup naukowców i studentów z kraju i z zagranicy, infrastruktury obliczeniowej Wrocławskiego Centrum Sieciowo-Superkomputerowego (2017-2021).

7 Inne informacje dotyczące kariery zawodowej

Dr inż. Mariusz Uchroński realizował liczne projekty badawczo–rozwojowe we Wrocławskim Centrum Sieciowo-Superkomputerowym:

1. *SpinLab* (Specjalizowana Polska Infrastruktura Naukowa na rzecz Laboratoriów Badawczych). W ramach projektu pełniłem funkcję lidera zespołu programistów. Do moich zadań należała koordynacja oraz prowadzenie prac programistycznych na potrzeby budowy aplikacji w formie dedykowanych Systemów Zarządzania Informacją Laboratoryjną (LIMS), których celem jest wsparcie realizacji prac laboratoryjnych. Zaangażowany byłem także w realizację dla zadań związanych z udostępnieniem Platformy E-SCIENCE.PL, która jest dedykowana środowiskom naukowym i pozwala na gromadzenie i współdzielenie danych oraz bezpieczną prezentację i wymianę wyników badań. 2014-2015, https://e-science.pl/

2. *PRACE-4IP* (Partnership for Advanced Computing in Europe 4th Implementation Phase Project). W ramach projektu zaangażowany byłem w wsparcie prowadzenia badań naukowych przez zespół Wydziału Podstawowych Problemów Techniki Politechniki Wrocławskiej w obszarze wykorzystanie akceleratorów GPU do symulacji w fizyce materiałów.[9] 2015-2017 https://prace-ri.eu/

3. *PRACE-5IP* (Partnership for Advanced Computing in Europe 5th Implementation Phase Project). W ramach projektu zaangażowany byłem w prace związane z realizacją zadania „Task Scheduling Library for Optimising Time-Scale Molecular Dynamics Simulations" [10] 2017-2019, https://prace-ri.eu/

[9] Uchroński M., Potasz P., Szymańska-Kwiecień A., Hruszowiec M., Using GPU accelerators for parallel simulations in material physics. Computational Methods in Science and Technology. 2018, vol. 24, nr 4, s. 249-258.

[10] O'Cais A., Swenson D., Uchroński M., Włodarczyk A. (2019). Task Scheduling Library for Optimising Time-Scale Molecular Dynamics Simulations, 2019, Zenodo, https://doi.org/10.5281/zenodo.3527643

4. *PRACE-6IP* (Partnership for Advanced Computing in Europe 6th Implementation Phase Project). W ramach projektu zaangażowany byłem w prace związane z realizacją zadania „Intelligent HTC for Committor Analysis"[11] 2019-2021, https://prace-ri.eu/

5. *AZON* (Atlas Zasobów Otwartej Nauki). W ramach realizacji projektu zajmowałem się koordynacją oraz prowadzenie prac programistycznych na potrzeby wytworzenia systemu deponowania oraz udostępniania cyfrowych zasobów nauki w sposób otwarty i spełniający kryteria dostępności WCAG2.0 i Five Stars of Open Data. 2016-2019, https://zasobynauki.pl/

6. *RegSOC* (Regionalne Centrum Bezpieczeństwa Cybernetycznego). W ramach realizacji projektu zajmowałem się koordynacją prac badawczych oraz programistycznych związanych z przygotowaniem Modułu Wykrywania Anomalii (MWA) w ruchu sieciowym. Moduł Wykrywania Anomalii (MWA) to oprogramowanie w Python służące do wykrywania anomalii w ruchu sieciowym w czasie pseudorzeczywistym uruchamiane w trybie wsadowym. Dane wejściowe pobierane są bezpośrednio z interfejsu sieciowego na który przekierowana jest kopia ruchu z monitorowanej (pod)sieci. Następnie wyodrębnione dane z nagłówków TCP/IP poddawane są agregacji i ekstrakcji do wektorów cech FAAC/SAAC/UNADA/EMAG. W kolejnym kroku następuje analiza zebranych wektorów cech poprzez: podział zebranych wektorów na klastry i detekcja odstających próbek (Density Grid-based Clustering Algorithm - DGCA), uczenie maszynowe z zastosowaniem autoenkoderów (AE i VAE), wykrywanie anomalii przez analizę gęstości rozkładu próbek (Robust Kernel-based Outlier Factor algorithm - RKOF). Założono, że statystyczne w czasie i wolumenie ruchu sieciowego anomalie są prawdziwymi wartościami odstającymi w stosunku do dominującego normalnego ruchu sieciowego. Wdrożenie MWA w infrastrukturze sieciowej wymaga dostrojenia parametrów konfiguracyjnych ze względu na charakter i wolumen ruchu sieciowego. Przeprowadzone eksperymenty z użyciem MWA wskazują że anomalie stanowią około 0,1% analizowanego ruchu sieciowego. 2018-2021, https://regsoc.pl/

7. *AZON2* (Atlas Zasobów Otwartej Nauki 2.0). W ramach realizacji projektu jestem odpowiedzialny za koordynację prac programistycznych pozwalających na rozbudowę Atlasu Zasobów Otwartej Nauki w celu zwiększenie dostępności, poprawę jakości oraz poszerzenie możliwości ponownego wykorzystania tych zasobów. 2020-2022, https://zasobynauki.pl/

8. *EuroHPC PL* (Narodowa Infrastruktura Superkomputerowa dla EuroHPC). W ramach realizacji projektu jestem odpowiedzialny za koordynacja prac programistycznych oraz biorę udział w pracach zespołu badawczego w ramach realizacji zadania „Platforma kwantowych badań operacyjnych i optymalizacji dyskretnej". Zakres realizowanych zadań obejmuje m.in. opracowanie modeli matematycznych dla problemów szeregowania zadań z dodatkowymi ograniczeniami (np. przezbrojenia) – sformułowanie jako QUBO (Quadratic Unconstrained Binary Optimization); analiza wymagań i ograniczeń w kontekście rozwiązywania

[11] Białczak M., O'Cais A., Swenson D., Uchroński M., Włodarczyk A. Intelligent HTC for Committor Analysis, 2019, Zenodo, https://doi.org/10.5281/zenodo.4572788

problemów szeregowania zadań na platformach obliczeń kwantowych: rozmiary rozwiązywanych problemów, wymagania co do czasu obliczeń i zasobów platform obliczeń kwantowych, możliwości redukcji liczby zmiennych; implementacja kwantowych algorytmów optymalizacji dla problemów szeregowania zadań. 2021–2023, www.eurohpc.pl

8. DARIAH-PL (Cyfrowa infrastruktura badawcza dla humanistyki i nauk o sztuce), W ramach projektu utworzona zostanie krajowa inteligentna cyfrowa infrastruktura badawcza dla humanistyki i nauk o sztuce. Będzie ona służyć pozyskiwaniu, przechowywaniu i integracji zróżnicowanych danych badawczych oraz przetwarzaniu, wizualizacji i udostępnianiu zasobów cyfrowych. W ramach realizacji projektu jestem odpowiedzialny za koordynację prac programistycznych w ramach przygotowania usługi „Usługa semantycznej anotacji konceptów znajdujących się na obrazach 2D" realizowanej w „Laboratorium zautomatyzowanego wzbogacania", 2021–2023, https://lab.dariah.pl

9. *SIEC11.5t* (Optymalizacja inwestycji drogowych w zakresie dostosowania sieci dróg krajowych do ruchu pojazdów o nacisku osi do 11,5 tony). Koordynacja prac programistycznych nad algorytmami optymalizacji kolejności wykonywania prac remontowych na sieci dróg oraz budowy systemu informatycznego wspomagającego przetwarzania danych pomiarowych o stanie nawierzchni dróg. 2020-2021.

Odbywałem wyjazdy do zagranicznych jednostek naukowo-badawczych związane z realizacją projektów:

1. Belgia, Antwerpia, University of Antwerp (2019-10-17 – 2019-10-19),

2. Włochy, Turyn, Polytechnic University of Turin. Współprowadzenie warsztatów w zakresie HPC (2019-07-01 – 2019-07-05),

3. Słowacja, Bratysława, Faculty of Natural Sciences of Comenius University, PRACE-6IP Kick-Off meeting (2019-05-28 – 2019-05-29),

4. Wielka Brytania, Edynburg, EPCC The University of Edinburgh, PRACE 5IP WP7 F2F Meeting (2018-10-22 – 2018-10-24),

5. Włochy, Turyn, Polytechnic University of Turin, Udział w warsztatach dotyczących realizacji projektu PRACE-5IP (zadanie WP7.2) (2018-07-16 – 2018-07-20),

6. Cypr, Nikozja, CaSToRC, The Cyprus Institute, Spotkanie robocze zadania WP7 projektu PRACE-5IP (2017-11-22 – 2017-11-24),

7. Grecja, Ateny, ATHINAIS Cultural Centre, PRACE-5IP Kick-Off and PRACE-4IP All-Hands Meeting (2017-01-31 – 2017-02-02),

8. Francja, Sophia, Antipolis French Institute for Research in Computer Science and Automation, Udział w spotkaniu F2F w ramach realizacji projektu PRACE-4IP (2016-12-05 – 2016-12-07),

9. Hiszpania, Barcelona, BSC PRACE (2016-08-31 – 2016-09-04),

10. Włochy, Rzym, CINECA (2016-04-12 – 2016-04-13),
11. Hiszpania, Barcelona, Barcelona Supercomputing Center (2016-01-21 – 2016-01-22),
12. Holandia, Amsterdam, SURFsara (2015-06-18 – 2015-06-19),
13. Czechy, Ostrava, VSB- Technical University of Ostrava (2015-04-28 –2015-04-29).

7.1 Autorstwo lub współautorstwo publikacji naukowych w czasopismach znajdujących się w bazie Journal Citation Reports (JCR) opublikowane po uzyskaniu stopnia doktora

1. **Uchroński, M.** Parallel Algorithm with Blocks for a Single-Machine Total Weighted Tardiness Scheduling Problem. Appl. Sci. 2021, 11, 2069. IF=2,679
2. Bożejko W[12]., **Uchroński M.**, Wodecki M., Blocks for two-machines total weighted tardiness flow shop scheduling problem. Bulletin of the Polish Academy of Sciences. Technical Sciences. 2020, vol. 68, nr 1, s. 31-41. IF=1.662
3. Bożejko, Chaczko Z., **Uchroński M.**, Wodecki M., Parallel patterns determination in solving cyclic flow shop problem with setups. Archives of Control Sciences. 2017, vol. 27, nr 2, s. s. 183-195. IF=1,545
4. Bożejko W., **Uchroński M.**, Wodecki M., Parallel metaheuristics for the cyclic flow shop scheduling problem. Computers and Industrial Engineering. 2016, vol. 95, s. 156-163. IF=2,623
5. Bożejko W., **Uchroński M.**, Wodecki M., Block approach to the cyclic flow shop scheduling. Computers and Industrial Engineering. 2015, vol. 81, s. 158-166. IF=2,086
6. Szczurek A., Maciejewska M., **Uchroński M.**, Determination of thermal preferences based on event analysis. Energy and Buildings. 2018, vol. 166, s. 210-216. IF=4,495
7. Włodarczyk A., **Uchroński M.**, Podsiadły-Paszkowsk A., Irek J., Szyja B.M., Mixing ReaxFF parameters for transition metal oxides using force-matching method, Journal of Molecular Modeling (accepted), https://doi.org/10.21203/rs.3.rs-930876/v1, IF=1.730

7.2 Prace w czasopismach spoza listy JCR opublikowane po uzyskaniem stopnia doktora

- **Uchroński M.**, Potasz P., Szymańska-Kwiecień A., Hruszowiec M., Using GPU Accelerators for Parallel Simulations in Material Physics, Computational Methods in Science and Technology, Volume 24 (4) 2018, 249–258.

[12]Kolejność autorów jest alfabetyczna.

- Bożejko W., **Uchroński M.**, Hejducki Z., Wodecki M., Rogalska M., Harmonogramowanie zintegrowanego systemu transportu i montażu kompozytowych elementów mostowych w systemie Just In Time, Przegląd Budowlany 2/2013, 39-44.

7.3 Otrzymane nagrody i wyróżnienia

- Nagroda Prezesa Rady Ministrów za rozprawę doktorską – 2015,
- Nagroda Rektora Politechniki Wrocławskiej w uznaniu wyróżniającego wkładu wspomagającego działalność Uczelni – 2018,
- Nagroda Rektora Politechniki Wrocławskiej w uznaniu wyróżniającego wkładu wspomagającego działalność Uczelni – 2020,
- Nagroda Rektora Politechniki Wrocławskiej w uznaniu wyróżniającego wkładu wspomagającego działalność Uczelni – 2021.

7.4 Recenzje artykułów

- Predicting of Large Scale IoT heart disease Data using of Map-Reduce Paradigm, Open Computer Science.
- Approximate Discontinuous Trajectory Hotspots, Open Computer Science.
- Why Robust Detection of Crypto-Ransomware is a Formidable Challenge: A Technical Justification, Open Computer Science.
- Exploring a flexible scoring scheme for a heuristic search technique: A case study of university timetables, International Journal of Mathematics and Mathematical Sciences.
- Multi Criteria Decision Making Model for Producing Multiple Products at the Same Time, Management and Production Engineering Review (MPER).
- Parallel Implementations of ARX-Based Block Ciphers on Graphic Processing Units, Mathematics, MDPI Mathematics (ISSN 2227-7390).
- Finding the best 3-OPT move in subcubic time, Algorithms, MDPI Algorithms (ISSN 1999-4893).
- CCA: Cost-Capacity-Aware Caching for In-Memory Data Analytics Frameworks, MDPI Sensors (ISSN 1424-8220).
- Bagging Machine Learning Algorithms: A Generic Computing Framework Based on Machine-Learning Methods for Regional Rainfall Forecasting in Upstate New York, MDPI Informatics (ISSN 2227-9709).
- A Novel Tradeoff Analysis between Traffic Congestion and Packing Density of Interconnection Networks for Massively Parallel Computers, MDPI Electronics (ISSN 2079-9292).

- On new stream algorithms generating sensitive digests of computer filesThe impact of vectorization and parallelization ofthe slope algorithm on performance and energyefficiency on multi-core architecture, FedCSIS 2021.

- Experimental Study of Hybrid Genetic Algorithms for the Maximum Scatter Travelling Salesman Problem, MDPI Algorithms (ISSN 1999-4893).

7.5 Projekty badawcze

- 2018–2020 projekt badawczy Narodowego Centrum Nauki OPUS nr 2017/25/B/ST7/02181 pt. „Zaawansowane metody optymalizacji zautomatyzowanych procesów cyklicznych", Politechnika Wrocławska, budżet 403 460 zł (wykonawca),

- 2013–2015 projekt badawczy Narodowego Centrum Nauki OPUS nr 2012/05/B/ST7/00102 pt. „Efektywne harmonogramowanie zadań w cyklicznych systemach wytwarzania", Politechnika Wrocławska, budżet 406 400 zł (wykonawca).

7.6 Współpraca z przemysłem

- 2018 *Toyota Motor Manufacturing Poland* – TNGA 2.0 – Optimalization of internal transport. Wykonania prac badawczo-rozwojowych dla TMMP dotyczących symulacji i optymalizacji transportu (Purchase Order No. 51516205 for TMMP). W ramach współpracy zajmowałem się opracowaniem oprogramowania do symulacji transportu w środowisku AutoMod z użyciem pojazdów typu AGV dla linii montażu silników oraz przygotowaniem symulacji transportu w środowisku AutoMod z użyciem pojazdów typu RCART dla linii montażu silników.

- 2018 *Toyota Motor Manufacturing Poland* – RDMC System for optimization of the chip collection process by AGVs (Purchase Order No. 29500037). W ramach współpracy zajmowałem się opracowaniem symulacji dla systemu automatycznego odbioru wiórów z obrabiarek oraz opracowaniem wraz z implementacją systemu do zbierania, przetwarzania i udostępniania danych o pozycjach pojazdów typu RCART.

8 Opis działalności naukowo badawczej

Działalność naukowo badawcza dotyczy efektywnych metod rozwiązywania silnie *NP*-trudnych problemów optymalizacji dyskretnej występujących między innymi w praktyce dyskretnych systemów wytwarzania. Prowadzone badania związane są z jednej strony z aspektami teoretycznymi optymalizacji dyskretnej, szczególnie w aspekcie obliczeń równoległych i rozproszonych, z drugiej z praktycznym ich stosowaniem w efektywnych algorytmach rozwiązywania rzeczywistych problemów. Rozważane problemy optymalizacji dyskretnej cechuje ogromny wymiar przestrzeni, wykładniczo rosnący czas obliczeń, nieregularne funkcje celu, chaotyczny rozkład ekstremów, brak efektywnych własności przyśpieszających obliczenia, itd. Stosowane obecnie przy ich rozwiązywaniu metody analizy oraz nawet najlepsze tradycyjne metody konstruowania algorytmów

wciąż pozostają dalekie od oczekiwań użytkowników, głównie ze względu na wysoki koszt prowadzenia obliczeń, ich czasochłonność oraz skomplikowanie implementacyjne.

Wysoki stopień skomplikowania problemów optymalizacji dyskretnej wymaga stosowania dedykowanych efektywnych metod w celu uzyskania rozwiązań bliskich optymalnym w rozsądnym czasie. Z jednej strony wykorzystanie obliczeń równoległych i rozproszonych pozwala na znaczne skrócenie procesu rozwiązywania problemów optymalizacji dyskretnej, z drugiej strony wykorzystanie specyficznych własności problemów pozwala na eliminację z procesu poszukiwań rozwiązań nierokujących, tj. takich, które nie dają możliwości poprawy wartości funkcji celu. Cechą wspólną wymienionych wyżej podejść jest możliwość zwiększenia obiecującego obszaru eksploracji przestrzeni rozwiązań dla problemów optymalizacji dyskretnej. W toku badań naukowych udowodnione zostały specyficzne własności problemów szeregowania zadań, a następnie własności te wykorzystane zostały do opracowania efektywnych algorytmów równoległych i rozproszonych.

Prowadzone przeze mnie badania w obszarze nauk inżynieryjno-technicznych obejmowały:

- Badania przestrzeni rozwiązań problemów pod kątem własności, które z powodzeniem można zastosować przy projektowaniu równoległych algorytmów metaheurystycznych.
- Implementacja algorytmów w środowisku obliczeń równoległych i rozproszonych dla szerokiej klasy architektur programowania współbieżnego, m.in. GPGPU (ang. *General Purpose Graphic Processing Unit*), procesorów wielordzeniowych oraz klastrów obliczeniowych.
- Zbadanie własności obliczeniowej algorytmów, a w szczególności ich efektywności, a także teoretycznego i rzeczywistego przyspieszenia.
- Zbadanie wpływu niepewności parametrów problemu (np. czasów trwania operacji, czasów przezbrojeń, itd.) na stabilność rozwiązań klasycznych metaheurystyk, tj. ich odporności na losowe zaburzenia.
- Badania rozkładu minimów lokalnych związane z hipotezą Wielkiej Doliny.

Główny nurt moich badań dotyczy konstrukcji efektywnych algorytmów dla silnie *NP*-trudnych problemów optymalizacji dyskretnej występujących między innymi w praktyce dyskretnych systemów wytwarzania, systemów obsługi, systemów transportowych, elastycznych systemów wytwarzania (FMS), telekomunikacji, systemów operacyjnych itd. Cechuje je ogromny wymiar przestrzeni, wykładniczo rosnący czas obliczeń, nieregularne funkcje celu, chaotyczny rozkład ekstremów, brak efektywnych własności przyśpieszających obliczenia, itd. Stosowane obecnie przy ich rozwiązywaniu metody analizy oraz nawet najlepsze tradycyjne metody konstruowania algorytmów wciąż pozostają dalekie od oczekiwań użytkowników, głównie ze względu na wysoki koszt prowadzenia obliczeń, ich czasochłonność oraz skomplikowanie implementacyjne. Znaczna część badań ma bezpośredni związek z rozwiązywaniem praktycznych problemów związanych z zarządzaniem w przemyśle, transporcie oraz budownictwie.

Na przestrzeni lat, moja współpraca z różnymi środowiskami naukowymi obejmowała wiele obszarów wspólnych badań, w tym między innymi:

- Optymalizację procesów transportowych w przemyśle motoryzacyjnym (z Krzysztofem Zielińskim oraz jego zespołem z Toyota Motor Manufacturing Poland),
- Wzorce, algorytmy równoległe ich wyznaczania oraz zastosowanie w metodach przybliżonych rozwiązywania problemów optymalizacji dyskretnej. We współpracy z dr. Zenonem Chaczką z University of Technology Sydney, Australia [1], wzorce zostały wykorzystane w konstrukcjach własności eliminacyjnych dla problemów szeregowania z przezbrojeniami maszyn.
- Własności przestrzeni rozwiązań, rozkłady minimów lokalnych, miary odległości w przestrzeni rozwiązań, "umiejscowiemie" *Wielkiej Doliny* oraz zastosowania uzyskanych wyników w konstrukcjach algorytmów (prof. Wojciech Bożejko, prof. Mieczysław Wodecki).
- Zastosowania metod optymalizacji w budownictwie (z prof. Zdzisław Hejducki z Wydziału Budownictwa Politechniki Wrocławskiej i dr hab. Magdalena Rogalska z Politechniki Lubelskiej, [2]).
- Szeregowaniem zadań w warunkach niepewności (współpraca z dr. Pawłem Rajbą z Instytutu Informatyki Uniwersytetu Wrocławskiego, [3]).

Prace te realizowałem w ramach zespołu pracowników Pracowni Systemów Dyskretnych (prof. Wojciech Bożejko, dr hab. Jarosław Pempera, prof. PWr., dr Mariusz Makuchowski) we współpracy z prof. Czesławem Smutnickim (Katedra Informatyki Technocznej PWr) oraz prof. Mieczysławem Wodeckim (Uniwersytet Wrocławski, Politechnika Wrocławska) jak również zespołem Wrocławskiego Centrum Sieciowo-Superkomputerowego. Stosowane są nowoczesne środowiska obliczeniowe – klastry systemów superkomputerowych oraz wydajne środowiska GPU oraz multi-GPU oparte na względnie niedrogich zestawach kart graficznych wykorzystywanych do obliczeń równoległych. Środowiska te stawiają zupełnie nowe jakościowo wyzwania dla projektowanych algorytmów, jeżeli tylko chcemy w pełni wykorzystać ich potencjał obliczeniowy.

Cel naukowy prowadzonych badań. Podstawowe cele moich badań obejmują od strony metodologii następujące zagadnienia:

- Opracowanie nowych klas algorytmów metaheurystycznych poprzez zaproponowanie i zbadanie specyficznych własności badanych problemów szeregowania zadań oraz wykorzystania tych własności przy konstrukcji zarówno sekwencyjnych jak i równoległych wersji znanych metaheurystyk jak i zupełnie nowych algorytmów.
- Implementacja zaproponowanych algorytmów równoległych i rozproszonych dla szerokiej klasy architektur programowania współbieżnego, m.in. GPGPU (General Purpose Graphic Processing Unit), procesorów wielordzeniowych oraz klastrów obliczeniowych.
- Zbadanie własności numerycznych zaimplementowanych algorytmów, a w szczególności ich efektywności w kontekście kosztowej optymalności (tj. osiągnięcia kosztu obliczeń tego samego rzędu jak koszt wykonania algorytmu sekwencyjnego) oraz teoretycznego przyspieszenia.

Wymienione powyżej cele należą do ogólniej sformułowanego zagadnienia jakim jest opisanie nowej dziedziny, tzn. klasy algorytmów wielowątkowych (tak równoległych jak i rozproszonych) rozwiązywania *NP-trudnych* problemów szeregowania zadań produkcyjnych. Wiele bowiem z proponowanych metod można niemal bez zmian przenieść także na szerszą klasę bardzo trudnych zagadnień optymalizacji dyskretnej, takich jak np. problemy transportowe (TSP,VRP), kwadratowy problem przydziału (QAP) czy problemy pakowania. W szczególności, w przypadku wielowątkowych algorytmów poszukiwań jednościeżkowych (ang. *single-walk parallelization*), o takiej samej trajektorii analizowanych rozwiązań jak trajektoria algorytmu sekwencyjnego) zaproponowane zostały nowe oryginalne metody zrównoleglenia wyznaczania wartości funkcji celu oraz równoległego wyznaczania otoczenia. Rozpatrywano problemy: jednomaszynowe, przepływowe (ang. *flow shop*), gniazdowe (ang. *job shop*) oraz elastyczne problemy gniazdowe, z maszynami równoległymi (ang. *flexible job shop*), w tym także cykliczne. Szczególnie ten ostatni przypadek, będący uogólnieniem klasycznego problemu gniazdowego, jest często spotykany w praktyce przy modelowaniu zagadnień np. w budownictwie oraz wieloasortymentowej organizacji produkcji. W zakresie wielowątkowych algorytmów wielościeżkowych (ang. *multiple-walk parallelization*) zaproponowano oryginalne metody rozwiązywania jedno- i wielomaszynowych klas problemów szeregowania zadań poprzez wielowątkowe algorytmy oparte na metodach: poszukiwania z zabronieniami (ang. *tabu search*), symulowanego wyżarzania (ang. *simulated annealing*), poszukiwania rozproszonego (ang. *scatter search*), algorytmu ewolucyjnego (ang. *evolutionary algorithm*) oraz populacyjnego (ang. *population-based metaheuristics*) a także algorytmu genetycznego (ang. *genetic algorithm*) i ich wielowątkowych wersji hybrydowych. W algorytmach hybrydowych wykorzystywano wielowątkowość nisko- i wysokopoziomowa, tj. zarówno na poziomie najbardziej czasochłonnych elementów algorytmu, jak i na poziomie zwielokrotnienia instancji procesów poszukiwań (wątków). Opracowanie metod zrównoleglania algorytmów dokładnych (np. metody B&B) ma na celu nie tyle stworzenia narzędzia do rozwiązywania dużych problemów szeregowania zadań, ile zaproponowanie algorytmu dokładnego wykorzystującego architekturę równoległą mogącego służyć do porównywania wyników metaheurystyk dla małych instancji problemów z rozwiązaniami dokładnymi. Równoległe algorytmy dokładne mogą mieć zastosowanie w rozwiązywaniu problemów cyklicznego szeregowania zadań.

Poniżej, zostaną dokładniej omówione główne nurty prowadzonych badań oraz ich wyniki (cytowania odnoszą się do najważniejszych prac umieszczonych w załączniku: **„Wykaz osiągnięć naukowych albo artystycznych, stanowiących znaczny wkład w rozwój określonej dyscypliny”**).

8.1 Obliczenia równoległe w optymalizacji dyskretnej

Zainteresowania obliczeniami wieloprocesorowymi wynikają z dwóch powodów. Pierwszy, to ogromna moc powszechnie już dostępnych komputerów wieloprocesorowych, która daje zupełnie nowe możliwości. Dzięki temu możliwe jest znaczne przyśpieszenie obliczeń ([C1,C4]). Drugi, to poprawienie szybkości zbieżności oraz wartości wyznaczanych przez algorytm rozwiązań (przy tej samej sumarycznej liczbie iteracji wszystkich procesorów jak przy uruchomieniu algorytmu na jednym procesorze, otrzymuje się lepsze rozwiązania). W literaturze pojawiło się nawet sformułowanie *równoległe meta-*

heurystyki – nowa klasa algorytmów. Właściwe zaprojektowanie algorytmu efektywnie wykorzystującego moc obliczeniową systemu komputerowego jest zadaniem nietrywialnym, ponieważ własności algorytmu równoległego (przyśpieszenie, efektywność itd.) zależą zarówno od wyboru strategicznego podejścia (poszukiwania jedno lub wielowątkowe itd.), jak i doboru jego elementów składowych oraz systemu kooperacji. Proste zrównoleglanie zwykle nie daje oczekiwanych wyników. Konieczność synchronizacji oraz częsta komunikacja niwelują korzyści płynące z wieloprocesorowości. Stąd, badania nad nowymi własnościami problemów oraz metodami konstrukcji algorytmów, które z jednej strony nie wymagają częstej komunikacji, a z drugiej zapewniają efektywną kooperację procesorów (np. celem dywersyfikacji poszukiwań). Ich efektem są opublikowane własności i algorytmy rozwiązywania różnych problemów optymalizacji dyskretnej oparte na metodach: poszukiwania z tabu ([11,51,67]), symulowanego wyżarzania ([53,66]), algorytmu genetycznego ([52,54,71]) oraz innych ([6,7,19,21]).

Pomimo znaczącego rozwoju w ostatnich latach teorii algorytmów oraz teorii optymalizacji, algorytmy heurystyczne wciąż pozostają często jedyną drogą dla uzyskania rozwiązań, które są zadawalające z punktu widzenia praktyki, zarówno co do rozmiaru rozwiązywanych w rozsądnym czasie przykładów, jak i dobroci (odległości od rozwiązania optymalnego) otrzymanych wyników. Zdecydowanie krótszą historię mają metody obliczeniowe wykorzystujące komputery wieloprocesorowe, choć klasyczny już dziś podział architektur tych komputerów dosyć dawno dokonał Flynn [12]. Faktycznie, dopiero w latach osiemdziesiątych ubiegłego wieku pojawiły się konstrukcje szybkich algorytmów równoległych.

Metaheurystyki oparte na metodzie lokalnych poszukiwań mogą być przedstawione jako procesy przeszukiwania grafu, w którym wierzchołkami są punkty przestrzeni rozwiązań (np. permutacje), a łuki odpowiadają relacji sąsiedztwa – łączą wierzchołki będące rozwiązaniami sąsiednimi w tej przestrzeni. Poruszanie się po takim grafie wyznacza pewną drogę (trajektorię). Wielowątkowe algorytmy metaheurystyczne korzystają z wielu wątków, zwykle uruchomionych na oddzielnych procesorach bądź rdzeniach, do współbieżnego generowania lub przeglądania grafu.

Można wyróżnić dwa podejścia do zrównoleglania procesu lokalnego poszukiwania, w zależności od liczby trajektorii generowanych współbieżnie w grafie sąsiedztwa.

- Pojedyncza trajektoria: algorytmy drobno- i średnioziarniste.
- Wiele trajektorii: algorytmy średnio- i gruboziarniste.

Podejścia te stawiają przed algorytmem pewne wymagania dotyczące częstotliwości komunikacji, co implikuje rodzaj ziarnistości. Algorytmy drobnoziarniste odpowiadają podejściu z częstszą komunikacją, gruboziarniste - z rzadszą.

Algorytmy jednościeżkowe. Algorytmy jednościeżkowe generują pojedynczą trajektorię, jednak mogą to czynić współbieżnie poprzez podział procesu badania otoczenia na kilka procesorów, z których każdy bada pewną część otoczenia szukając najlepszego elementu. Idea ta została zaproponowana najwcześniej dla sekwencyjnych algorytmów poszukiwań, patrz Nowicki i Smutnicki [16] pod nazwą metody reprezentantów (ang. *representatives*). Pochodzenie nazwy jest ściśle związane z działaniem metody, bowiem z każdej części otoczenia zostaje wybrany reprezentant, a dopiero spośród nich

najlepszy reprezentant jako następny punkt trajektorii poszukiwań. Odpowiedniki równoległe metody reprezentantów pojawiły się w literaturze później.

Metodologia zrównoleglania w oparciu o podejście jednościeżkowe została zastosowana w pracach cyklu [C3], [C4] oraz [C6]. W pracy [C3] zrównoleglono wyznaczanie tzw. wzorców, służących do przyspieszenie procesu przeszukiwania przestrzeni rozwiązań poprzez eliminowanie nieperspektywicznych rozwiązań („kierunków" poszukiwań). Z kolei w pracach [C4] oraz [C6] wykorzystano podeście jednościeżkowe do wyznaczania tzw. bloków zadań, tj. kryteriów eliminacyjnych służących do pośredniego przeszukiwania elementów przestrzeni.

Algorytmy wielościeżkowe. Algorytmy w których konstrukcji wykorzystano model wielościeżkowy badają współbieżnie przestrzeń rozwiązań za pomocą równolegle działających wątków poszukiwań. Algorytmy te można dodatkowo podzielić na podklasy ze względu na wymieniane informacje o aktualnym stanie poszukiwań:

- Niezależne procesy poszukiwań.
- Kooperujące procesy poszukiwań.

W przypadku gdy współbieżnie działające procesy poszukiwań nie wymieniają pomiędzy sobą żadnych informacji, mówimy o niezależnych (*independent*) procesorach poszukiwań. Jeśli zaś informacja uzyskana w trakcie eksploracji trajektorii przez proces poszukiwań jest przekazywana innemu procesowi a następnie wykorzystywana przez ten procesor, to można mówić o procesach kooperujących (*cooperative*). Spotykany jest także model mieszany, tzw. pół-niezależny (*semi-independent*) wykonujący niezależne procesy poszukiwań przy zachowaniu pewnych wspólnych parametrów.

Równoległe obliczenia dla jednej trajektorii. Jest to metoda służąca do przyspieszenia przeszukiwania grafu sąsiedztwa poprzez zrównoleglenie najbardziej czasochłonnych operacji - czyli obliczania wartości funkcji celu, bądź zrównoleglenie procesu generowania sąsiadów. W przypadku zrównoleglania obliczania wartości funkcji celu przyspieszenie obliczeń może być uzyskane przy zachowaniu identycznej trajektorii przejścia przez graf, jak trajektorii algorytmu sekwencyjnego. W drugim przypadku – dekompozycji generowania otoczenia na procesory równoległe - zaistnieć może sytuacja w której algorytm, sprawdzając równolegle większą ilość sąsiadów niż to czyni wersja sekwencyjna (najczęściej zaopatrzona w mechanizm redukcji rozmiarów otoczenia), poruszać się będzie po trajektorii lepszej niż sekwencyjny odpowiednik, wyznaczając korzystniejszą trasę przejścia przez graf i tym samym dochodząc do lepszych wyników obliczeń (wartości funkcji celu). Pierwsze aplikacje bazujące na opisywanym modelu pojawiły się w kontekście zrównoleglenia metody symulowanego wyżarzania i algorytmu genetycznego. Chociaż równoległa dekompozycja sąsiedztwa nie zawsze prowadzi do redukcji czasu obliczeń, jest jednak często stosowana do zwiększania rozpatrywanego sąsiedztwa. Tego typu algorytm równoległy przeszukiwania z tabu dla problemu komiwojażera został zaproponowany przez Fiechtera [1]. Synchroniczny tabu serach był także badany przez Porto i Ribeiro [14]. W pracy [15] prezentują oni równoległe algorytmy tabu oparte na modelu master - slave z dynamiczną strategią obciążania

procesorów. Bożejko, Pempera i Smutnicki [3] zaprezentowali równoległe podejście jednościeżkowe w rozwiązywaniu problemu przepływowego. Aarts i Verhoeven [1] różnicują klasę jednościeżkowych algorytmów równoległego przeszukiwania na dwie podklasy. Klasa jednokrokowa (*single - step*) obejmuje algorytmy, w których badanie otoczenia jest dzielone pomiędzy równoległe procesory, ale jako wynik wybierany jest jeden ruch. Z kolei w klasie wielokrokowej (*multiple - step*) sekwencja kolejnych ruchów w grafie sąsiedztwa jest wykonywana współbieżnie.

Równoległe obliczenia dla wielu trajektorii. Implementacje algorytmów opartych na równoległym wielościeżkowym przeszukiwaniu przestrzeni rozwiązań są aplikacjami gruboziarnistymi, czyli wymagającymi rzadkiej komunikacji. Są one łatwiejsze w zastosowaniu w systemach rozproszonych, jak na przykład klastrach komputerów klasy PC, dysponujących korzystnym wskaźnikiem ilorazu mocy obliczeniowej do ceny. Oprócz przyspieszenia obliczeń, uzyskać można także poprawę jakości otrzymywanych rozwiązań. Procesy poszukiwań mogą być niezależne lub kooperujące.

Niezależne procesy poszukiwań. W tej klasie można rozróżnić dwa podstawowe podejścia:

1. Przeszukiwanie przestrzeni rozwiązań za pomocą wielu trajektorii. Każdy z procesorów startuje z innego rozwiązania początkowego (lub różnych populacji w przypadku algorytmu genetycznego). Wątki poszukiwań mogą stosować ten sam lub różne algorytmy lokalnego poszukiwania, z takimi samymi lub różnymi wartościami parametrów strojących (np. długość listy tabu, wielkość populacji, itp.). Trajektorie mogą się przecinać w jednym lub wielu miejscach grafu sąsiedztwa. Tego typu podejście zostało zastosowane w pracy [C1] cyklu. W pracy [C1] zaproponowano wykorzystanie algorytmu równoległego, tabu search w wersji wielościeżkowej przejścia opartego na modelu MPSS (*Multiple Start Points Single Strategy*). Do komunikacji pomiędzy procesami obliczeniowymi wykorzystano bibliotekę MPI (*Message Passing Interface*) uruchomioną na wielu rdzeniach procesora, który realizuje własne procesy tabu search.

2. Równoległe badanie podgrafów grafu sąsiedztwa wyznaczonych przez dekompozycję problemu na kilka podproblemów (np. przez ustalenie pewnych zmiennych). Podgrafy grafu sąsiedztwa są badane współbieżnie bez przecinania się trajektorii. Otrzymujemy w tym przypadku całkowitą dekompozycję grafu sąsiedztwa na rozłączne podgrafy.

Pierwsza równoległa implementacja algorytmu tabu opartego na wielościeżkowym badaniu przestrzeni rozwiązań została zaproponowana przez Taillarda i dotyczyła kwadratowego zagadnienia przydziału (QAP) [16] oraz problemu gniazdowego [17]. Zrównoleglenie algorytmu genetycznego z użyciem niezależnych wątków poszukiwań nawiązuje do tak zwanego modelu wyspowego, bez komunikacji pomiędzy podpopulacjami zamieszkującymi poszczególne wyspy [7]. Chociaż zauważono pewne przyspieszenie, nie otrzymano poprawy wyznaczanych w ten sposób rozwiązań w stosunku do wyników sekwencyjnego algorytmu genetycznego z jedną dużą populacją. Fakt ten można wytłumaczyć szybką stagnacją podpopulacji (brakiem dalszej poprawy średniej wartości

funkcji celu po pewnej liczbie wykonanych iteracji) na każdym z procesorów pozbawionym komunikacji z pozostałymi.

Kooperujące procesy poszukiwań. Model ten jest najogólniejszym i najbardziej obiecującym typem strategii przeszukiwania przestrzeni rozwiązań przez równoległy algorytm metaheurystyczny. Wymaga jednak większej wiedzy programistycznej i znajomości specyfiki rozwiązywanego problemu. Kooperacja oznacza w tym wypadku wymianę informacji - doświadczeń dotyczących dotychczasowego procesu przeszukiwania przestrzeni przez równoległe procesy. Wymieniać należy specyficzne informacje, charakterystyczne dla problemu i metody, np. najlepsze znalezione rozwiązania, rozwiązania elitarne (mało różniące się od najlepszych znanych), częstotliwości ruchów, listy tabu, podpopulacje i ich rozmiary, i inne. Generowanie minimów lokalnych oraz kooperacja procesów poszukiwań wykorzystana została w pracy [C6] cyklu, a także innych pracach autora.

8.2 Metody przyspieszania przeszukiwania przestrzeni rozwiązań

Jest to główny kierunek intensywnie prowadzonych, na przestrzeni ostatnich lat, badań bezpośrednio związanych z konstrukcjami efektywnych algorytmów, głównie metaheurystycznych, rozwiązywania *NP-trudnych* problemów optymalizacji dyskretnej. Inspiracją były pewne aspekty dotyczące optymalizacji, wynikające z twierdzenia "*no free lunch*" zamieszczonego w pracy Wolpert i Macready [26]. Wynika bowiem z niego, że nie istnieje algorytm optymalizacyjny rozwiązywania wszystkich problemów, który byłby lepszy od innych algorytmów. Średnie zachowanie w takim przypadku każdego algorytmu jest dokładnie takie samo. Wykluczone jest więc istnienie w optymalizacji pewnej uniwersalnej, a przy tym skutecznej, strategii rozwiązywania takich problemów. Aby więc otrzymać ponad przeciętne wyniki, należy stosować silnie wyspecjalizowane strategie. Niezbędna jest więc intensyfikacja badań nad strukturą przestrzeni rozwiązań oraz identyfikacją cech mających decydujący wpływ na stopień trudności konkretnego problemu, a nawet przykładu (takie badania są przedstawione w pracy[C5]). Należy przy tym także brać pod uwagę ewentualną efektywność implementacji równoległej uzyskanych wyników, tj. znaczące przyśpieszenie algorytmu równoległego. Dla realizacji powyższych wymagań związanych z konstrukcjami efektywnych algorytmów, badania były prowadzone głównie w trzech obszarach, w celu:

1. Ograniczenia przestrzeni poszukiwań, tj. eliminacja pewnych elementów przestrzeni rozwiązań poprzez przegląd pośredni, z gwarancję zachowania optymalności rozwiązania.

2. Zmniejszenia złożoności obliczeniowej (skrócania czasu obliczeń) procedur przeszukiwania podzbiorów przestrzeni rozwiązań (głównie otoczeń) poprzez agregację obliczeń.

3. Wyznaczania reprezentantów pewnych podzbiorów przestrzeni rozwiązań, tj. rozwiązań nie gorszych ze względu na wartość funkcji kryterialnej.

Praktyczne zastosowania wymagają rozwiązywania problemów o dużych rozmiarach w „rozsądnym" czasie i z „akceptowalną" dokładnością. Jest to dużym wyzwaniem bowiem czas i dokładność, są w algorytmach rozwiązywania problemów optymalizacji dyskretnej zazwyczaj wzajemnie przeciwstawne. Prowadzone systematycznie badania teoretyczne są w silnej zależności z wykonywanymi jednocześnie eksperymentami obliczeniowymi. Ich efektem jest szereg oryginalnych wyników.

Kompleksowe badania przestrzeni rozwiązań problemów optymalizacji dyskretnej były prowadzone głównie pod kątem konstrukcji bloków oraz wzorców, tj. zmniejszenia rozmiaru przestrzeni rozwiązań poprzez przegląd pośredni pewnych elementów. Dotyczyły problemów optymalizacyjnych z kryterium C_{max} oraz kryteriami sumokosztowymi, w tym nieregularnymi. W przypadku, gdy rozwiązania problemu są reprezentowane przez permutacje, bloki są ciągami bezpośrednio po sobie występującymi elementami permutacji, w których "*dowolna zmiana elementów w bloku nie generuje lepszego rozwiązania*". Jest to tzw. własność eliminacyjna bloku. Sprowadza się więc do przeglądu pośredniego pewnych elementów i w konsekwencji zmniejszenia liczby elementów przestrzeni. Bloki gwarantują więc lokalną optymalność pewnych rozwiązań. Zostały zastosowane w konstrukcjach algorytmów zamieszczonych w pracach [C1-C6]. Z kolei wzorce są pewnymi "fragmentami" (np. podciągami) rozwiązań dla których można udowodnić, że istnieje rozwiązanie optymalne problemu, w którym występuje wzorzec. Na przykład w przypadku problemu komiwojażera są to ciągi miast będące fragmentem pewnego rozwiązania optymalnego. Wzorce występują w algorytmach opisanych w pracy [C3]. Przeprowadzone eksperymenty obliczeniowe wykazały, że efektem zastosowania zarówno bloków jak i wzorców w konstrukcjach algorytmów metaheurystycznych jest znaczna poprawa efektywności działania algorytmów.

Do przyspieszania przeszukiwania przestrzeni rozwiązań bloki oraz wzorce zostały zastosowane przy rozwiązywaniu cyklicznego problemu przepływowego rozpatrywanego w pracy [C3] (ang. *Flow Shop*). Jest to wielomaszynowy system produkcji cyklicznej, w którym dowolny element z ustalonej partii (mieszanki) przechodzi kolejno przez każdą z maszyn. Pomiędzy kolejno wykonywanymi elementami należy dodatkowo dokonać przezbrojenia maszyny. Problem polega na minimalizacji czasu cyklu, tj. czasu po którym może nastąpić wykonywanie następnej partii tych samych elementów. Udowodniono *silną NP-trudność* już dla pewnego szczególnego przypadku rozpatrywanego problemu. Jest on bowiem pewnym rozszerzeniem *silnie NP-trudnego*, klasycznego w teorii szeregowania zadań, permutacyjnego problemu przepływowego (oznaczanego w literaturze prze $F^*||C_{\max}$). O trudności zagadnienia świadczy także fakt, że pewna jego uproszczona wersja sprowadza się do rozwiązania problemu komiwojażera. Z tego powodu, do efektywnego wyznaczania rozwiązań skonstruowano szybkie algorytmy przybliżone, w których wykorzystano udowodnione w pracy specyficzne własności.

Metodykę prowadzonych badań oraz ich efekty przedstawię na przykładzie dwóch problemów szeregowania zadań z różną liczbą maszyn oraz różnymi kryteriami:

1. problemu jednomaszynowego z kryterium sumokosztowym ($1||\sum w_iT_i$), praca[C1], oraz

2. wielomaszynowego problemu przepływowego z minimalizacją czasu cyklu (pośrednio kryterium $C_{\max}$) opisanego w pracy [C3].

W obu przypadkach, zastosowanie wyniki prowadzonych badań dało znaczną poprawę

efektywności działania zarówno algorytmów sekwencyjnych jak i równoległych bowiem uzyskano, między innymi, znaczące przyśpieszenia.

8.2.1 Kryteria sumokosztowe

Pierwsze z szerzej omawianych zagadnień, jednomaszynowy problem minimalizacji sumy kosztów spóźnień (ang. *Total Weighted Tardiness Problem*, w skrócie TWT) można sformułować następująco:

Problem TWT. Każde zadanie ze zbioru $\mathcal{J} = \{1, 2, \ldots, n\}$ należy wykonać na jednej maszynie. Muszą być przy tym spełnione następujące ograniczenia:

(a) wykonywanie dowolnego zadania może się rozpocząć w chwili 0,

(b) w dowolnym momencie maszyna może wykonywać co najwyżej jedno zadanie,

(c) wykonywanie zadania nie może być przerwane,

(d) z każdym zadaniem $j \in \mathcal{J}$ są związane następujące parametry:

(i) czas wykonywania p_j,

(ii) żądany termin zakończenia d_j,

(iii) waga funkcji kosztu w_j.

Należy wyznaczyć kolejność wykonywania zadań, która minimalizuje sumę kosztów spóźnień.

Dowolne rozwiązanie rozpatrywanego problemu (kolejność wykonywania zadań na maszynie) może być reprezentowane przez permutację zadań (elementów zbioru $\mathcal{J}$). Przez Φ oznaczamy zbiór wszystkich takich permutacji. Niech $\pi \in \Phi$ będzie pewną permutacją zadań. Dla zadania $\pi(i)$ $(i = 1, 2, \ldots, n)$, niech będzie:

$C_{\pi(i)}$	-	*terminem zakończenia* wykonywania zadania,
$T_{\pi(i)} = \max\{0, C_{\pi(i)} - d_{\pi(i)}\}$	-	*spóźnieniem*,
$f_{\pi(i)}(C_{\pi(i)}) = w_{\pi(i)} \cdot T_{\pi(i)}$	-	*kosztem spóźnienia* (wykonania zadania).

W rozpatrywanym problemie należy wyznaczyć kolejność wykonywania zadań (permutację $\pi \in \Phi$) minimalizującą *sumę kosztów spóźnień*, tj. sumę

$$\mathcal{F}(\pi) = \sum_{i=1}^{n} f_{\pi(i)}(C_{\pi(i)}) = \sum_{i=1}^{n} w_{\pi(i)} T_{\pi(i)}. \tag{1}$$

W literaturze problem ten jest oznaczany przez $1||\sum w_i T_i$ i należy do klasy problemów *NP-trudnych*.

Dla permutacja $\pi \in \Phi$,

$$C_{\pi(i)} = \sum_{j=1}^{i} p_{\pi(j)}$$

jest *terminem zakończenia* wykonywania zadania $\pi(i)$ $(i = 1, 2, \ldots, n)$ w permutacji π. Zadanie $\pi(i)$ jest *terminowe* (ang. *early*), jeżeli czas jego zakończenia nie jest większy

od żądanego terminu zakończenia (tj. $C_{\pi(i)} \leqslant d_{\pi(i)}$) oraz *spóźnione* (ang. *late*), jeżeli czas ten jest większy od żądanego terminu zakończenia, czyli $C_{\pi(i)} > d_{\pi(i)}$.

Problem TWT polega więc na wyznaczeniu permutacji optymalnej $\pi^* \in \Phi$, która minimalizuje wartość funkcji kryterialnej $\mathcal{F}$ na zbiorze Φ to jest takiej, dla której

$$\mathcal{F}(\pi^*) = \min\{\mathcal{F}(\pi) : \ \pi \in \Phi\}. \tag{2}$$

W dowolnej permutacji $\pi \in \Phi$, istnieją subpermutacje (podciągi kolejnych zadań) dla których:

1) wykonywanie każdego zadania z subpermutacji kończy się przed jego żądanym terminem zakończenia nawet, gdy przestawimy je na ostatnią pozycję, albo

2) wykonywanie każdego zadania z subpermutacji kończy się za jego żądanym terminem zakończenia nawet, gdy przestawimy je na pierwszą pozycję.

W związku z tym wprowadzimy subpermutacje (bloki) zadań terminowych oraz zadań spóźnionych. Będą one stosowane do eliminowania ze zbioru Φ "gorszych" rozwiązań.

Bloki zadań w permutacji. Wyznaczanie bloków wymaga sformułowania i udowodnienia własności umożliwiających wydzielenie w permutacji ciągów elementów - subpermutacji, tj. występujących bezpośrednio po sobie elementów w permutacji, które umożliwiają eliminację pewnych rozwiązań. Prowadzi to bezpośrednio do poprawy efektywności działania algorytmów.

Subpermutacja zadań $\pi^{\mathcal{T}}$ w permutacji $\pi \in \Phi$ jest *blokiem zadań terminowych*, w skrócie $\mathcal{T}$-*blokiem*, jeżeli:

a) każde zadanie $j \in \pi^{\mathcal{T}}$ jest terminowe oraz $d_j \geqslant C_{last}$, gdzie C_{last} jest terminem zakończenia wykonywania ostatniego zadania z $\pi^{\mathcal{T}}$,

b) $\pi^{\mathcal{T}}$jest maksymalną subpermutacją spełniającą ograniczenie (a).

Z definicji wynika, że jeżeli $\pi^{\mathcal{T}}$ jest $\mathcal{T}$-blokiem, to spełniona jest nierówność $\min\{d_j : j \in \pi^{\mathcal{T}}\} \geqslant C_{last}$. Wobec tego w dowolnej permutacji elementów z $\pi^{\mathcal{T}}$ każde zadanie w permutacji π jest terminowe. Algorytm wyznaczania $\mathcal{T}$-bloku w permutacji jest zamieszczony w pracy [C1]. Ma złożoność obliczeniowa $O(n)$.

Podobnie, jak w przypadku zadań terminowych, subpermutacja zadań $\pi^{\mathcal{D}}$ w permutacji $\pi \in \Phi_n$ jest *blokiem zadań spóźnionych*, w skrócie $\mathcal{D}$-*blokiem*, jeżeli:

a′) każde zadanie $j \in \pi^{\mathcal{D}}$ jest spóźnione oraz $d_j < S_{first} + p_j$, gdzie S_{first} jest terminem rozpoczęcia wykonywania pierwszego zadania z $\pi^{\mathcal{D}}$,

b′) $\pi^{\mathcal{D}}$jest maksymalną subpermutacją spełniającą ograniczenie (a′).

Z powyższej definicji wynika, że każde zadanie należące do $\pi^{\mathcal{D}}$ jest w permutacji π spóźnione. Algorytm wyznaczania $\mathcal{D}$-*bloku*, zamieszczony w pracy [C1] ma złożoność obliczeniową $O(n)$.

Twierdzenie 1 ([C1]) *Dla dowolnej permutacji $\pi \in \Phi$ istnieje algorytm, o złożoności obliczeniowa $O(n)$, podziału (rozbicia) π na $\mathcal{T}$ oraz $\mathcal{D}$ bloki.* ■

Jeżeli $\pi^{\mathcal{D}}$ jest $\mathcal{D}$-blokiem w permutacji π, to dla dowolnego zadania $\pi(i) \in \pi^{\mathcal{D}}$

$$d_{\pi(i)} < S_{first} + p_{\pi(i)}.$$

Wobec tego funkcja kosztu wykonywania tego zadania

$$f_{\pi(i)}(x) = w_{\pi(i)}(x - d_{\pi(i)}), \;\; \forall x \in \mathcal{R}, \; x \geqslant S_{first} + p_{\pi(i)},$$

jest funkcją liniową. Z twierdzenia Smitha [20] wynika, że zadania w $\pi^{\mathcal{D}}$ występują w optymalnej kolejności wtedy i tylko wtedy, gdy

$$\frac{w_{\pi(i-1)}}{p_{\pi(i-1)}} \geqslant \frac{w_{\pi(i)}}{p_{\pi(i)}}, \;\; i = a+1, a+2, \ldots, b \tag{3}$$

gdzie subpermutacja $\pi^{\mathcal{D}} = (\pi(a), \pi(a+1), \ldots, \pi(b)), \; 1 \leqslant a < b \leqslant n$.

Permutacja $\pi \in \Phi$ jest *uporządkowana* (w skrócie $\mathcal{D}$-OPT) ze względu na podział $\mathcal{B}$, jeżeli w dowolnym $\mathcal{D}$-bloku każda para sąsiednich zadań spełnia relację (3), a więc występują one w optymalnej kolejności.

Twierdzenie 2 ([C1]) *Zmiana kolejności zadań w dowolnym bloku permutacji $\mathcal{D}$-OPT nie generuje permutacji o mniejszej wartości funkcji kryterialnej.* ■

Z powyższego twierdzenia wynika tzw. *własność eliminacyjna bloków*. Będzie ona stosowana przy generowaniu otoczeń.

Własność 1 *Dla dowolnej* $\mathcal{D}$-OPT *permutacji $\pi \in \Phi$, jeżeli $\beta \in \Phi$ oraz*

$$\mathcal{F}(\beta) < \mathcal{F}(\pi), \tag{4}$$

to w permutacji β przynajmniej jedno zadanie pewnego bloku z podziału π zostało przestawione przed pierwsze lub za ostatnie zadanie tego bloku.

Wobec tego, generując z $\mathcal{D}$-OPT permutacji $\pi \in \Phi$ nowe rozwiązania problemu TWT będziemy jedynie przestawiali element bloku przed pierwszy lub za ostatni element tego bloku.

Ruchy i otoczenia. W algorytmie przeszukiwania z tabu rozwiązywania problemu TWT zastosowano otoczenia generowane przez przekształcenia zwane *ruchami*, tj. zmianami pewnych pozycji elementów w permutacji polegającymi na:

1. Zamianie pozycjami pary elementów w permutacji (ang. *swap move*). Ruch s_l^k polega na zamianie pozycjami elementów $\pi(k)$ z $\pi(l)$, generując permutację $s_l^k(\pi) = \pi_l^k$. W skrócie będzie on nazywany *s-ruchem*. Złożoność obliczeniowa jego wykonania jest $O(1)$.

2. Przestawieniu elementu w permutacji na inną pozycję (ang. *insert move*). Element $\pi(k)$ jest przestawiany na pozycję l, generując permutację $i_l^k(\pi) = \pi_l^k$. Ruch i_l^k będzie w skrócie nazywany *i-ruchem*. Jego złożoność obliczeniowa jest $O(1)$.

Jeżeli $\mathcal{M}(\pi)$ jest pewnym zbiorem ruchów określonych dla permutacji $\pi \in \Phi$, to

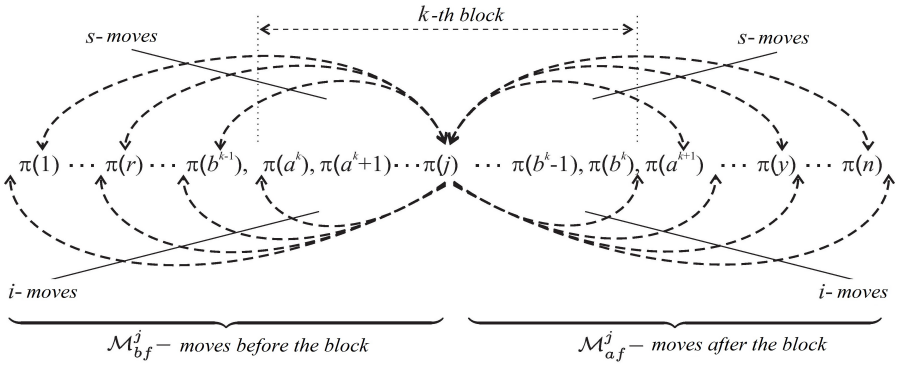
$$\mathcal{N}(\pi) = \{m(\pi) : \ m \in \mathcal{M}(\pi)\}$$

jest *otoczeniem* π generowanym przez ruchy z $\mathcal{M}(\pi)$.

W każdej iteracji algorytmu opartego na metodzie lokalnego poszukiwania, korzystając z generatora otoczeń (zbioru ruchów), wyznacza się pewien podzbiór zbioru rozwiązań – otoczenie. Niech

$$\mathcal{B} = [B^1, B^2, \dots B^\nu],$$

będzie podziałem uporządkowanej ($\mathcal{D}$-OPT) permutacji π na bloki. Rozpatrujemy zadanie $\pi(j)$ należące do pewnego bloku z podziału $\mathcal{B}$. Ruch, który może przynieść poprawę wartości kryterium, polegają na przestawieniu zadania $\pi(j)$ przed pierwsze (ang. *before*) lub za ostatnie (ang. *after*) zadanie tego bloku. Niech $\mathcal{M}^j_{bf}$ oraz $\mathcal{M}^j_{af}$ będą zbiorami tych ruchów (tj. odpowiednio wszystkich takich *i-ruchów* oraz *s-ruchów*). Zbiory te są przedstawione symbolicznie na rysunku 1. Zadanie $\pi(a^k)$ jest pierwszym, a $\pi(b^k)$ ostatnim elementem bloku B^k, do którego należy także rozpatrywane zadanie $\pi(j)$.



Rysunek 1: Ruchy zadania $\pi(j)$ mogące przynieść poprawę wartości funkcji celu.

Niech zbiór ruchów

$$\mathcal{M}(\pi) = \bigcup_{j=1}^{n} \mathcal{M}^j_{bf} \cup \bigcup_{j=1}^{n} \mathcal{M}^j_{af}, \tag{5}$$

będzie zbiorem wszystkich ruchów, które mogą przynieść poprawę (Własność 1), tj. ruchów przed lub za bloki pewnego podziału permutacji π.

W pracy [C1] przedstawiono pewne własności *i-ruchów* oraz *s-ruchów*, które można stosować w wyznaczaniu subotoczeń. Są to zarówno kryteria eliminacyjne jak i procedury wyznaczania pewnych podzbiorów ruchów i ich reprezentantów.

Reprezentanci ruchów. Ruch $m^* \in \mathcal{M}(\pi)$ jest *reprezentantem* zbioru ruchów z pewnego zbioru ruchów $\mathcal{W} \subset \mathcal{M}(\pi)$, jeśli

$$\forall r \in \mathcal{W}, \ \ \mathcal{F}(r(\pi)) \geqslant \mathcal{F}(m^*(\pi)).$$

Załóżmy, że permutacja $\pi \in \Phi$ jest $\mathcal{D}$-OPT, a otoczenie jest generowane przez ruchy typu wstaw (*i-ruchy*). Jeżeli i^k_l jest reprezentantem ruchów zbioru $\mathcal{M}^k \subset \mathcal{M}(\pi)$

($i_l^k \notin \mathcal{M}^k$), to ze zbiorów $\mathcal{M}_{bf}^k$ i $\mathcal{M}_{af}^k$ usuwa się ruchy należące do $\mathcal{M}^k$, tj. modyfikuje się je następująco:

$$\mathcal{M}_{bf}^k \leftarrow \mathcal{M}_{bf}^k \setminus \mathcal{M}^k \ \textbf{and} \ \mathcal{M}_{af}^k \leftarrow \mathcal{M}_{af}^k \setminus \mathcal{M}^k.$$

Umożliwia to pominięcie w procesie generowania otoczenia tych elementów, które nie dają bezpośrednio poprawy wartości kryterium.

Twierdzenie 3 ([C1]) *Jeżeli zadanie $\pi(k)$ po przestawieniu na pozycję l (wykonaniu ruchu i_l^k), $1 \leqslant k < l \leqslant n$, w permutacji π jest terminowe (tj. $C_{\pi(l)} \leqslant d_{\pi(k)}$), to dla pary ruchów i_{l-1}^k, i_l^k zachodzi*

$$\mathcal{F}(\pi_{l-1}^k) \geqslant \mathcal{F}(\pi_l^k).$$

■

Twierdzenie 4 [C1] *Niech $\pi(a)$ będzie pierwszym, a $\pi(b)$ ostatnim elementem pewnego $\mathcal{T}$-bloku permutacji π. Jeżeli dla zadania $\pi(k)$ $(0 < k < a)$ żądany termin zakończenia $d_{\pi(k)}) < C_{\pi(a)}$, to*

$$\mathcal{F}(\pi_l^k) \leqslant \mathcal{F}(\pi_{l+1}^k), \quad l = a-1, a, a+1, \ldots, b-1, b.$$

■

Dla zadania $\pi(k)$ $(1 \leqslant k \leqslant n)$, niech

$$\tau(k) = \begin{cases} 0, & \text{jeżeli } d_{\pi(k)} < p_{\pi(k)}, \\ \max\{j : 1 \leqslant j \leqslant n \wedge C_{\pi_j^k(j)} \leqslant d_{\pi(k)}\}, & \text{w przeciwnym przypadku.} \end{cases} \tag{6}$$

Jest to więc ostatnia pozycja w permutacji π, na którą wstawione zadanie $\pi(k)$, po wykonaniu *i-ruchu*, jest terminowe. Wobec tego zadanie $\pi(k)$ jest także terminowe w każdej z permutacji π_l^k, $l = 1, 2, \ldots, \tau(k)$.

Własność 2 ([C1]) *Dla ciągu ruchów $i_1^k, i_2^k, \ldots, i_{\tau(k)}^k$ zadania $\pi(k)$ zachodzi:*

$$\mathcal{F}(\pi_1^k) \geqslant \mathcal{F}(\pi_2^k) \geqslant \ldots \geqslant \mathcal{F}(\pi_{\tau(k)}^k),$$

gdzie parametr $\tau(k)$ jest wyznaczony przez (6). ■

Wobec tego ruch $i_{\tau(k)}^k$ jest reprezentantem zbioru ruchów $\{i_1^k, i_2^k, \ldots, i_{\tau(k)-1}^k\}$, a więc ruchy z tego zbioru można pominąć, modyfikując odpowiednio zbiory:

$$\mathcal{M}_{bf}^k \leftarrow \mathcal{M}_{bf}^k \setminus \{i_1^k, i_2^k, \ldots, i_{\tau(k)-1}^k\},$$

$$\mathcal{M}_{af}^k \leftarrow \mathcal{M}_{af}^k \setminus \{i_1^k, i_2^k, \ldots, i_{\tau(k)-1}^k\}. \tag{7}$$

Własność 3 ([C1]) *Niech $\pi(a)$ i $\pi(b)$ będą pierwszym i ostatnim zadaniem $\mathcal{T}$-bloku w permutacji π. Jeżeli $1 \leqslant k < a$ oraz $a \leqslant \tau(k) \leqslant b$, gdzie parametr $\tau(k)$ jest określony w* (6)*, to dla ciągu ruchów $i_{\tau(k)+1}^k$, $i_{\tau(k)+2}^k, \ldots, i_{b-1}^k$, i_b^k zachodzi:*

$$\mathcal{F}(\pi_{\tau(k)+1}^k) \leqslant \mathcal{F}(\pi_{\tau(k)+2}^k) \leqslant \ldots \leqslant \mathcal{F}(\pi_{b-1}^k) \leqslant \mathcal{F}(\pi_b^k).$$

■

Ruch $i^k_{\tau(k)+1}$ jest więc reprezentantem elementów zbioru $\{i^k_{\tau(k)+2},\ i^k_{\tau(k)+3}, \ldots, i^k_{b-1},\ r^k_b\}$. Można więc przyjąć, że

$$\mathcal{M}^k_{af} \leftarrow \mathcal{M}^k_{af} \backslash \{i^k_{\tau(k)+2},\ i^k_{\tau(k)+3}, \ldots, i^k_b\}. \tag{8}$$

Złożoność obliczeniowa algorytmów sprawdzania każdej z Własności 2–3 jest $O(n)$.

Własności ruchów zamień. Do eliminacji pewnych *s-ruchów* generujących subotoczenie będą stosowane kryteria eliminacyjne.

Na zbiorze zadań $\mathcal{J}$ wprowadzamy relację częściowego porządku $\Im$. Niech $\Gamma^-(i)$ oraz $\Gamma^+(i)$ będą odpowiednio zborami *poprzedników* i *następników* zadania $i \in \mathcal{J}$ w relacji $\Im$. Własności pozwalające na wyznaczenie elementów relacji $\Im$ zwane są, w literaturze, *kryteriami eliminacyjnymi*. Twierdzenie 5 umożliwia ustalenie relacji $\Im$ na zbiorze zadań $\mathcal{J}$.

Twierdzenie 5 ([C1]) *Jeżeli spełniony jest jeden z warunków:*

a) $p_r \leqslant p_j,\ \ w_r \geqslant w_j,\ \ d_r \leqslant d_j$, *lub*

b) $p_r \leqslant p_j,\ \ w_r \geqslant w_j,\ \ d_r \leqslant \sum_{l\in\Gamma^-_j} p_l + p_j$, *lub*

c) $w_r \geqslant w_j,\ \ d_r \leqslant d_j,\ \ d_r \geqslant \sum_{l\in N\backslash\Gamma^+_j} p_l + p_j$,

to istnieje rozwiązanie optymalne, w którym zadanie r poprzedza j, tj. $r\Im j$. ■

Po ustaleniu nowej relacji pomiędzy elementami i oraz j należy, dla uniknięcia powstania cyklu, wykonać tranzytywne domknięcie relacji.

Korzystając z relacji $\Im$ (stosując twierdzenie 5), będziemy ze zbioru *s-ruchów* generujących subotoczenie usuwali pewne elementy.

Własność 4 ([C1]) *Dla dowolnego s-ruchu $s^k_l \in \mathcal{M}^k_{af} \cup \mathcal{M}^k_{bf}$, jeżeli*

$$\pi(l) \in \Gamma^+_{\pi(k)},\ \ to\ \ \mathcal{M}^k_{af} \leftarrow \mathcal{M}^k_{af}\backslash\{s^k_l\},$$

a jeżeli

$$\pi(l) \in \Gamma^-_{\pi(k)},\ \ to\ \ \mathcal{M}^k_{bf} \leftarrow \mathcal{M}^k_{bf}\backslash\{s^k_l\}.$$

■

Do rozwiązywania problemu TWT zastosowano algorytm metaheurystyczny oparty na metodzie przeszukiwania z tabu. Poniżej przedstawiono główne jego elementy.

Otoczenie. Jest generowane przez ruchy *zamień* i *wstaw*. Niech $\mathcal{B}$ będzie rozbiciem $\mathcal{D}$-OPT permutacji π na bloki. Korzystając z "własności eliminacyjnych bloków" dla ruchów typu *wstaw* definiujemy zbiór ruchów przed i za blok:

$$\mathcal{M}(\pi) = \bigcup_{j\in\mathcal{I}} (\mathcal{M}^j_{bf} \cup \mathcal{M}^j_{af}). \tag{9}$$

Następnie:

1. Korzystając z własności 2 oraz 3 dla *i-ruchów* z $\mathcal{M}(\pi)$ usuwamy pewne podzbiory ruchów pozostawiając ich reprezentantów.

2. Korzystając z Własności 4 dla *s-ruchów* modyfikujemy odpowiednio zbiory ruchów $\mathcal{M}_{bf}^{j}$ oraz $\mathcal{M}_{af}^{j}$.

Ostatecznie, otoczenie permutacji π

$$\mathcal{N}(\pi) = \{m_l^k(\pi) : \ \ m_l^k \in \mathcal{M}(\pi)\}. \tag{10}$$

Procedura generowania tego otoczenia ma złożoność obliczeniową $O(n^2)$.

Rozwiązanie startowe. Dla każdego przykładu za rozwiązanie startowe algorytmu przyjęto najlepsze z rozwiązań wyznaczonych przez algorytmy konstrukcyjne [C1].

Warunek zakończenia. Algorytm kończył działanie po upływie z góry przyjętego czasu obliczeń.

Lista ruchów zakazanych. Lista $\mathcal{LT}$ jest ona obsługiwana na zasadzie kolejki FIFO. Dokładnie jest ona opisana w pracy [C1].

Eksperymenty obliczeniowe. Równoległa implementacja algorytmu przeszukiwania z tabu rozwiązywania problemu TWT jest opisana poniżej, w rozdziale "**Równoległe implementacje metod przyspieszania obliczeń**". Obecnie przedstawiamy przebieg testów wersji sekwencyjnej algorytmu, których jedynym celem było sprawdzenie wpływu stosowania subotoczeń na wartości wyznaczanych rozwiązań. Dla wyznaczenia wartości parametrów algorytmu dokonano wstępnych obliczeń. Na ich podstawie przyjęto:

- długość listy tabu: 7,
- długość listy pamięci długoterminowej: 5,
- rozwiązania startowe wyznaczone zostały przez algorytm META, [C1],
- kryterium stopu: $t = 120$ sekund.

Dla liczby zadań $n = 200, 500, 1000$ parametry zadań (liczby naturalne) były generowane losowo, zgodnie z rozkładem jednostajnym, z przedziału:

- p_i – $[1, 100]$,
- w_i – $[1, 10]$,
- d_i – $[P(1 - T_F - R_{DD}/2), P(1 - T_F + R_{DD}/2)]$,

gdzie $P = \sum_{i=1}^{v} p_i$, $R_{DD} = 0.2, 0.4, 0.6, 0.8, 1.0$ oraz $F_T = 0.2, 0.4, 0.6, 0.8, 1.0$. Dla każdej z 25 par wartości R_{DD} i T_F generowano 25 przykładów. W sumie wygenerowano 375 przykładów. Po 125 o różnej trudności, dla każdego n.

Średnie błędy względne rozwiązań, w stosunku do rozwiązań algorytmu META, są przedstawione w Tabeli 1, gdzie n jest liczbą zadań. W kolumnie TS zamieszczono wyniki algorytmu bez bloków, a TS_b - z blokami. Algorytm z blokami okazał się nieznacznie lepszy. Średnia względna poprawa wartości rozwiązań każdego z algorytmów wynosi ponad 8%.

Niektóre z przedstawionych własności można bezpośrednio zaadoptować w algorytmach rozwiązywania problemu z oknami czasowymi, tj. z najwcześniejszymi i najpóźniejszymi momentami zakończenia wykonywania zadań. W tym przypadku kryterium $\sum (u_i E_i + w_i T_i)$ jest funkcją nieregularną ("przesunięcia zadania w lewo może pogorszyć rozwiązanie"), co znacznie utrudnia rozwiązanie problemu.

przykładów	$n = 200$		$n = 500$		$n = 1000$	
group	TS	TS_b	TS	TS_b	TS	TS_b
001-025	-18.43	-18.49	-15.55	-16.85	-5.21	-7.54
026-050	-11.38	-11.29	-13.68	-13.88	-4.05	-3.54
051-075	-9.84	-9.78	-9.18	-9.14	-3.79	-3.65
076-100	-5.77	-5.73	-3.32	-3.46	-4.60	-3.82
101-125	-2.80	-2.79	-3.99	-3.93	-1.32	-0.78
Średnio	-9.64	-9.62	-9.15	-9.45	-3.79	-3.87

Tabela 1: Średnie błędy względne

8.2.2 Problemy z minimalizacją czasu cyklu

Drugi z szerzej omawianych problemów, permutacyjny problem przepływowy z minimalizacją czasu cyklu (ang. *Cyclic Flow Shop Scheduling*, w skrócie **CFS**) można sformułować następująco

Problem CFS. Dany jest zbiór n zadań $\mathcal{J} = \{1, 2, \ldots, n\}$, które należy wykonywać cyklicznie (w sposób powtarzalny) na maszynach ze zbioru $\mathcal{M} = \{1, 2, \ldots, m\}$. Dowolne zadanie należy wykonać kolejno, na każdej z m maszyn $1, 2, \ldots, m$ (porządek technologiczny). Zadanie $j \in \mathcal{J}$ jest ciągiem m operacji $O_{1,j}, O_{2,j}, \ldots, O_{m,j}$. Operacja $O_{k,j}$ odpowiada czynności wykonywania zadania j na maszynie k, w czasie $p_{k,j}$ $(k = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n)$. Po zakończeniu pewnej, a przed rozpoczęciem następnej operacji należy wykonać przezbrojenie maszyny. Niech $s^k_{i,j}$ $(k \in \mathcal{M},\ i \neq j\ i, j \in \mathcal{J})$ będzie czasem przezbrojenia maszyny k pomiędzy operacją $O_{k,i}$ oraz $O_{k,j}$. Należy wyznaczyć kolejność wykonywania zadań (taką samą na każdej maszynie), która minimalizuje czas cyklu, tj. terminy rozpoczęcia wykonywania zadań ze zbioru $\mathcal{J}$ w kolejnych cyklach. Muszą być przy tym spełnione następujące ograniczenia:

(a) każda operacja może być wykonywana tylko przez jedną, określoną przez ciąg technologiczny, maszynę,

(b) żadna maszyna nie może wykonywać jednocześnie więcej niż jedną operację,

(c) zachowany musi być porządek technologiczny wykonywania operacji,

(d) wykonywanie żadnej operacji nie może być przerwane przed jej zakończeniem,

(e) pomiędzy kolejno wykonywanymi, na tej samej maszynie, operacjami należy wykonać przezbrojenie,

(f) każde zadanie jest kolejno wykonywane po upływie czasu cyklu.

Zbiór zadań $\mathcal{J}$ wykonywanych w pojedynczym cyklu nazywany jest MPS-em (ang. *Minimal Part Set*). MPS-y są przetwarzane bezpośrednio jeden po drugim w sposób cykliczny. W każdym z MPS-ów zadania ze zbioru $\mathcal{J}$ są na każdej maszynie wykonywane w takiej samej kolejności (ang. *Permutation Flow Shop*).Wobec tego, każde uszeregowanie zadań na maszynach może być reprezentowane przez pewną permutację

$\pi = (\pi(1), \ldots, \pi(n))$ elementów ze zbioru $\mathcal{J}$. Niech Φ będzie zbiorem wszystkich takich permutacji (rozwiązań dopuszczalnych).

Rozpatrywany w pracy problem sprowadza się do wyznaczenia permutacji zadań (tj. momentów rozpoczęcia wykonywania zadań na maszynach spełniających ograniczenia (a)-(f)), aby czas cyklu (czas po którym dowolne zadanie jest wykonywane w kolejnym MPS-ie) był minimalny.

Model matematyczny problemu. Niech $\pi \in \Phi$ będzie pewną kolejnością wykonywania zadań na maszynach (taką samą dla każdego MPS-u). Przez $[S^h]_{m\times n}$ oznaczamy macierz terminów rozpoczęcia wykonywania zadań w h-tym MPS-ie, gdzie $S^h_{i,\pi(j)}$ oznacza termin rozpoczęcia wykonywania zadania $\pi(j)$ na maszynie i w h-tym MPS-ie. Zakładamy, że nie tylko kolejność zadań jest cyklicznie powtarzana w każdym z MPS-ów ale także, że harmonogram czasowy pracy systemu (tj. wykonywanie kolejnych MPS-ów) jest cykliczny. Oznacza to, że istnieje stała (*okres*) $T(\pi)$ taka, że

$$S^{h+1}_{i,\pi(j)} = S^h_{i,\pi(j)} + T(\pi), \;\; i = 1, ..., m, \;\; j = 1, ..., n, \;\; h = 1, 2, ... \tag{11}$$

Okres $T(\pi)$ zależy oczywiście od permutacji π i jest nazywany *czasem cyklu* systemu dla permutacji $\pi \in \Phi$. Minimalną wartość $T(\pi)$, będziemy nazywać *minimalnym czasem cyklu* i oznaczać przez $T^*(\pi)$).

Optymalną wartość czasu cyklu $T^*(\pi^*)$ (rozwiązanie problemu **CFS**) można wyznaczyć rozwiązując następujące zadanie optymalizacyjne:

wyznaczyć

$$T^*(\pi^*) = \min\{T^*(\pi) : \; \pi \in \Phi\}, \tag{12}$$

przy ograniczeniach:

$$S^h_{i,\pi(j)} + p_{i,\pi(j)} \leqslant S^h_{i+1,\pi(j)}, \;\; i = 1, ..., m-1, \;\; j = 1, ..., n, \tag{13}$$

$$S^h_{i,\pi(j)} + p_{i,\pi(j)} + s^i_{\pi(j),\pi(j+1)} \leqslant S^h_{i,\pi(j+1)}, \;\; i = 1, ..., m, \;\; j = 1, ..., n-1, \tag{14}$$

$$S^h_{i,\pi(n)} + p_{i,\pi(n)} + s^i_{\pi(n),\pi(1)} \leqslant S^{h+1}_{i,\pi(1)}, \;\; i = 1, ..., m, \tag{15}$$

$$S^{h+1}_{i,\pi(j)} \leqslant S^h_{i,\pi(j)} + T^*(\pi), \;\; i = 1, ..., m-1, \tag{16}$$

gdzie $h = 1, 2, ...$ Ponadto przyjmujemy, że termin rozpoczęcia wykonywania pierwszego zadania na pierwszej maszynie w pierwszym MPS-ie $S^1_{1,\pi(1)} = 0$. Ostatnie ograniczenie (16) jest charakterystyczne dla wytwarzania cyklicznego, bowiem ustala zależność pomiędzy momentami rozpoczęcia wykonywania zadań w następujących po sobie MPS-ach.

Dla ustalonej permutacji $\pi \in \Phi$ oraz pierwszego MPS-a, przez

$$T_k(\pi) = \sum_{i=1}^{n-1} (p_{l,\pi(i)} + s^k_{\pi(i),\pi(i+1)}) + p_{l,\pi(n)} + s^k_{\pi(n),\pi(1)} \tag{17}$$

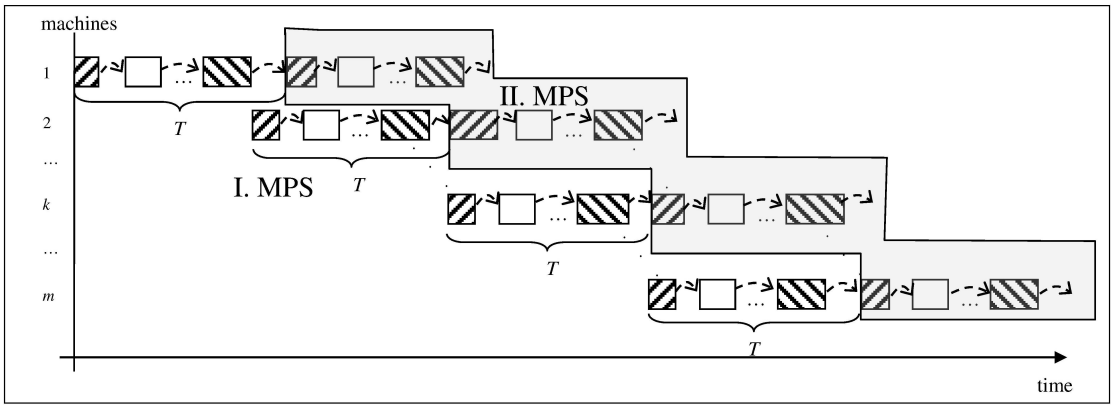
oznaczamy czas wykonywania zadań w kolejności π, wraz z przezbrojeniami, na k-tej maszynie (suma ta zawiera także czas przezbrojenia pomiędzy ostatnią operacją $\pi(n)$ pierwszego MPS-a, a pierwszą $\pi(1)$ operacją drugiego MPS-a). W skrócie czs ten będziemy nazywali k-tym *pikiem* (Rysunek 3).

Łatwo zauważyć, że dla permutacji zadań $\pi \in \Phi$ minimalny czas cyklu

$$T^*(\pi) = \max\{T_k(\pi) : \ \ k = 1, 2, \ldots, m\}. \tag{18}$$

Wyznaczanie czasu cyklu. Niech $\pi \in \Phi$ będzie pewną ustaloną permutacją zadań. Ponieważ kolejność wykonywania zadań w ramach każdego MPS-u jest taka sama, więc wystarczy wyznaczyć momenty rozpoczęcia wykonywania zadań $S^h_{k,\pi(i)}$ dla *pierwszego* MPS-a i dokonać odpowiedniego ich przesunięcia o wielkość $kT(\pi)$ ($k = 1, 2, \ldots$) i w ten sposób otrzymamy terminy rozpoczęcia zadań w następnych MPS-ach.

Na rysunku 2 przedstawiono fragment diagramu Gantta dla dwóch pierwszych MPS-ów. Przyjęto, że maksimum w równości (18) osiągnięto dla k-tej maszyny.



Rysunek 2: Diagram Gantta dwóch pierwszych MPS-ów.

Wyznaczenie optymalnego czasu cyklu sprowadza się do wyznaczenia permutacji optymalnej π^*, dla której

$$T^*(\pi^*) = \min\{T^*(\pi) : \ \pi \in \Phi\}, \tag{19}$$

gdzie

$$T^*(\pi) = \max\{T_l(\pi) : \ l = 1, 2, \ldots, m\}. \tag{20}$$

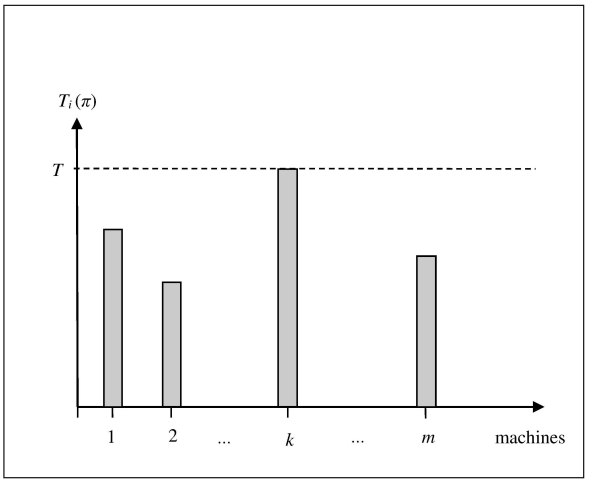
Załóżmy, że maksimum w równości (20) zostało osiągnięte dla k-tej maszyny, tj. $T^*(\pi) = T_k(\pi)$. Rozpatrujemy ciąg czasów pracy maszyn (pików) $T_1(\pi), T_2(\pi), \ldots, T_m(\pi)$. Symboliczne jest on przedstawiony na rysunku 3.

Warunkiem koniecznym skrócenie minimalnego czasu cyklu $T^*(\pi)$ jest zmniejszenie maksymalnego pika $T_k(\pi)$. W tym celu dla k-tej maszyny ($k \in \mathcal{M}$) konstruujemy symetryczny graf pełny

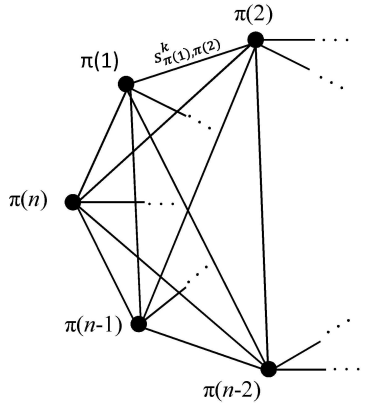
$$H_k = \langle V, E; p; s \rangle, \tag{21}$$

z obciążonymi wierzchołkami i krawędziami, gdzie:

- zbiór wierzchołków: $V = \mathcal{J}$,
- zbiór krawędzi: $E = \{\{v, u\} : \ v \neq u, \ v, u \in \mathcal{J}\}$,
- wagi wierzchołków: $p : \ V \to R$, $p(v) = p_{k,v}$ $v \in V$,
- wagi krawędzi: $s : \ E \to R, \ \forall e \in E, \ s(e) = s^k_e$.



Rysunek 3: Wartości $T_k(\pi),\ k = 1, 2, \ldots, m$.



Rysunek 4: Fragment grafu H_k dla k-tej maszyny.

Wagi wierzchołków są równe czasom wykonywania operacji, a krawędzi - czasom przezbrojeń pomiędzy operacjami. Fragment konstrukcji grafu H_k z przykładowymi wagami wierzchołków oraz krawędzi przedstawiono na rysunku 4.

Poniżej przedstawiono pewne związki pomiędzy czasami wykonywania zadań przez maszyny, a drogami w grafie H_k. Dowody są zamieszczone w pracy [C3].

Lemat 1 *Dla permutacji zadań $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, czas wykonywania zadań $T_k(\pi)$ (17) na maszynie k jest równy długości $L_k(\pi)$ cyklu Hamiltona*

$$P_k(\pi) = (\pi(1), \pi(2), \ldots, \pi(n), \pi(1))$$

w grafie $H_k,\ k \in \mathcal{M}$. ■

Lemat 2 *Permutacja $\beta \in \Phi$ jest optymalną kolejnością wykonywania zadań na k-tej maszynie (tj. minimalizuje wartość (17)) wtedy i tylko wtedy, gdy ciąg wierzchołków*

$$P_k^* = (\beta(1), \beta(2), \ldots, \beta(n), \beta(1))$$

jest cyklem komiwojażera w grafie $H_k,\ k \in \mathcal{M}$. ■

Wniosek (z Twierdzenia 1 [C3]). *Rozpatrywany problem cyklicznego szeregowania zadań* ***CFS*** *jest silnie NP-trudny.* ■

Niech $T^*(\pi)$ będzie minimalnym czasem cyklu dla permutacji $\pi \in \Phi$. Załóżmy, że $T_k(\pi)$ jest pikiem dominującym. Wobec tego

$$T^*(\pi) = T_k(\pi) = L_k(\pi),$$

gdzie $L_k(\pi)$ jest długością cyklu Hamiltona $(\pi(1), \pi(2), \ldots, \pi(n), \pi(1))$ w grafie $H_k(\pi)$. Tak więc "*aby skrócić czasu cyklu* $T(\pi)$ *należy zmniejszyć wartość dominującego pika, czyli zmniejszyć długość cyklu Hamiltona w grafie* $H_k(\pi)$". Zmniejszenie długości tego cyklu wymaga wygenerowania z π nowej permutacji, tj. zmiany pozycji pewnych elementów w π. Należy przy tym sprawdzić, czy po tej zmianie nie powstanie, na którejś z maszyn, nowy (większy) dominujący pik. Idee algorytmu wyznaczania suboptymalnego czasu cyklu (tj. rozwiązania problemu **CFS**) opartej na powyższych rozważaniach, można zapisać następująco:

π – bieżące rozwiązanie;
$\pi^* \leftarrow \pi$;
Krok 1: niech
$T_k(\pi) = \max\{T_i(\pi) :\ i = 1, 2, \ldots, m\}$; $\{maksymalny\ pik\}$
Krok 2: wygenerować z π permutację β
zamieniając pozycjami pewne elementy w π; $\{wykonanie\ ruchu\}$
Krok 3:
if $T_i(\beta) < T_k(\pi),\ i = 1, 2, \ldots, m,\ i \neq k$ $\{poprawa\}$
then
begin
$\pi \leftarrow \beta$;
if $T^*(\pi) < T^*(\pi^*)$ **then**
$\pi^* \leftarrow \pi$;
end
if (warunek zakończenia) **then** STOP **else goto** Krok 1.

W dalszej części pokażemy, że pewne ruchy generujące permutacje β w Kroku 2, które nie zmniejszają dominującego pika, można pominąć.

Bloki zadań w permutacji. Niech $\pi_k^* = (\pi_k^*(1), \pi_k^*(2), \ldots, \pi_k^*(n), \pi_k^*(1))$, będzie cyklem komiwojażera (optymalnym cyklem Hamiltona) w grafie $H_k(\pi)$, $k \in \mathcal{M}$. Jest to optymalna (ze względu na sumę czasów) kolejność wykonywania zadań na k-tej maszynie. Permutację tą będziemy nazywali *wzorcem* dla k-tej maszyny.

Niech

$$B = (\pi(a), \pi(a+1), \ldots, \pi(b)), \tag{22}$$

będzie ciągiem bezpośrednio występujących po sobie zadań w permutacji $\pi \in \Phi$, π_k^* *wzorcem* dla k-tej maszyny oraz u, v $(u \neq v,\ u, v = 1, \ldots, n)$ parą pozycji elementów w π_k^* takich, że:

W1: $\pi(a) = \pi_k^*(u), \pi(a+1) = \pi_k^*(u+1), \ldots, \pi(b-1) = \pi_k^*(v-1)$, $\pi(b) = \pi_k^*(v)$, lub

W2: $\pi(b) = \pi_k^*(u), \pi(b-1) = \pi_k^*(u+1), \ldots, \pi(a+1) = \pi_k^*(v-1)$, $\pi(a) = \pi_k^*(v)$,

W3: B jest maksymalnym podciągiem ze względu na zawieranie, tj. nie można go powiększyć ani o element $\pi(a-1)$, ani o $\pi(b+1)$ tak, aby były spełnione ograniczenia **W1** lub **W2**.

Jeżeli ciąg zadań (22) w permutacji π spełnia warunki **W1** i **W3** lub **W2** i **W3**, to nazywamy go *blokiem* na k-tej maszynie ($k \in \mathcal{M}$).

Twierdzenie 6 ([C3]) *Dla dowolnej maszyny istnieje rozbicie permutacji $\pi \in \Phi$ na bloki.* ■

Algorytm rozbicia (podziału) permutacji zadań na bloki ma złożoność obliczeniową $O(n)$.

Twierdzenie 7 ([C3]) *Jeżeli permutacja β została wygenerowana z permutacji π przez zmianę kolejności elementów w pewnym bloku na maszynie $k \in \mathcal{M}$, to*

$$T_k(\beta) \geqslant T_k(\pi).$$

■

Z powyższego twierdzenia wynika tzw. "*własność eliminacyjna bloków*". Przy generowaniu otoczeń w algorytmach lokalnych poszukiwań, można pominąć rozwiązania wyznaczone przez zmiany kolejności elementów w bloku wewnętrznym. Nie dają bowiem one bezpośredniej poprawy wartości rozwiązania. Własność ta znacznie przyspiesza działanie algorytmu. Pozostaje jednak problem wyznaczania wzorców.

Wzorce. Do wyznaczania bloków na dowolnej maszynie konieczny jest wzorzec. Jego wyznaczenie sprowadza się do rozwiązania problemu komiwojażera w grafie H_k.

Własność 5 ([C3]) *Jeżeli $(v_1, v_2, \ldots, v_n, v_1)$ jest cyklem komiwojażera w grafie H_k ($k \in \mathcal{M}$), to $(v_t, v_{t+1}, \ldots, v_l)$ $(1 \leqslant t < l \leqslant n)$ jest najkrótszą drogą z wierzchołka v_t do v_l zawierającą elementy zbioru $\{v_{t+1}, v_{t+2}, \ldots, v_{l-1}\}$.* ■

Wyznaczenie wzorca (optymalnego cyklu komiwojażera w rafie H_k) jest problemem *NP-trudnym*. Dla mniejszych przykładów można stosować algorytmy dokładne. Natomiast, dla większych przykładów istnieją dobre i szybkie algorytmy przybliżone, np. *2-opt*. Zastosowanie algorytmu aproksymacyjnego powoduje, że mogą nie być spełnione założenia Lematu 2. W wyniku tego z otoczenia permutacji $\pi \in \Phi$ mogą być wyeliminowane "dobre" (tj. dające poprawę minimalnego czasu cyklu) elementy. Efektywność stosowania takich "wzorców" zweryfikowano wykonując odpowiednie eksperymenty obliczeniowe.

Otoczenia. Do rozwiązywania problemu wyznaczania optymalnego czasu cyklu, w pracy [C3], zastosowano algorytm opartego na metodzie przeszukiwania z tabu. Zasadniczym elementem tej metody jest otoczenie, tj. sposób jego generowania oraz

przeszukiwania. Aby zmniejszyć czas wykonywania pojedynczej iteracji, zastosowano *subotoczenia* (podzbiory przestrzeni rozwiązań) generowane z wykorzystaniem "własności eliminacyjnych bloków". Jak wykazały eksperymenty obliczeniowe dzięki temu, znacznie poprawiła się efektywność działania algorytmu.

W literaturze opisano wiele ruchów polegających na zamianie kolejności elementów w permutacji. Na podstawie Twierdzenia 7 można wysnuć przypuszczenie, że ruch typu „wstaw" (ang. *insert*), polegający na przestawieniu zadania przed lub za blok może być, dla opisywanego problemu, skuteczną metodą generowania otoczenia. Generalnie, ruch taki sprowadza się do przestawienia zadania z jego pozycji w permutacji przed lub za inne zadanie. Dokładniej, dla różnych pozycji t, l w permutacji $\pi \in \Phi$ ruch typu wstaw (w skrócie *i-ruch*) i_l^t generuje nową permutację $\pi_l^t = i_l^t(\pi)$ przez przestawienie zadania $\pi(t)$ z pozycji t na pozycję l w π. Dokładnie ruchy takie są opisane i stosowane w algorytmach zamieszczonych w pracy [C1].

Jeżeli $\mathcal{I}(\pi)$ jest zbiorem *i-ruchów*, to **otoczeniem** permutacji $\pi \in \Phi$, jest zbiór

$$\mathcal{N}(\mathcal{I}(\pi)) = \{\pi_l^t : \ i_l^t \in \mathcal{I}(\pi)\}.$$

Liczba elementów otoczenia (moc zbioru $\mathcal{N}(\mathcal{I}(\pi))$ generowanego przez i-ruchy jest rzędu $O(n^2)$. W algorytmie przeszukiwania z tabu będą stosowane subotoczenia bazujące na własnościach eliminacyjnych bloków.

Dla r-tego bloku

$$B^r = (\pi(a^r), \pi(a^r + 1), \ldots, \pi(b^k)), \ r = 1, 2, \ldots, s,$$

w permutacji π definiujemy zbiory zadań **kandydatów** do przestawienia w ramach i-ruchu

$$\mathcal{J}_{af}^r = \{\pi(1), \ldots, \pi(a^r - 1), \pi(b^r + 2), \pi(b^r + 3), \ldots, \pi(n)\},$$
$$\mathcal{J}_{bf}^r = \{\pi(1), \ldots, \pi(a^r - 3), \pi(a^r - 2), \pi(b^r + 1), \pi(b^r + 2), \ldots, \pi(n)\}.$$

Zbiory te zawierają elementy, które będą przestawiane „**za**" lub „**przed**" odpowiedni blok, tj. za ostatni lub przed pierwszy element bloku. Na tej podstawie definiujemy zbiór i-ruchów za blok

$$\mathcal{I}_{af}^r(\pi) = \{i_l^x : \ \pi(x) \in \mathcal{J}_{af}^r\},$$

oraz zbiór ruchów przed blok

$$\mathcal{I}_{bf}^r(\pi) = \{i_t^x : \ \pi(x) \in \mathcal{J}_{bf}^r\}.$$

Oczywiście ruchy ze zbiorów $\mathcal{I}_{af}^r(\pi)$ i $\mathcal{I}_{bf}^r(\pi)$ **mogą** (lecz nie muszą) przynieść poprawę bieżącej wartości minimalnego czasu cyklu.

Na podstawie przedstawionych rozważań w algorytmie przeszukiwania z tabu będziemy ostatecznie stosowany zbiór ruchów

$$\mathcal{I}(\pi) = \bigcup_{r=1}^{s} [\mathcal{I}_{af}^r(\pi) \cup \mathcal{I}_{bf}^r(\pi)], \tag{23}$$

generujący subotoczenie $\mathcal{N}(\mathcal{I}(\pi))$ permutacji π. Rozmiar tego otoczenia nie zależy bezpośrednio od liczby zadań n i maszyn m.

Agregacja obliczeń. Przejdziemy obecnie do opisu metody wyznaczania zadania, które powinno zostać przestawione za ostatni $\pi(b^r)$ (lub przed pierwszy $\pi(a^r)$) element r-tego bloku. Zgodnie ze strategią przeszukiwania otoczenia szukamy takiego ruchu $i_l^x \in \mathcal{I}(\pi)$, którego wykonanie wygeneruje permutację π_l^x o możliwie najmniejszej wartości dominującego pika (tj. bieżącej minimalnej wartości czasu cyklu).

Dla uproszczenia, w dalszej części zapisu, przyjmujemy następujące założenia:

(i) bieżące rozwiązanie π jest permutacją naturalną, tj. $\pi = (1, 2, 3, \ldots, n)$,

(ii) rozpatrujemy r-ty blok $B = (a, a+1, \ldots, b-1, b)$.

Zgodnie z wcześniej przyjętym założeniem pomijamy także indeksy maszyn (np. przy oznaczeniach czasów przezbrojeń).

Dla dowolnego ruchu $i_l^v \in \mathcal{I}_{af}(\pi)$ wprowadzamy oznaczenie

$$\Delta_{af}(v,l) = s_{v-1,v+1} - s_{v-1,v} - s_{v,v+1} + s_{l-1,v} + s_{v,l} - s_{l-1,l}. \tag{24}$$

Podobnie dla ruch $i_l^v \in \mathcal{I}_{bf}(\pi)$

$$\Delta_{bf}(v,l) = s_{v-1,v+1} - s_{v-1,v} - s_{v,v+1} + s_{l-1,v} + s_{v,l} - s_{l-1,l}. \tag{25}$$

Złożoność obliczeniowa wyznaczenia wartości wyrażenia $\Delta_\lambda(v,l)$, $\lambda \in \{af, bf\}$ jest $O(1)$. Kolejne dwa twierdzenia podają zależności pomiędzy wartościami pików permutacji π oraz permutacjami generowanymi przez ruchy ze zbioru $\mathcal{I}(\pi)$, który został zdefiniowany w (23).

Twierdzenie 8 ([C3]) *Jeżeli permutacja π_v^l została wygenerowana z $\pi \in \Phi$ przez wykonanie ruchu $i_v^l \in \mathcal{I}_{af}(\pi)$, to wartość pika*

$$T(\pi_l^v) = T(\pi) + \Delta_{af}(v,l).$$

■

Podobnie, możemy obliczyć wartość pika dla permutacji generowanej przez ruch ze zbioru $\mathcal{I}_{bf}(\pi)$.

Twierdzenie 9 ([C3]) *Jeżeli permutacja π_v^l została wygenerowana z $\pi \in \Phi$ przez wykonanie ruchu $i_v^l \in \mathcal{I}_{bf}(\pi)$, to wartość pika*

$$T(\pi_l^v) = T(\pi) + \Delta_{bf}(v,l).$$

■

Przestawiając zadanie $\pi(x) \in \mathcal{I}_{af}$ przed pierwszy a lub $\pi(x) \in \mathcal{I}_{bf}$ za ostatni b element bloku generujemy permutację π_a^x lub π_b^x, dla której możemy wyznaczyć, korzystając odpowiednio z Twierdzenia 8 lub 9 wartości pików na każdej maszynie. Tak więc wartości $\Delta_\lambda(x)$, $\lambda \in \{af, bf\}$ będą stosowane w kryteriach wyboru z otoczenia elementu do przestawienia. Będziemy więc wybierali ruch o minimalnej wartości wyrażenia $\Delta_\lambda(v,l)$.

Eksperymenty obliczeniowe. W pracy [C3] przedstawiono eksperymenty obliczeniowe, których celem była ocena wpływu przedstawionych własności problemu CFS na efektywność działania algorytmu przeszukiwania z tabu. Badanie przeprowadzono w dwóch etapach. W pierwszej kolejności sprawdzono wpływ zastosowania bloków oraz agregacji obliczeń na wartości wyznaczanych rozwiązań, a w drugiej - na czasy obliczeń. Algorytm został zaimplementowany w języku C++ w komputerze PC z procesorem Intel Core 2 Duo 2,60 GHz. Obliczenia wykonano na 960 zmodyfikowanych przykładach dla hybrydowego problemu przepływowego rozpatrywanego w pracy [19].

Zastosowanie bloków lub/i agregacji obliczeń daje niewielką, bo o mniej niż 1%, poprawę średniej względnej wartości rozwiązań. Dokładne wyniki są przedstawione w Tabeli 2 zamieszczonej w pracy [C3]. Z kolei poniżej, w Tabeli 2 (jest to Tabela 3 z pracy [C3]) przedstawiono porównanie czasów obliczeń algorytmów. Jako kryterium zakończenia obliczeń przyjęto osiągnięcie (lub przekroczenie) założonej wartość rozwiązania referencyjnego. Poszczególne kolumny tabeli oznaczają odpowiednio:

- t – czas wykonania algorytmu przeszukiwania z tabu TSF ([14]),
- t_B – czas wykonania algorytmu TSF z blokami,
- t^* – czas wykonania algorytmu TSF z agregacją obliczeń,
- t^*_B – czas wykonania algorytmu TSF z blokami oraz agregacją obliczeń.

Zastosowanie bloków powoduje ponad 20-krotne skrócenie średniego czasu obliczeń (z 902.21 sekundy TSF do 39.63 sekundy z blokami). Natomiast, sama akceleracja skraca czas obliczeń o około 40%. Uwzględnienie w konstrukcji algorytmu bloków oraz agregacji obliczeń daje bardzo duże, bo ponad 40 krotne skrócenie średniego czasu obliczeń z 902.21 sek. do 23.77 sek.

$n \times m$	$t[s]$	$t_B[s]$	$t^*[s]$	$t^*_B[s]$
20×5	2,24	4,24	1,90	5,33
20×10	3,01	0,58	2,44	0,24
20×20	4,67	0,23	3,53	0,28
50×5	35,72	66,68	28,29	68,53
50×10	48,42	0,80	34,27	0,60
50×20	73,87	0,90	46,86	0,56
100×5	292,08	10,16	220,26	7,72
100×10	391,91	11,10	266,12	7,23
100×20	604,70	12,86	358,37	6,72
200×10	3212,17	141,63	2067,67	85,28
200×20	5255,53	186,79	2754,64	79,03
Średnio	**902,21**	**39,63**	**525,85**	**23,77**

Tabela 2: Czas obliczeniowy do momentu uzyskania ustalonej wartości funkcji kryterialnej.

Reasumując, przedstawione w pracy [C3] własności, zastosowane w konstrukcji algorytmu przeszukiwania z tabu dla problemu minimalizacji czasu cyklu, w sposób istotny

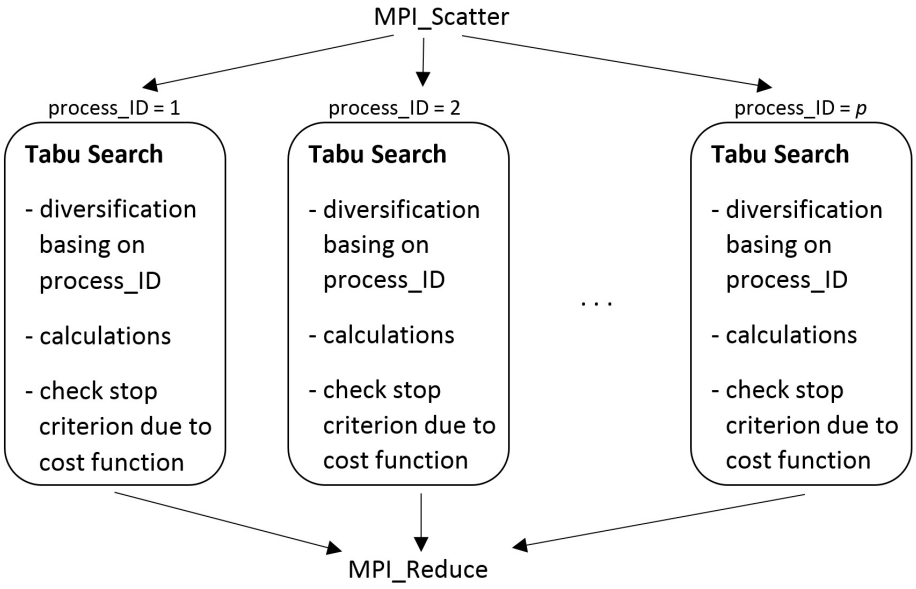
poprawiły efektywność algorytmu. Znacznie skrócił się czas obliczeń, przy jednocześnie niewielkiej poprawie wartości wyznaczonego rozwiązania. Dzięki temu, w tym samym czasie można wykonać zdecydowanie więcej iteracji algorytmu. Zazwyczaj prowadzi to do poprawy wartości wyznaczanych rozwiązań.

8.3 Równoległe implementacje metod przyspieszania obliczeń

Dla zagadnienia jednomaszynowego z kryterium $\sum w_i T_i$ zaproponowałem w pracy [C1] następujący algorytm przeszukiwania z zabronieniami (inaczej: przeszukiwania z tabu), w wersji równoległej, zaprojektowanej dla klastra obliczeniowego z biblioteką MPI.

8.3.1 Zrównoleglenie algorytmu przeszukiwania z tabu

Przedstawiony w pracy [C1] algorytm przeszukiwania tabu (w skrócie TSA) został zrównoleglony przy wykorzystaniu biblioteki MPI według schematu przeszukiwania przestrzeni przez niezależne procesy startujące z różnych punktów startowych (MSSS według klasyfikacji Voß'a [25]) wyznaczonych przez algorytm META([C1]). Na platformie klastra zaimplementowano równolegle niekooperujące procesy z mechanizmem dywersyfikacji rozwiązań startowych w oparciu o ideę metody Scatter Search. Liczba iteracji każdego z niezależnie wykonujących się algorytmów była proporcjonalna do rozmiaru problemu i liczby procesorów. Do zebrania danych zastosowano równoległy mechanizm redukcji (`MPI_Bcast`). Główną ideą algorytmu równoległego MWPTSA (Multiple-Walk Parallel Tabu Search Algorithm) przedstawiono na na Rysunku 5, a pseudokod - Algorithm 1.



Rysunek 5: Schemat zrównoleglania MWPTSA algorytmu przeszukiwania z tabu.

Eksperymenty obliczeniowe. Algorytm MWWTS równoległego przeszukiwania tabu został zaimplementowany w C++ przy użyciu biblioteki MPI. Obliczenia wyko-

Algorithm 1: Multiple-Walk Parallel Tabu Search Algorithm (MWPTSA)

Wejście: $\Pi^0 = \{\pi_1^0, \pi_2^0, \ldots, \pi_p^0\}$ – zbiór rozwiązań startowych;
p – numer procesora w MPI;
Wyjście: π^* – najlepsze znalezione rozwiązanie;

```
1  parfor i = 1, 2, ..., p do
2      π_i ← π_i^0; (aktualne rozwiązanie, lokalne dla procesu)
3      π_i^* ← π_i^0; (najlepsze lokalne rozwiązanie do tej pory znalezione)
4      repeat
5          LT_i ← ∅; (lista tabu)
6          Wygeneruj sąsiedztwo N(π_i) biorąc pod uwagę listę tabu LT_i oraz
            kryterium aspiracji;
7          δ = arg min_{π∈N(π_i)} F(π);
8          if F(δ) < F(π_i^*) then
9              π_i^* ← δ;
10         π ← δ;
11         Dodaj atrybuty rozwiązania do lokalnej listy tabu LT_i;
12     until Kryterium Stop (osiągnięcie rozwiązania przy założonej lub lepszej
        wartości funkcji kosztu);
13 Zmniejsz π_i^*, i = 1, 2, ..., p do najlepszej wartości przy użyciu opartego na
    drzewie schematu obliczeń równoległych;
14 return Θ^*
```

nano na klastrze BEM zainstalowanym we Wrocławskim Centrum Sieciowo-Superkomputerowym. Zadania obliczeń równoległych zostały uruchomione na procesorach Intel Xeon E5-26702.30GHz pod kontrolą systemu kolejki PBS.

Przykładowe dane testowe o różnej wielkości i stopniu trudności, na których wykonano obliczenia, podzielono na dwie grupy:

a) pierwsza grupa obejmuje 375 przykładów o trzech różnych rozmiarach ($n = 40, 50, 100$). Wraz z najlepszymi rozwiązaniami umieszczane są na stronie internetowej OR-Library [17].

b) druga grupa obejmująca instancje testowe dla $n = 200, 500, 1000$ została wygenerowana w następujący sposób:

- p_i – losowa liczba całkowita z zakresu [1100] o rozkładzie równomiernym,
- w_i – losowa liczba całkowita z zakresu $[1, 10]$ o rozkładzie równomiernym,
- d_i – losowa liczba całkowita z zakresu $[P(1 - T_F - R_{DD}/2), P(1 - T_F + R_{DD}/2)]$ z uniformem dystrybucja.

gdzie $P = \sum_{i=1}^{v} p_i$, $R_{DD} = 0.2, 0.4, 0.6, 0.8, 1.0$ (względny zakres czasów) i $F_T = 0, 2, 0, 4, 0, 6, 0, 8, 1, 0$ (średni współczynnik spóźnienia). Dla każdej z 25 par wartości R_{DD} i F_T wygenerowano pięć wystąpień. Wygenerowano łącznie 325 instancji — 125 dla każdej wartości n. Instancje testowe zostały opublikowane na stronie [22].

Tabela 3: PRD [%] w odniesieniu do najlepszych znanych rozwiązań uzyskanych przy użyciu algorytmów SWPT, EDD, COVERT, AU oraz META.

instance group	$SWPT$	$COVERT$	AU	EDD	$META$
$wt40$	579,92	152,39	15,77	138,55	10,46
$wt50$	4315,94	951,99	21,93	155,21	10,21
$wt100$	4692,65	1417,07	23,64	143,28	12,90
average	3196,17	840,49	20,45	145,68	11,19

Eksperymenty obliczeniowe algorytmów przedstawionych w pracy przeprowadzono w kilku etapach. Najpierw porównano algorytmy konstrukcyjne SWPT, EDD, COVERT, AU i META. Obliczenia wykonano na przykładach z grupy a). Otrzymane wyniki (procentowy błąd względny (26)) przedstawiono w tabeli 3, w porównaniu z przyjętymi rozwiązaniami referencyjnymi oraz danymi benchmarków z OR-Library [17]. Najlepsze rozwiązania wyznaczył algorytm META [18]. Rozwiązania tego algorytmu są punktem wyjścia algorytmu TSA. Uruchamiając te algorytmy na wielu procesorach, każdy z nich zaczynał od innego rozwiązania początkowego. Zróżnicowanie rozwiązań początkowych zostało wykonane przez uruchomienie sekwencyjnego algorytmu z niewielką liczbą iteracji $10i$, $i = 1, 2, \ldots, p$, gdzie p jest liczbą procesorów.

W celu wyznaczenia wartości parametrów algorytmów aproksymacyjnych dokonano wstępnych obliczeń na niewielkiej liczbie losowo wyznaczonych przykładów. Na podstawie analizy uzyskanych wyników przyjęto:

- długość listy tabu: 7,
- długość listy dla pamięci długotrwałej: 5,
- algorytm wyznaczania rozwiązania startowego: META,
- warunek zatrzymania algorytmu: czas obliczeń $t = 120$ sekund.

Jako rozwiązanie algorytmu równoległego znalazło się najlepsze rozwiązanie uzyskane przez poszczególne procesory. Dla każdego rozwiązania wyznaczono procentowy błąd względny:

$$PRD = \frac{F_{ref} - F_{alg}}{F_{ref}} \cdot 100\%, \tag{26}$$

gdzie:

F_{ref} – wartość rozwiązania referencyjnego uzyskana algorytmem META,
F_{alg} – wartość rozwiązania określona przez testowany algorytm.

Tabela 4 zawiera wartości poprawy PRD względem rozwiązania otrzymanego przez algorytm META, który jest również rozwiązaniem wyjściowym dla TSA (dlatego są ujemne) dla większych instancji rozmiarów ($n = 200$, 500 i 1000).

Tabela 5 zawiera wartości poprawy PRD w stosunku do rozwiązania uzyskanego przez algorytm META, który jest również rozwiązaniem wyjściowym dla równoległego TSA (dlatego są one ujemne) dla większych instancji rozmiarów ($n = 200$, 500 i 1000). Eksperymenty przeprowadzono dla liczby procesorów $p = 8, 16, 32$ i 64. Można zaobserwować, że zwiększenie liczby procesorów skutkuje poprawą jakości otrzymywanych rozwiązań, zwłaszcza dla algorytmu z blokami.

Tabela 4: Wyniki obliczeń dla sekwencyjnego algorytmu przeszukiwania tabu ($t = 120s$, META jako odniesienie).

instance	$wt200$		$wt500$		$wt1000$	
grupa	TS	TS_b	TS	TS_b	TS	TS_b
001-025	-18.43	-18.49	-15.55	-16.85	-5.21	-7.54
026-050	-11.38	-11.29	-13.68	-13.88	-4.05	-3.54
051-075	-9.84	-9.78	-9.18	-9.14	-3.79	-3.65
076-100	-5.77	-5.73	-3.32	-3.46	-4.60	-3.82
101-125	-2.80	-2.79	-3.99	-3.93	-1.32	-0.78
średnia	-9.64	-9.62	-9.15	-9.45	-3.79	-3.87

Tabela 5: Wyniki obliczeń dla równoległego algorytmu przeszukiwania tabu ($t = 120s$, META jako odniesienie).

number of	$wt200$		$wt500$		$wt1000$	
procesory	TS	TS_b	TS	TS_b	TS	TS_b
1	-9.64	-9.62	-9.15	-9.45	-3.79	-3.87
8	-9.65	-9.63	-9.17	-9.53	-3.88	-3.91
16	-9.65	-9.64	-9.17	-9.55	-3.88	-3.92
32	-9.65	-9.66	-9.17	-9.55	-3.88	-3.94
64	-9.66	-9.71	-9.19	-9.59	-3.88	-3.95

Tabela 6: Wartości przyspieszenia dla równoległego algorytmu wyszukiwania tabu.

	$p = 8$		$p = 16$	
	s_{TS}	s_{TS_b}	s_{TS}	s_{TS_b}
$wt40$	182.60	61491.57	185.57	61466.13
$wt50$	39.17	28943.19	39.54	23866.83
$wt100$	5.63	2150.46	5.78	2234.89
$wt200$	3.99	58752.36	4.24	55967.32
$wt500$	3.42	21.80	3.58	22.21
$wt1000$	2.23	1280.58	2.27	1724.82

Tabela 7: Wartości przyspieszenia dla równoległego algorytmu wyszukiwania tabu.

	$p = 32$		$p = 64$	
	s_{TS}	s_{TS_b}	s_{TS}	s_{TS_b}
$wt40$	193.85	60606.01	186.14	60595.42
$wt50$	40.01	53659.18	38.98	50530.73
$wt100$	5.94	2768.76	5.88	2793.09
$wt200$	4.13	56021.05	4.13	55445.96
$wt500$	3.47	22.10	3.53	22.90
$wt1000$	2.29	985.34	2.30	1020.59

Tabele 6 i 7 zawierają wyniki przyspieszenia. Wartości obliczono z zależności $s = t_s/t_p$, gdzie

- t_s czas, w którym algorytm sekwencyjny uzyskuje rozwiązanie F,
- t_p to czas, po którym algorytm równoległy uzyskuje rozwiązanie F.

Na podstawie wyników zamieszczonych w tabeli można stwierdzić, że duże przyspieszenia są uzyskiwane dla problemów o mniejszych rozmiarach (wartości s_{TS_b} powyżej 60000 dla $n = 40$). Algorytm blokowy daje lepsze przyspieszenie (średnio ponad 2600 razy lepsze). Wynika to prawdopodobnie z tego, że wielkość przestrzeni rozwiązania problemu (40!, 50!, 100!, 200!, 500!, 1000!) rośnie znacznie szybciej niż liczba procesorów (8, 16, 32, 64). Wzrost przyspieszenia w obrębie jednego rozmiaru problemu również nie jest duży, jak pokazano powyżej. Duża dysproporcja w przyspieszeniu algorytmu z blokami i bez bloków wynika z tego, że algorytm blokowy silnie ogranicza przestrzeń przeszukiwanych rozwiązań.

Jak można zauważyć zwiększenie rozmiaru problemu zmniejsza wartość uzyskanego przyspieszenia – co prawdopodobnie wynika z ograniczonego czasu obliczeń dla większej przestrzeni rozwiązań, jednak zwiększenie liczby procesorów nie ma istotnego wpływu na zwiększeni wartości uzyskiwanych przyspieszeń – nawet przy zwiększaniu rozmiaru problemu.

Podsumowanie. Na podstawie przeprowadzonej analizy problemu i przeprowadzonych eksperymentów obliczeniowych możemy wysnuć następujące wnioski. Metoda przeszukiwania tabu z wykorzystaniem właściwości bloków pozwala rozwiązać problem nie tylko szybciej niż klasyczna metaheurystyka przeszukiwania z tabu bez użycia tych właściwości, ale także z ogromnymi przyspieszeniami. Przyspieszenia osiągnięte przez algorytm równoległego przeszukiwania tabu z blokami są znacznie większe niż w przypadku klasycznego algorytmu przeszukiwania tabu, co potwierdza zasadność korzystania z właściwości eliminowania bloków. Fakt osiągnięcia tak ogromnych przyspieszeń powinien być przedmiotem dalszych badań. Zaproponowane kryteria eliminacji bloków można również wykorzystać do konstruowania wydajnych algorytmów równoległych rozwiązujących inne NP-trudne problemy szeregowania.

8.3.2 Równoległe wyznaczanie wzorców w rozwiązywaniu cyklicznego problemu przepływowego z przezbrojeniami

Poniżej przedstawiono także metodę przyspieszania obliczeń w problemie przepływowym. Opis ten został oparty na pracy [C3]. Rozpatrywany jest tam cykliczny problem przepływowy z czasami przezbrojeń. Silna NP-trudność już wielu najprostszych wersji problemu cyklicznego szeregowania, a rozpatrywanego problemu w szczególności, ogranicza zakres stosowania algorytmów dokładnych do instancji o małej liczbie zadań, choć w kontekście minimalizacji czasu cyklu konstrukcja i zastosowanie algorytmów dokładnych wydaje się całkowicie uzasadnione (Brucker et al. [10]). Niemniej, ze względu na NP-trudność, do wyznaczania satysfakcjonujących rozwiązań, stosuje się powszechnie szybkie algorytmy przybliżone oparte na technikach przeszukiwań lokalnych, np. symulowane wyżarzanie (w wersji równoległej Bożejko et al. [8]) czy tabu search (Bożejko et al. [9]). Metody tego typu opierają się zazwyczaj na dwupoziomowej dekompozycji problemu: wyznaczenie optymalnej kolejności zadań (poziom górny) oraz wielokrotne

wyznaczenie minimalnej wartości kryterium dla danej kolejności zadań (poziom dolny). O ile dla klasycznych, nie-cyklicznych problemów szeregowania rozwiązanie problemu dolnego poziomu można otrzymać w sposób efektywny czasowo przez analizę specyficznego grafu, to w przypadku postawionego problemu rozwiązanie zagadnienia dolnego poziomu jest stosunkowo czasochłonne. Stąd, wszelkie własności szczególne, w tym pozwalające na bardziej efektywne wyliczanie czasu cyklu, poszukiwanie harmonogramu oraz ograniczenie liczności lokalnie przeglądanego otoczenia lub przyspieszenie szybkości jego przeglądania są bardzo pożądane.

W pracy [C3] wykorzystano nowe własności eliminacyjne, tzw. wzorce, do zmniejszenie liczby rozwiązań przeglądanych podczas generowania otoczenia przez algorytmy lokalnych poszukiwań. Wyznaczanie wzorców może być wykonana zarówno sekwencyjne, jak i równolegle, z wykorzystaniem wieloprocesorowego środowiska obliczeń. Odpowiednie własności sformułowane zostały z wykorzystaniem modelu maszyny PRAM, będącej standardem teoretycznej weryfikacji złożoności obliczeniowej algorytmów równoległych.

Metoda rozwiązania. W wielu algorytmach heurystycznych rozwiązywania problemów NP-trudnych są przeglądane otoczenia, tj. podzbiory przestrzeni rozwiązań. W przypadku, gdy rozwiązaniami problemu są permutacje, zazwyczaj otoczenia są generowane przez ruchy typu „wstaw" lub „zamień" oraz ich złożenia [5]. Polegają one na zmianie pozycji elementów w permutacji. W praktycznych zastosowaniach (przy dużych n), przeglądanie otoczenia jest najbardziej czasochłonnym elementem algorytmu. Z opisu, zamieszczonych w literaturze eksperymentów obliczeniowych wynika, że liczba iteracji algorytmu ma bezpośredni wpływ na jakość wyznaczanych rozwiązań. Stąd, poszukiwanie metod przyspieszających działanie pojedynczej iteracji algorytmu. Jedną z nich jest zmniejszenie liczby elementów otoczenia oraz równoległe ich generowanie i przeglądanie. W przypadku problemów szeregowania zadań na wielu maszynach z minimalizacją czasu wykonywania zadań (C_{max}) są w tym celu z powodzeniem stosowane „blokowe własności eliminacyjne" [13]. Podobne własności zastosujemy w algorytmie rozwiązywania problemu wyznaczania minimalnego czasy cyklu, a dokładniej – minimalnego czasu pracy pojedynczej maszyny. Umożliwiają one eliminację elementów z otoczenia, które bezpośrednio nie dają poprawy najlepszego do tej pory znalezionego rozwiązania.

W pracy [C3] oraz rozdziale 8.2.2. Autoreferatu sformułowano cykliczny problem gniazdowy oraz przedstawiono jego własności. Poniżej przedstawiamy te elementy pracy, które są zrównoleglane.

Bloki zadań. Niech

$$B = (\pi(a), \pi(a+1), \ldots, \pi(b)), \tag{27}$$

będzie ciągiem bezpośrednio występujących po sobie zadań w permutacji $\pi \in \Phi$, π_k^* *wzorcem* dla k-tej maszyny oraz u, v ($u \neq v$, $1 \leqslant u, v \leqslant n$) parą liczb takich, że:

W1: $\pi(a) = \pi^*(u), \pi(a+1) = \pi^*(u+1), \ldots, \pi(b-1) = \pi^*(v-1)$,
$\pi(b) = \pi^*(v)$, lub

W2: $\pi(b) = \pi^*(u), \pi(b-1) = \pi^*(u+1), \ldots, \pi(a+1) = \pi^*(v-1)$, $\pi(a) = \pi^*(v)$

W3: B jest maksymalnym podciągiem ze względu na zawieranie, tj. nie można go powiększyć ani o element $\pi(a-1)$, ani o $\pi(b+1)$, spełniającym ograniczenia **W1** lub **W2**),

Jeżeli ciąg zadań (27) spełnia warunki **W1** i **W3** lub **W2** i **W3**, to nazywamy go *blokiem* na k-tej maszynie ($k \in \mathcal{M}$).

Poniżej przedstawiamy algorytm sekwencyjny wyznaczania wszystkich bloków w permutacji.

Algorytm 1. ***Alg_SeqBlock***

$\pi = (\pi(1), \pi(1), \ldots, \pi(n))$ - permutacja;
$\pi^* = (\pi^*(1), \pi^*(1), \ldots, \pi^*(n))$ - wzorzec permutacji π;
t - liczba bloków;
$(b_1, b_2, \ldots, b_t)$ - wektor pozycji początkowych bloków w π;
$t \leftarrow 1$; $i \leftarrow 1$;
while $(i \leqslant n)$ **do**
  $b_t \leftarrow i$; $q \leftarrow (\pi^*)^{-1}(\pi(i))$;
  while $(\pi(i) = \pi^*(i))$ **do**
    $q \leftarrow q + 1$; $i \leftarrow i + 1$;
    $i \leftarrow i + 1$;

Złożoność obliczeniowa algorytmu wynosi $O(n)$.

Wyznaczanie wzorca (optymalnej drogi komiwojażera w grafie H_k) jest problemem NP-trudnym. Dlatego będziemy stosowali algorytmy przybliżone, np. 2-opt. Dla każdej maszyny wzorzec należy wyznaczyć raz, przed uruchomieniem właściwego algorytmu.

Równoległe wyznaczanie bloków. Aby przyspieszyć działanie algorytmu wyznaczania minimalnego czasu cyklu przedstawiamy metodę zrównoleglenia najbardziej czasochłonnej, wykonywanej w każdej iteracji, procedury wyznaczania bloków.

Własność 1 *Wyznaczenie bloków dla cyklicznego problemu przepływowego z przezbrojeniami można wykonać w czasie $O(\log n)$ na mn-procesorowej maszynie CREW PRAM.*

Sposób wyznaczania bloków prezentuje Algorytm 2.

Algorytm 2. ***Alg_pblocks***
Wejście: permutacje: $\pi = (\pi(1), \pi(1), \ldots, \pi(n))$
oraz $\pi^* = (\pi^*(1), \pi^*(1), \ldots, \pi^*(n))$
Wyjście: wektor pozycji początkowych bloków $(b_1, b_2, \ldots, b_k)$, k – liczba bloków oraz wektor pozycji końcowych bloków $(e_1, e_2, \ldots, e_k)$

$$\pi = (8, 10, 7, 4, 5, 6, 3, 1, 9, 2)$$

$$\pi^* = (9, 3, 1, 6, 10, 7, 4, 5, 2, 8)$$

$$k = 2,\ b = (2, 7),\ e = (5, 8)$$

$$B_1 = (10, 7, 4, 5),\ B_2 = (3, 1)$$

Krok 1. parfor $r \in \{1, 2, \ldots, p\}$ **do**
$\pi(0) := \pi(n+1) := \pi^*(0) := \pi^*(n+1) := -1;$
$(\pi^*)^{-1}(0) := (\pi^*)^{-1}(n+1) := -1;$
if (poprzednia pozycja w π jest identyczna jak w π^*, tj.
$\pi(r-1) = \pi^*((\pi^*)^{-1}(\pi(r)) - 1))$ **then**
$B_b[r] := 1;$
else
$B_b[r] := 0;$
if (następna pozycja w π jest identyczna jak w π^*, tj.
$\pi(r+1) = \pi^*((\pi^*)^{-1}(\pi(r)) + 1))$ **then**
$B_e[r] := 1;$
else
$B_e[r] := 0;$
Krok 2. parfor $r \in \{1, 2, \ldots, p\}$ **do**
if $((B_b[r] = 0)$ and $(B_b[r+1] = 1))$ **then**
$B_b[r] := 1;$
else
$B_b[r] := 0;$

$$B_b = (0, 1, 0, 0, 0, 0, 1, 0, 0, 0)$$

Krok 3. parfor $r \in \{1, 2, \ldots, p\}$ **do**
if $((B_e[r] = 0)$ and $(B_e[r-1] = 1))$ **then**
$B_e[r] := 1;$
else
$B_e[r] := 0;$

$$B_e = (0, 0, 0, 0, 1, 0, 0, 1, 0, 0)$$

Krok 4. Wyznaczyć sumy prefiksowe P elementów z tablicy B_b, tj.

$$\forall_{r \in \{1,2,\ldots,p\}} P[r] = \sum_{i=1}^{r} B_b[i]$$

$$B_b = (0, 1, 0, 0, 0, 0, 1, 0, 0, 0)$$

$$P = (0, 1, 1, 1, 1, 1, 2, 2, 2, 2)$$

Krok 5. parfor $r \in \{1, 2, \ldots, p\}$ **do**
if $(B_b[r] = 1)$ **then**
$b[P[r]] := r;$
if $(B_e[r] = 1)$ **then**
$e[P[r]] := r;$

$$B_e = (0, 0, 0, 0, 1, 0, 0, 1, 0, 0)$$

$$B_b = (0, 1, 0, 0, 0, 0, 1, 0, 0, 0)$$

$$P = (0, 1, 1, 1, 1, 1, 2, 2, 2, 2)$$

$$b = (2, 7),\ e = (5, 8)$$

Na Rysunku 6 przedstawiono implementację Algorytmu 2 w postaci procedury `pblocks` wyznaczania bloków za pomocą n procesorów z wykorzystaniem biblioteki zrównoleglania OpenMP. Algorytm ten został następnie uruchomiony na koprocesorze Intel Xeon Phi. Danymi wejściowymi są: rozmiar permutacji w której wyznaczane są bloki `n`, permutacja `pi` oraz permutacja – wzorzec `pi_ptr`. Tablice `b_b` oraz `b_e` zawierają po zakończeniu procedury pozycje początków i końców kolejnych bloków.

```
int pblocks(int n, int *pi, int *pi_ptr,
    int *b_b, int *b_e) {
  int *pi_ptr_ = new int[n+2];
  int *b = new int[n+1];
  int *b_ = new int[n+1];
  int *bk = new int[n+1];
  int *p = new int[n+2];
  int *p_ = new int[n+2];
  pi_ptr[0] = pi_ptr[n+1] = -1;
  zeros_pi(n, pi_ptr_);
  pi_ptr_[0] = pi_ptr_[n+1] = -1;
  #pragma omp parallel for
  for(int i=1; i<=n; ++i)
    pi_ptr_[pi_ptr[i]] = i;
  #pragma omp parallel for
  for(int i=1; i<=n; ++i)
  {
    if(pi[i-1]==pi_ptr[pi_ptr_[pi[i]] - 1])
      b[i] = 1; else b[i] = 0;
    if(pi[i+1]==pi_ptr[pi_ptr_[pi[i]] + 1])
      bk[i] = 1; else bk[i] = 0;
    p[i] = p_[i] = 0;
  }
  p[1] = b[1]; p[0] = 0;
  #pragma omp parallel for
  for(int i=1; i<=n; ++i)
    if(b[i] == 0 and b[i+1] == 1)
      b[i] = 1; else b[i] = 0;
  #pragma omp parallel for
  for(int i=n; i>=1; --i)
    if(bk[i] == 0 and bk[i-1] == 1)
      bk[i] = 1; else bk[i] = 0;
  for(int i=2; i<=n; ++i)
    p[i] = p[i-1] + b[i];
  int nblocks = 0;
  #pragma omp parallel for
  for(int i=1; i<=n; ++i)
    if(b[i] == 1)
      b_b[p[i]] = i;
    if(bk[i] == 1)
      b_e[p[i]] = i;
  #pragma omp parallel for
  for(int i=1; i<=n; ++i)
  {
    #pragma omp critical
    if(b_b[i] != 0 or b_e[i] != 0)
      nblocks++;
  }
  #pragma omp parallel for
  for(int i=1; i<=nblocks; ++i)
    if(b_b[i] == 0 and b_e[i] != 0)
      b_b[i] = 1;
    if(b_e[i] == 0 and b_b[i] != 0)
      b_e[i] = n;

  delete[] pi_ptr_; delete[] b; delete[] b_;
  delete[] p; delete[] p_;
  return nblocks;
}
```

Rysunek 6: Funkcja równoległego wyznaczania bloków.

Eksperymenty obliczeniowe. Równoległa procedura wyznaczania bloków została zaimplementowana w języku C++ z wykorzystaniem biblioteki OpenMP. Dane na potrzeby eksperymentów obliczeniowych (permutacje zadań) zostały wygenerowane losowo. Liczba elementów permutacji zmieniała się w zakresie od 10^3, aż do 10^7. Eksperymenty obliczeniowe zostały przeprowadzone środowisku obliczeniowym z pamięcią współdzieloną – koprocesorze Intel Xeon Phi 3120A (6GB, 1.1 GHz) pozwalającym na wykorzystanie 228 rdzeni.

Pierwsza faza eksperymentów miała na celu wykazanie efektywności działania (sekwencyjnego) mechanizmu bloków opartych na wzorcach. W tym celu procedurę wyznaczania bloków umieszczono w klasycznym algorytmie przeszukiwania z zabronieniami (*tabu search*, TS), z listą tabu o długości 7. Uruchomiono dwie wersje algorytmu - z blokami oraz bez, dla ustalonej liczby 1000 iteracji. Badano procentowy błąd względny (*Percentage Relative Deviation*, PRD) do rozwiązania referencyjnego otrzymanego

Tabela 8: Porównanie PRD do NEH dla algorytmu TS z blokami oraz bez – 1000 iteracji.

$n \times m$	$t[s]$	$t_B[s]$	PRD	PRD_B
20×5	2,2	0,8	-30,0	-32,7
20×10	3,0	0,9	-29,5	-31,6
20×20	4,7	1,2	-29,8	-30,7
50×5	35,7	17,7	-32,5	-34,1
50×10	48,4	22,0	-29,6	-34,5
50×20	73,9	28,3	-28,3	-31,8
100×5	292,1	181,8	-30,7	-36,7
100×10	391,9	203,8	-28,1	-33,6
100×20	604,7	266,4	-27,9	-31,6
200×10	3212,2	1791,1	-26,7	-32,9
200×20	5255,5	2497,7	-26,2	-31,1
Średnio	902,2	455,6	-29,0	-32,8

Tabela 9: Przyspieszenie procedury `pblocks`.

$\lambda^{(1)}$	$p = 2$	$p = 4$	$p = 6$	$p = 8$	$p = 10$
1	0,003	0,001	0,001	0,001	0,001
2	0,008	0,004	0,003	0,004	0,003
5	0,018	0,008	0,008	0,007	0,007
10	0,037	0,016	0,017	0,017	0,015
20	0,078	0,033	0,036	0,043	0,033
50	0,308	0,154	0,165	0,143	0,147
100	0,632	0,441	0,458	0,443	0,400
200	0,961	0,915	0,986	1,047	0,980
500	1,229	1,578	1,889	2,061	2,171
1000	1,309	1,943	2,498	2,773	3,024
2000	1,302	2,255	2,994	3,490	3,810
5000	1,366	2,478	3,431	4,160	4,880
10000	1,398	2,584	3,612	4,461	5,266

(1) $\lambda = n \cdot 10^3$, koprocesor Intel Xeon Phi 3120A

przy pomocy algorytmu NEH (Nawaz i in. [23]) dla danych testowych pochodzących z pracy Ruiz i Stützle [24]. Wyniki zamieszczone w Tabeli 8 wskazują, że przy prawie dwukrotnie krótszym czasie działania algorytmu otrzymano wyniki znacząco lepsze (32,8% poprawy względem NEH) w porównaniu do wersji algorytmu bez mechanizmu bloków (29,0% poprawy względem NEH).

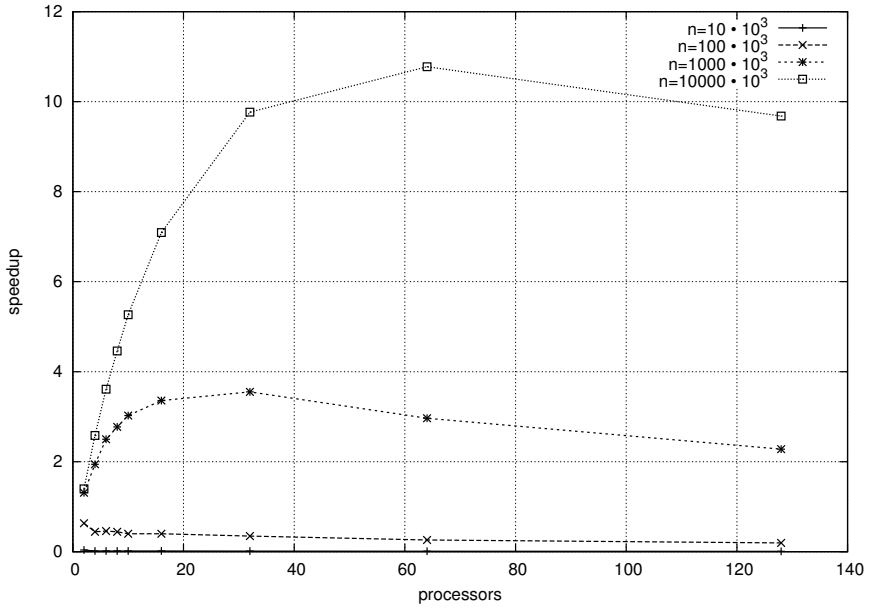
Druga faza eksperymentów polegała na zmierzeniu przyspieszenia procedury równoległego wyznaczania bloków. Wyniki eksperymentów obliczeniowych dla koprocesora Intel Xeon Phi 3120A zostały zamieszczone w tabelach 9 oraz 10 oraz zaprezentowane na Rys. 7 oraz 8. Dla różnej liczby zadań w permutacji przyspieszenie początkowo dość szybko wzrasta, osiąga maksimum, a następnie powoli zmniejsza się. Liczba procesorów dla którego osiągane jest maksimum przyspieszenia zależy od rozmiaru problemu. Posługując się pojęciem *skalowalności* algorytmów równoległych można powiedzieć, że dla $5000 \cdot 10^3$ zadań w permutacji równoległa metoda wyznaczania bloków charakteryzuje się dla $p = 1, 2, \ldots, 32$ silną skalowalnością, ponieważ wraz ze wzrostem liczby procesorów wzrasta przyspieszenie (Rys. 8).

Podsumowanie. W pracy [C3] przedstawiono nową koncepcję bloków dla cyklicz-

Tabela 10: Przyspieszenie procedury `pblocks`,

$\lambda^{(1)}$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
1	0,001	0,001	0,001	0,001
2	0,003	0,002	0,002	0,001
5	0,007	0,006	0,006	0,004
10	0,015	0,013	0,012	0,008
20	0,033	0,029	0,023	0,019
50	0,129	0,129	0,100	0,073
100	0,400	0,347	0,261	0,195
200	0,988	0,884	0,697	0,507
500	2,234	2,222	1,799	1,285
1000	3,360	3,552	2,967	2,278
2000	4,679	5,307	4,961	3,853
5000	6,306	8,270	8,154	7,043
10000	7,091	9,766	10,775	9,679

(1) $\lambda = n \cdot 10^3$, koprocesor Intel Xeon Phi 3120A

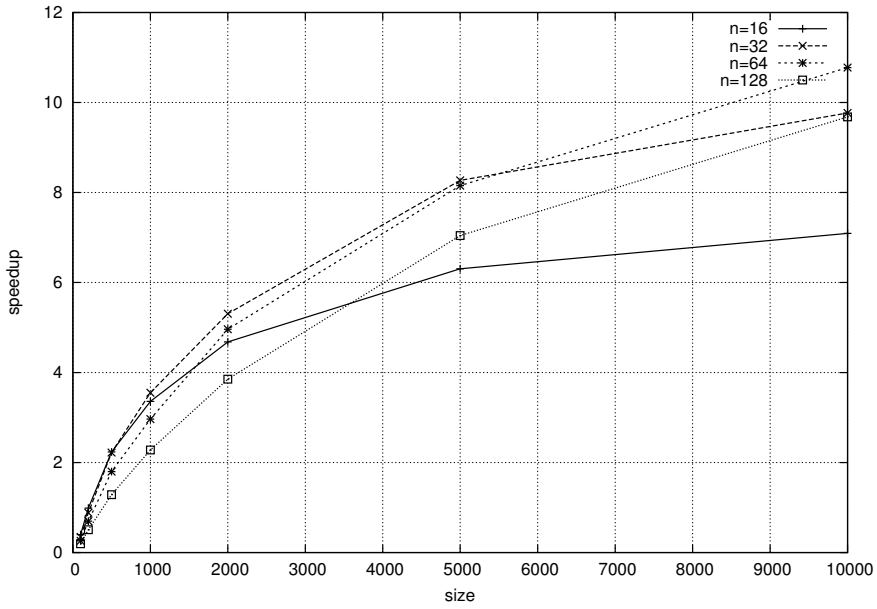


Rysunek 7: Zależność przyspieszenia od liczby procesorów – Intel Xeon Phi 3120A.

nego problemu przepływowego z przezbrojeniami maszyn, wykorzystującą wielokrotne wzorce o różnych rozmiarach. Wzorce reprezentują optymalną kolejności odwiedzania miast w pewnym problemie komiwojażera, zapewniając własności blokowe problemu. Zastosowanie tych własności blokowych pozwala na znaczną redukcję liczby przeglądanych sąsiadów w metaheurystycznych algorytmach opartych na przeglądaniu otoczeń.

8.4 Badanie rozkładu minimów lokalnych w przestrzeni rozwiązań

W pracy [C5] przedstawiono badania dotyczące metod konstrukcji nowych algorytmów rozwiązywania problemów optymalizacji dyskretnej bazujące na analizie krajobrazu



Rysunek 8: Zależność przyspieszenia od liczby zadań – Intel Xeon Phi 3120A.

przestrzeni[1] rozwiązywanej, konkretnej instancji problemu. Ogólną idee tej metody można przedstawić następująco.

W trakcie obliczeń obszar poszukiwań jest dynamicznie modyfikowany. Postawą zmian jest statystyczna analiza generowanych przez algorytm minimów lokalnych. Umożliwia ona ewentualną lokalizację tzw. *Wielkiej Doliny* (ang. *Big Valley*) i prowadzi do intensyfikacji obliczeń. Jeżeli okaże się jednak, że utknęliśmy w pewnym minimum lokalnym, wówczas stosujemy mechanizm powiększenia obszaru poszukiwań - dywersyfikację obliczeń. Reasumując, jest to iteracja metoda konstrukcji algorytmów składająca się z dwóch cykli: zewnętrznego i wewnętrznego. W cyklu zewnętrznym, na bazie historii obliczeń, wyznaczamy parametry obszarów poszukiwań (podzbiorów przestrzeni rozwiązań). W cyklu wewnętrznym wyznaczamy, przeszukując wyznaczone w cyklu zewnętrznym obszary, minima lokalne. Zbiór tych minimów stanowi dane do analizy w cyklu zewnętrznym. Dzięki temu możemy elastycznie zmieniać wielkość (promień) promień obszaru poszukiwań.

Eksperymenty obliczeniowe wykonane na przykładach problemu gniazdowego (ang. *job shop*), którego opis jest zamieszczony w pracy [C5]. Proces obliczeń algorytmu przebiegał następująco. Próbkując zbiór rozwiązań dopuszczalnych wyznaczamy histogram częstości występowania minimów lokalnych i na jego podstawie weryfikujemy hipotezę statystyczną dotyczącą rozkładu (normalnego) występowania tych minimów. Przeprowadzone eksperymenty obliczeniowe są obiecujące i stanowią inspiracje do prowadzenie dalszych w tym kierunku badań.

Opis metody. Rozważamy problem optymalizacyjny

$$\min\{F(\pi) : \pi \in \Phi\},$$

[1]Krajobraz przestrzeni, to trójka $(\mathcal{S}, F, d)$, gdzie $\mathcal{S}$ jest przestrzenią rozwiązań, F funkcją celu, a d miarą odległości pomiędzy (ang. *fitness landscape*).

gdzie Φ jest zbiorem permutacji elementów zbioru $\mathcal{J} = \{1, 2, \ldots, n\}$, a $F : \Phi \to \mathcal{R}^+$ funkcją celu (kryterium optymalizacji). Dla permutacji $\pi \in \Phi$, przez $\mathcal{M}(\pi)$ oznaczamy zbiór ruchów (przekształceń) permutacji. Dowolny ruch $m \in \mathcal{M}(\pi)$ transformuje π w permutację $m(\pi) \in \Phi$. Wówczas, zbiór permutacji

$$\mathcal{N}(\pi) = \{m(\pi) : \ m \in \mathcal{M}(\pi)\}.$$

jest otoczeniem permutacji π.

Dla ustalonego zbioru ruchów (generatora otoczeń) $\mathcal{M}(\pi)$, niech

$$G_\tau = (\Phi, \mathcal{A}; \tau) \tag{28}$$

będzie siecią, tj. grafem skierowanym z obciążonymi łukami, gdzie:

- Φ jest zbiorem wierzchołków,
- $\mathcal{A} = \{(\alpha, \beta) : \ \beta \in \mathcal{N}(\alpha), \ \alpha \in \Phi\}$ zbiorem łuków, a
- τ miarą (odległością) określoną na parach elementów Φ (wierzchołkach grafu G_τ).

Dla pewnego zbioru minimów lokalnych D rozpatrywanego problemu optymalizacyjnego konstruujemy graf z obciążonymi krawędziami $H_c = (D, U; c)$, gdzie

- zbiór krawędzi $U = \{\{u, v\} : \ u, v \in D, u \neq v\}$, a
- waga krawędzi $c(u, v)$ jest długością najkrótszej drogi z wierzchołka u do v w sieci G_τ (28).

Niech

- $d(u, v)$ - będzie długością najkrótszej drogi z wierzchołka u do v w grafie H oraz
- $d_u = \max\{d(u, v) : \ v \in U\}$.

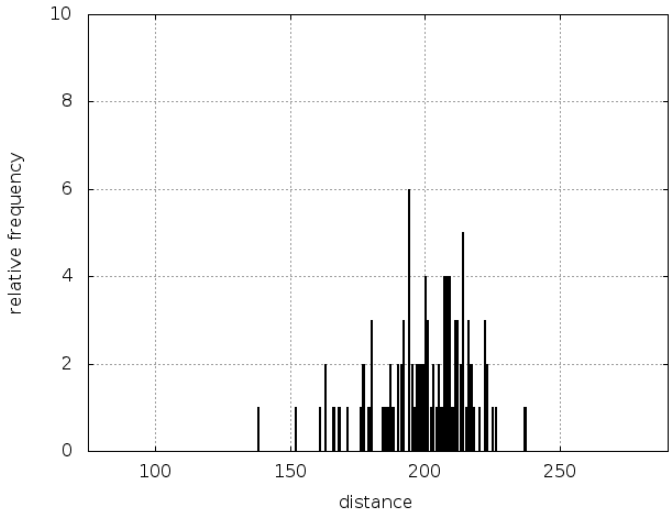
Wielkość

$$d^* = \min\{d_u : \ u \in U\}$$

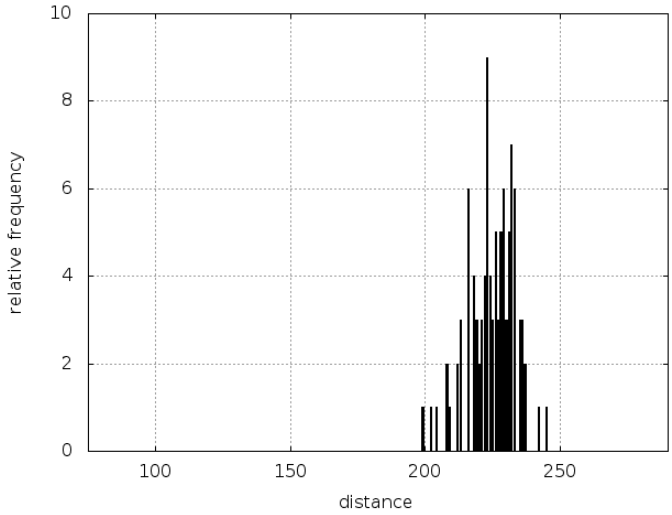
jest *promieniem* grafu H_c, a wierzchołek $v \in D$ taki, że $d_v = d^*$ *wierzchołkiem centralnym* w grafie H_c.

W celu zbadania rozkładu minimów lokalnych w przestrzeni rozwiązań, przeprowadzono wstępne obliczenia. Dla przykładu Tai11 (Taillard [21]) wyznaczono zbiór minimów lokalnych D. Do rozwiązywania tego przykładu wykorzystano równoległą implementację algorytmu TSAB, bazującą na metodzie przeszukiwania z tabu, której opis wersji sekwencyjnej jest zamieszczony w pracy [16]. Wielokrotnie uruchamiano ten algorytm z różnych, losowo wyznaczanych, punktów startowych. Najlepsze z wyznaczonych rozwiązań (minimów lokalnych) przyjęto za elementy zbioru minimów lokalnych D. Następnie, zbadano położenie tych minimów względem:

- rozwiązania suboptymalnego, tj. najlepszego obecnie znanego rozwiązania tego przykładu,
- centrum grafu H_c.



Rysunek 9: Rozkład minimów lokalnych, dla przykładu `ta11`, względem najlepszego znanego rozwiązania suboptymalnego.



Rysunek 10: Rozkład minimów lokalnych, dla przykładu `ta11`, względem wierzchołka centralnego grafu H_c.

Za miarę odległości pomiędzy permutacjami zbioru $H(\mathcal{D})$ przyjęto miarę Cayleya. Na rysunku 9 i 10, przedstawiono, dla przykładu Tai11, wykres częstości występowania minimów. Na rysunku 9 względem centrum c^* grafu H_c, a na rysunku 10, względem rozwiązania suboptymalnego (na osi X oznaczono odległość, a na Y - liczbę minimów).

Na rysunku 10 niemal wszystkie minima są w odległości pomiędzy 200 i 250. Ponadto, wykres ten można dość dokładnie aproksymować funkcją gęstości rozkładu normalnego. Natomiast, na rysunku 9 rozkład częstości minimów nie jest już tak regularny. Eksperymenty obliczeniowe przeprowadzone także na wielu innych przykładach potwierdzają tę obserwację, tj. wykresy częstości występowania minimów, w stosunku do centrum grafu, są podobne do przedstawionych na rysunku 10. Na tej podstawie przyjęto hipotezę, że

"*odległości pomiędzy minimami lokalnymi, a rozwiązaniem optymalnym mają rozkład normalny* $\Theta \sim N(\mu, \sigma)$", gdzie μ, σ są odpowiednio średnią i odchyleniem standardowym z odległości minimów lokalnych od centrum grafu.

Na podstawie analizy wyników dodatkowo przeprowadzonych eksperymentów można stwierdzić, że średnia odległość minimów lokalnych od centrum jest znacząco mniejsza niż średnia odległość losowo wyznaczonych rozwiązań od centrum. Świadczy to pośrednio o istnieniu *Wielkiej Doliny* skupiającej minima lokalne. W związku z tym wierzchołek centralny przyjmujemy za punkt odniesienia względem którego będziemy wyznaczali obszar poszukiwań. Promień tego obszaru (maksymalnie odległe od centrum c^* rozwiązania) będzie zależny od odchylenia standardowego σ rozkładu odległości $\Theta \sim N(\mu, \sigma)$.

Przeszukiwanie Wielkiej Doliny.

Na bazie analizy wyników przeprowadzonych eksperymentów obliczeniowych przyjęto następujące założenia związane z hipotezą istnienia *Wielkiej Doliny*:

(a) minima lokalne są skoncentrowane w pewnym obszarze przestrzeni rozwiązań,

(b) zależność pomiędzy odległością od rozwiązania optymalnego, a liczbą minimów ma rozkład normalny.

Korzystając ze wstępnie wykonanych obliczeń, wyznaczamy *obszar poszukiwań* w zbiorze rozwiązań będącym obiecującym fragmentem *Wielkiej Doliny*. W zależności od potrzeb, właściwie dobieramy parametr δ determinujący promień poszukiwań $r = \delta \cdot \sigma$. Obszar ten możemy zmniejszać (intensyfikacja obliczeń) lub powiększać (dywersyfikacja obliczeń). Algorytm wyznaczania ograniczonego obszaru poszukiwań (fragment *Wielkiej Doliny*) można przedstawić następująco.

Algorytm IWD {*Identyfikacja Wielkiej Doliny*}

Krok 1: *Próbkowanie przestrzeni rozwiązań*

Wyznaczyć zbiór losowych rozwiązań startowych

$$\mathcal{P} = \{\pi_1, \pi_2, \ldots, \pi_n\}.$$

Krok 2: *Wyznaczanie minimów lokalnych*

Stosując algorytm typu popraw (np. TS, SW), startując

kolejno z rozwiązań ze zbioru $\mathcal{P}$, wyznaczamy zbiór minimów lokalnych

$$\mathcal{P}^* = \{\pi_1^*, \pi_2^*, \ldots, \pi_n^*\}.$$

Zbiór $\mathcal{P}^*$ jest losową próbką ze zbioru (wszystkich) minimów lokalnych.

Krok 3: *Wyznaczanie wierzchołka centralnego*

Wyznaczanie wierzchołka centralnego π_{cen}^* w zbiorze $\mathcal{P}^*$.

Krok 4: *Empiryczny rozkład minimów lokalnych - weryfikacja hipotezy*

Zweryfikować hipotezę:

H_0: elementy zbioru $\mathcal{P}^*$, ze względu na wybraną miarę odległości, mają rozkład normalny $N(\mu, \sigma)$.

H_1: rozkład odległości nie jest $N(\mu, \sigma)$.

if (hipoteza H_0 jest prawdziwa) **then**

[wyprowadź: μ, σ, π_{cen}^*]

else [nie znaleziono *Wielkiej Doliny*]

Wśród danych wejściowych algorytmu jest współczynnik istotności testu. Jego wartość ma znaczący wpływ na akceptację hipotezy H_0 w **Kroku 4** algorytmu. Wstępny obszar poszukiwań definiujemy następująco:

$$\mathcal{OP} = \{\beta : \ \tau(\pi_{cent}^*, \beta) \leqslant \sigma \cdot \delta\},$$

gdzie τ jest miarą na zbiorze rozwiązań Φ, $r = \sigma \cdot \delta$ średnią obszaru poszukiwań, a δ $(0 < \delta \leqslant 3)$ parametrem promienia. W zależności od potrzeb zbiór $\mathcal{OP}$ możemy zmniejszać lub powiększać zmieniając wartość parametru δ. Poniżej przedstawiono ideę algorytmu wyznaczania elementu suboptymalnego w ograniczonym obszarze poszukiwań (fragmencie *Wielkiej Doliny*).

Algorytm POO {*Przeszukiwani ograniczonego obszaru*}

- za punkt startowy π przyjąć wierzchołek centralny π_{cen}^*;
- ustalić wielkość promieniaposzukiwań $\theta = \delta \cdot \sigma$, gdzie $1 \leqslant \delta \leqslant 3$;

repeat

Step 1: wygenerować elementy otoczenia $\mathcal{N}(\pi)$ odległe od π o nie więcej niż θ;

Step 2: wyznaczyć najlepszy element $\pi_{\min}$ w otoczeniu $\mathcal{N}(\pi)$ i podstawić go za π;

Step 3: zmienić lub pozostawić wartość promienia θ;

until Kryterium zatrzymania.

Kryterium zatrzymania algorytmu może być: maksymalna liczba iteracji, czas obliczeń, wyznaczenie rozwiązania o pewnej wartości, itp. W procesie przeszukiwania zbioru $\mathcal{OP}$ można zastosować dowolną metaheurystykę bazującą na metodzie lokalnych przeszukiwań.

Eksperymenty komputerowe. Głównym celem przeprowadzonych eksperymentów obliczeniowych było sprawdzenie hipotezy dotyczącej istnienia Wielkiej Doliny oraz

sprawdzenia efektów jej przeszukiwania. Algorytm **INTS**, który powstał na na bazie zamieszczonego w pracy Bożejko i in. [7] algorytmu **NTS** został zaimplementowany w języku C++ i uruchomiony na klastrze BEM we Wrocławskim Centrum Sieciowo-Superkomputerowym. Obliczenia, wykonano na przykładach Taillarda dla problemu gniazdowego. Przykładowe wyniki są przedstawione w Tabeli 11.

W kolumnie pierwszej jest nazwa przykładu, w drugiej - rozmiar, gdzie n jest liczbą zadań, a m liczbą maszyn. W kolejnych dwóch kolumnach przedstawiono procentowe średnie względne odchylenie, w stosunku do najlepszego znanego rozwiązania, dla algorytmu NTS oraz różnych wartości parametru δ w algorytmie INTS. Dla każdego przykładów generowano 300 minimów lokalnych.

			lNTS		
problem	$n \times m$	NTS	$\delta = 1$	$\delta = 2$	$\delta = 3$
TA01-10	15×15	0.5344	0.6736	0.3662	0.5355
TA11-20	20×15	1.0371	1.1340	1.1454	1.0657
TA21-30	20×20	1.0101	0.9690	0.8529	0.9973
TA31-40	30×15	0.9911	0.7171	1.0652	0.9911
TA41-50	30×20	1.7772	1.9260	1.7074	1.7772
TA51-60	50×15	0.0915	0.0915	0.0915	0.0915
TA61-70	50×20	0.1479	0.2935	0.0747	0.1235
TA71-80	100×20	0.0089	0.0000	0.0089	0.0089
average		0.6998	0.7256	0.6640	0.6989

Tabela 11: Średni błąd względny w stosunku do najlepszych znanych rozwiązań.

Najlepsze rozwiązania otrzymano dla wartości parametru $\delta = 2$. Średnio są one gorsze (względnie) od najlepszych, obecnie znanych w literaturze, o około 0.66%. Należy podkreślić, że najlepsze rozwiązania wyznaczono stosując różne algorytmy i różne czasy obliczeń. Uzyskane wyniki są więc obiecujące i stanowią inspirację do prowadzenia dalszych badań w tym kierunku.

8.5 Optymalizacja z niepewnymi parametrami

W procesie rozwiązywania klasycznych problemów szeregowania zadań zakłada się, że wszystkie parametry są deterministyczne. W przypadku niepewnych danych są one modelowane za pomocą rozkładów zmiennych losowych lub liczb rozmytych. W przypadku badań nad nowymi problemami lub wykorzystanie nowych technologii, wiedza dotycząca danych statystycznych z przeszłości zwykle nie istnieje. Dlatego podejście oparte na wykorzystaniu liczb rozmytych jest w tym przypadku w pełni uzasadnione.

W pracy [C6] jest rozpatrywany ogólny problem gniazdowy z niepewnymi czasami wykonania operacji reprezentowanymi przez liczby rozmyte. Do jego rozwiązania zastosowano równoległy algorytm przeszukiwania z tabu. W jego konstrukcji wykorzystano własności eliminacyjne bloków.

Ogólny problem gniazdowy (ang. *Flexible Job Shop problem*, w skrócie **FJS**) można sformułować następująco:

Problem FJS: Dany jest zbiór zadań $\mathcal{J} = \{1, 2, \ldots, n\}$, które należy wykonać na maszynach ze zbioru $\mathcal{M} = \{1, 2, \ldots, m\}$. Istnieje rozbicie zbioru maszyn na typy, tj. podzbiory maszyn o tych samych własnościach funkcjonalnych. Zadanie jest ciągiem pewnych operacji. Każdą operację należy wykonać na odpowiedniego typu maszynie w ustalonym czasie. Problem polega na przydzieleniu zadań do maszyn z odpowiedniego typu oraz na wyznaczeniu kolejności wykonywania operacji na maszynach, aby zminimalizować czas wykonania wszystkich zadań. Muszą być przy tym spełnione ograniczenia:

(i) każde zadanie może być wykonywane jednocześnie tylko na jednej, odpowiedniego typu, maszynie,

(ii) żadna maszyna nie może wykonywać jednocześnie więcej niż jedno zadanie,

(iii) wykonywanie zadania nie może być przerwane,

(iv) zachowany musi być porządek technologiczny wykonywania operacji.

Niech $\mathcal{O} = \{1, 2, \ldots, o\}$ będzie zbiorem wszystkich operacji. Zbiór ten można rozbić na ciągi odpowiadające zadaniom, przy czym zadanie $j \in \mathcal{J}$ jest ciągiem o_j operacji, które będą kolejno wykonywane na odpowiednich maszynach (tj. w ciągu technologicznym). Operacje te są indeksowane liczbami $(l_{j-1} + 1, \ldots, l_{j-1} + o_j)$, gdzie $l_j = \sum_{i=1}^{j} o_i$ jest liczbą operacji pierwszych j zadań, $j = 1, 2, \ldots, n$, przy czym $l_0 = 0$, a $o = \sum_{i=1}^{n} o_i$.

Zbiór maszyn $\mathcal{M} = \{1, 2, \ldots, m\}$ można także rozbić na q podzbiorów maszyn tego samego typu (*gniazd*), przy czym i-ty ($i = 1, 2, \ldots, q$) typ $\mathcal{M}^i$ zawiera m_i maszyn, które są indeksowane liczbami $(t_{i-1} + 1, \ldots, t_{i-1} + m_i)$, gdzie $t_i = \sum_{j=1}^{i} m_j$ jest liczbą maszyn w pierwszych i typach, $i = 1, 2, \ldots, q$, przy czym $t_0 = 0$, a $m = \sum_{j=1}^{m} m_j$.

Operację $v \in \mathcal{O}$ należy wykonać w gnieździe $\mu(v)$, tj. na jednej z maszyn zbioru $\mathcal{M}^{\mu(v)}$ w czasie $p_{v,j}$, gdzie $j \in \mathcal{M}^{\mu(v)}$.

Niech

$$\mathcal{O}^k = \{v \in \mathcal{O} : \ \mu(v) = k\}$$

będzie zbiorem operacji wykonywanych w k-tym ($k = 1, 2, \ldots, q$) gnieździe. Ciąg zbiorów operacji

$$\mathcal{Q} = [\mathcal{Q}^1, \mathcal{Q}^2, \ldots, \mathcal{Q}^m],$$

takich, że dla każdego $k = 1, 2, \ldots, q$

$$\mathcal{O}^k = \bigcup_{i=t_{k-1}+1}^{t_{k-1}+m_k} \mathcal{Q}^i \ \text{ oraz } \ \mathcal{Q}^i \cap \mathcal{Q}^j = \emptyset, \ i \neq j, \ i, j = 1, 2, \ldots, m,$$

nazywamy *przydziałem operacji zbioru $\mathcal{O}$ do maszyn ze zbioru $\mathcal{M}$* (w skrócie **przydziałem operacji do maszyn**). Ciąg $[\mathcal{Q}^{t_{k-1}+1}, \mathcal{Q}^{t_{k-1}+2}, \ldots, \mathcal{Q}^{t_{k-1}+m_k}]$ jest ***przydziałem*** operacji do maszyn w i-tym gnieździe (w skrócie *przydziałem* w i-tym gnieździe).

Dowolne rozwiązanie dopuszczalne problemu FJS może być reprezentowane przez parę $(\mathcal{Q}, \pi(\mathcal{Q}))$, gdzie $\mathcal{Q}$ jest przydziałem operacji do maszyn, a $\pi(\mathcal{Q}) = (\pi_1, \pi_2, \ldots, \pi_m)$ jest ciągiem m permutacji wyznaczających kolejności wykonywania operacji na poszczególnych maszynach. Dla uproszczenia zapisu dowolne rozwiązanie dopuszczalne będzie

oznaczane przez $\Theta = (\mathcal{Q}, \pi)$. Niech Φ będzie zbiorem wszystkich rozwiązań dopuszczalnych problemu FJS.

Reprezentacja grafowa rozwiązania. Dowolne rozwiązanie $\Theta = (\mathcal{Q}, \pi(\mathcal{Q})) \in \Phi$ (gdzie $\mathcal{Q}$ jest przydziałem operacji do maszyn, a $\pi(\mathcal{Q})$ kolejnością wykonywania operacji na każdej z maszyn) problemu FJS możne być przedstawione w postaci grafu skierowanego (digrafu) z obciążonymi wierzchołkami (sieci) $G(\Theta) = (\mathcal{V}, \mathcal{R} \cup \mathcal{E}(\Theta))$, gdzie $\mathcal{V}$ jest zbiorem wierzchołków, a $\mathcal{R} \cup \mathcal{E}(\Theta)$ zbiorem łuków, przy czym:

1) $\mathcal{V} = \mathcal{O} \cup \{s, c\}$, gdzie s i c są dodatkowymi operacjami reprezentującymi odpowiednio „start" i „zakończenie". Wierzchołek $v \in \mathcal{V} \setminus \{s, c\}$ ma dwie cechy:

   - $\lambda(v)$ – numer maszyny na której należy wykonać operację $v \in \mathcal{O}$,
   - $p_{v,\lambda(v)}$ – wagę wierzchołka równą czasowi wykonywania operacji v na maszynie $\lambda(v),\ v \in \mathcal{O}$.

   Wagi dodanych wierzchołków $p_s = p_c = 0$.

2) $\mathcal{R} = \bigcup_{j=1}^{n} \left[\bigcup_{i=1}^{o_j - 1} \{(l_{j-1} + i, l_{j-1} + i + 1)\} \cup \{(s, l_{j-1} + 1)\} \cup \{(l_{j-1} + o_j, c)\} \right].$

   Zbiór $\mathcal{R}$ zawiera łuki łączące kolejne operacje tego samego zadania oraz łuki z wierzchołka s do pierwszej operacji każdego zadania i łuki od ostatniej operacji każdego zadania do wierzchołka c.

3) $\mathcal{E}(\Theta) = \bigcup_{k=1}^{m} \bigcup_{i=1}^{|\mathcal{O}^k|-1} \{(\pi_k(i), \pi_k(i+1))\}.$

   Z kolei łuki ze zbioru $\mathcal{E}(\Theta)$ łączą operacje wykonywane na tej samej maszynie (π_k jest permutacją operacji wykonywanych na maszynie M_k, tj. operacji ze zbioru $\mathcal{O}^k$).

Łuki ze zbioru $\mathcal{R}$ wyznaczają kolejność wykonywania operacji w zadaniach (porządek technologiczny), a łuki ze zbioru $\mathcal{E}(\Theta)$ kolejność wykonywania operacji na każdej z maszyn.

Własności eliminacyjne bloków. Łatwo zauważyć, że czas zakończenia wykonywania wszystkich operacji $C_{\max}(\Theta)$, zgodnie z przydziałem operacji $\mathcal{Q}$ i kolejnością (uszeregowaniem) $\pi(\mathcal{Q})$, jest równy długości $L(s, c)$ drogi krytycznej (najdłuższej) $C(s, c)$ w grafie $G(\Theta)$ z wierzchołka s do c. Rozwiązanie problemu FJS sprowadza się więc do wyznaczenia takiego rozwiązania dopuszczalnego $\Theta = (\mathcal{Q}, \pi(\mathcal{Q})) \in \Phi$, dla którego odpowiadający mu graf $G(\Theta)$ ma najkrótszą drogę krytyczną, tj. minimalizuje $L(s, c)$. Problem ten jest *silnie NP-trudny.*

Niech $C(s, c) = (s, v_1, v_2, \ldots, v_w, c)$, gdzie $v_i \in \mathcal{O}$ $(1 \leqslant i \leqslant w)$ będzie drogą krytyczną w grafie $G(\Theta)$ z wierzchołka początkowego s do końcowego c. Drogę tę można podzielić na podciągi wierzchołków

$$\mathcal{B} = [B^1, B^2, \ldots, B^r],$$

zwane **blokami** w permutacji na drodze krytycznej $C(s, c)$, przy czym

(a) blok jest podciągiem wierzchołków z drogi krytycznej zawierającym kolejne operacje wykonywane bezpośrednio jedna po drugiej,

(b) blok zawiera operacje wykonywane na tej samej maszynie,

(c) przekrój dwóch dowolnych bloków jest zbiorem pustym.

(d) blok jest maksymalnym (ze względu na zawieranie) podzbiorem operacji z drogi krytycznej spełniającym ograniczenia (a)-(c).

Jeżeli B^k $(k = 1, 2, ..., r)$ jest blokiem na maszynie M_i $(i = 1, 2, ..., m)$ należącej do gniazda t $(t = 1, 2, ..., q)$, to będziemy go oznaczali następująco:

$$B^k = (\pi_i(a^k), \pi_i(a^k + 1), \ldots, \pi_i(b^k - 1), \pi_i(b^k)),$$

gdzie $1 \leqslant a^k \leqslant b^k \leqslant |Q^i|$. Operacje $\pi(a^k)$ i $\pi(b^k)$ w bloku B^k są nazwane odpowiednio *pierwszą* i *ostatnią*. Z kolei blok bez pierwszej i ostatniej operacji nazywamy *blokiem wewnętrznym*.

Twierdzenie 10 [C6] *Jeśli $\widehat{B}_k$ jest blokiem, to zmiana kolejności operacji z bloku wewnętrznego nie generuje rozwiązania o mniejszej wartości funkcji kryterialnej.* ■

Aby więc poprzez zmianę kolejności wykonywania operacji na maszynach wygenerować graf o mniejszej długości ścieżki krytycznej należy przynajmniej jedną operację z dowolnego bloku przestawić przed pierwszą lub za ostatnią operację tego bloku. Ta właściwość (tzw. *właściwość eliminacyjna bloku*) zostanie wykorzystana do wygenerowania otoczenia w algorytmie przeszukiwania z tabu. Z powyższego twierdzenia wynika, że poprawę wartości bieżącego rozwiązania można uzyskać jedynie poprzez:

1. przestawienie operacji z pewnego bloku wewnętrznego przed pierwszą lub ostatnią operacją tego bloku, lub

2. przeniesieniem operacji na inną maszynę, z tego samego gniazda.

Oba ruchy są dokładnie opisane w pracy [6] i będą stosowane do generowania otoczenia.

Rozmyte czasy wykonania operacji. Niech $p_{\upsilon j}$ $(j \in \mathcal{M}^{\mu(\upsilon)})$ będzie czasem wykonania operacji $\upsilon \in \mathcal{O}$ w gnieździe $\mu(\upsilon)$, czyli na jednej z maszyn ze zbioru $\mathcal{M}^{\mu(\upsilon)}$. Zakładamy, że niepewne czasy wykonania operacji są reprezentowane przez funkcję przynależności γ opisaną przez cztero elementowe ciągi liczb $\widehat{p}_{\upsilon,j} = (p_{\upsilon,j}^{min}, p_{\upsilon,j}^{med1}, p_{\upsilon,j}^{med2}, p_{\upsilon,j}^{max})$ takie, że:

- $(p_{\upsilon,j}^{\min} \leqslant p_{\upsilon,j}^{med1} \leqslant p_{\upsilon,j}^{med2} \leqslant p_{\upsilon,j}^{\max})$,
- $\gamma(x) = 0$ dla $x \leqslant p_{\upsilon,j}^{\min}$ lub $x \geqslant p_{\upsilon,j}^{\max}$,
- $\gamma(p_{\upsilon,j}^{med1}) = \gamma(p_{\upsilon,j}^{med2}) = 1$,
- γ rośnie w przedziale $[p_{\upsilon,j}^{min}, p_{\upsilon,j}^{med1}]$ i maleje w przedziale $[p_{\upsilon,j}^{med2}, p_{\upsilon,j}^{max}]$,

W tym przypadku funkcja przynależności

$$\gamma(x) = \begin{cases} \frac{x - p_{v,j}^{min}}{p_{v,j}^{med1} - p_{v,j}^{min}} & \text{dla } x \in [p_{v,j}^{min}, p_{v,j}^{med1}) \\ 1 & \text{dla } x \in [p_{v,j}^{med1}, p_{v,j}^{med2}) \\ \frac{p_{v,j}^{max} - x}{p_{v,j}^{max} - p_{v,j}^{med2}} & \text{dla } x \in [p_{v,j}^{med2}, p_{v,j}^{max}] \end{cases} \tag{29}$$

Graficzna reprezentacja funkcji $\gamma(x)$ ma kształt trapezu, dlatego reprezentowane przez nią liczba rozmyta jest nazywana *trapezową* liczbą rozmytą (w szczególności, gdy $p_{v,j}^{med1}, p_{v,j}^{med2}$ wówczas jest to *trójkątna* liczba rozmyta). W pracy Dubois [11] zdefiniowano operacji arytmetycznych na liczbach rozmytych oraz obliczanie maksimum/minimum z liczb rozmytych. Wynikiem są także liczby rozmyte. Stąd, jeżeli czasy wykonania operacji są liczbami rozmytymi, to także liczbami rozmytymi są momenty zakończenia wykonywania poszczególnych operacji. W algorytmach optymalizacyjnych są wielokrotnie porównywane pewne wartości (na przykład wartości funkcji kryterialnej) będące liczbami rozmytymi. Konieczna jest więc *defuzyfikacja*, tj. zamiana liczby rozmytej na liczbę rzeczywistą.

W pracy [C6] do defuzyfikacji liczby rozmytej zastosowano metodę *środka ciężkości*. Niech $\widehat{a}$ będzie liczbą rozmytą o funkcji przynależności $\gamma(x)$, $x \in \mathbb{R}$. Wynikiem defuzyfikacji $\widehat{a}$ jest liczba rzeczywista

$$COA(\widehat{a}) = \left\lceil \frac{\int x\gamma(x)\,\mathrm{d}x}{\int \gamma(x)\,\mathrm{d}x} \right\rceil. \tag{30}$$

Jeżeli $\widehat{a}$ jest trapezową liczbą rozmytą o funkcji przynależności

$$\gamma(x) = \begin{cases} \frac{x-a}{b-a} & \text{dla } x \in [a, b) \\ 1 & \text{dla } x \in [b, c) \\ \frac{d-x}{d-c} & \text{dla } x \in [c, d] \end{cases} \tag{31}$$

to w wyniku defuzyfikacji (30) otrzymujemy liczbę rzeczywistą:

$$COA(\widehat{a}) \;=\; \left\lceil \frac{\int x\gamma(x)\,\mathrm{d}x}{\int \gamma(x)\,\mathrm{d}x} \right\rceil = \left\lceil \frac{a^2 + ab + b^2 - c^2 - cd - d^2}{3(a + b - c - d)} \right\rceil. \tag{32}$$

Bloki dla problemu z rozmytymi czasami wykonywania operacji. Ponieważ czasy wykonania operacji i momenty ich zakończenia są liczbami rozmytymi więc, korzystając z defuzyfikacji, można wyznaczyć ścieżkę krytyczną $\widehat{C}(s, c)$ w grafie, a następnie rozbić ją na bloki

$$\widehat{B} = [\widehat{B}_1, \widehat{B}_2, \ldots, \widehat{B}_r]. \tag{33}$$

Wówczas, można skorzystać z własności blokowych, które umożliwiają eliminację gorszych rozwiązań w procedurze przeszukiwania otoczeń. Może to znacznie zmniejszyć liczbę elementów przestrzeni rozwiązań.

Eksperymenty obliczeniowe. Do rozwiązywania problemu FJS zastosowano algorytm przeszukiwania tabu z wykorzystaniem sąsiedztwa golfowego ([6]). W tym algorytmie, otoczenie jest generowany przez złożenie ruchów typu wstaw oraz transfer ([5]). Pierwszy - zmienia kolejność operacji na maszynie, natomiast drugi – przestawia

operację na inną maszynę (z tego samego gniazda). Równoległe algorytmu: deterministyczna pTSGPU i rozmyta pFzTSGPU zostały zaimplementowane w języku C++ z wykorzystaniem CUDA. Do komunikacji pomiędzy kartami obliczeniowymi GPU wykorzystano bibliotekę MPI. Eksperymenty obliczeniowe przeprowadzono na serwerze HP z procesorem Intel i7 (3,33 GHz) oraz procesorem graficznym Tesla S2050. Ich celem było zbadanie wpływu losowych zaburzeń danych (czasów wykonywania operacji) na zmiany wartości wyznaczonych rozwiązań. Jako miarę przyjęto *współczynnik stabilności* rozwiązania, który zdefiniowano następująco.

Dla ciągu deterministycznych czasów wykonywania operacji $\delta = [p_{v,j}]_{o\times 1}$, niech $D(\delta)$ będzie zbiorem danych generowanych z δ przez zaburzenie czasów polegające na zamianie elementów macierzy $\delta = [p_{v,j}]_{o\times 1}$ na generowane losowo wartości.

Niech $A \in \{TS, TSF\}$, gdzie TS i TSF są odpowiednio algorytmy deterministycznym i rozmytym, tj. dla danych reprezentowanych przez liczby rozmyte. Przez π_δ^A oznaczamy rozwiązanie (permutację) wyznaczone przez algorytm A dla danych δ. Wówczas, $C_{max(\pi_\delta^A,\varphi)}$ jest terminem zakończenia zadania, gdy są one wykonywane w kolejności π (wyznaczonej przez algorytm A) i φ jest macierzą czasów wykonywania operacji. Średni błąd względny dla zaburzonych danych ze zbioru $D(\delta)$ definiujemy następująco:

$$\Delta(A, \delta, D(\delta)) = \frac{1}{|D(\delta)|} \sum_{\varphi \in D} \frac{C_{max(\pi_\delta^A,\varphi)} - C_{max(\pi_\delta^{TS},\varphi)}}{C_{max(\pi_\delta^{TS},\varphi)}}. \tag{34}$$

Jeżeli Ω jest zbiorem przykładów deterministycznych, to *współczynnik stabilności* algorytmu A na zbiorze Ω

$$S(A, \Omega) = \frac{1}{|\Omega|} \sum_{\delta \in \Omega} \Delta(A, \delta, D(\delta)). \tag{35}$$

Im mniejsza wartość tego współczynnika, tym większa odporność rozwiązań wyznaczonych przez algorytm A na losowe zaburzenia danych.

Dla wyznaczenia współczynnika stabilności algorytmów: pTSGPU oraz pFzTSGPU rozwiązywania problemu FJS obliczenia wykonano dla trójkątnej i trapezoidalnej reprezentacji liczb rozmytych (czasów wykonywania operacji). Parametry liczb rozmytych generowano losowo, zgodnie z rozkładem jednostajnym (dokładny opis jest zamieszczony w pracy [C6]). Dla każdej grupy przykładów ustalono, zgodnie z (34), wartości (minimalnej, maksymalnej i średniej) parametrów Δ, które są przedstawione w tabeli 12. Współczynnik stabilności algorytmu pTSGPU wynosi $S(pTSGPU, \Omega) = 3.41\%$, natomiast algorytm pFzTSGPU – $S(pFzTSGPU, \Omega) = 3,45\%$. Stabilności obu algorytmów różnią się od siebie bardzo niewiele. Wynika to, m.in. z tego, że rozmyte wartości momentów zakończenia zadań (po defuzyfikacji) dla danych deterministycznych i rozmytych różnią się niewiele.

Jednym z dodatkowych celów przeprowadzonych eksperymentów obliczeniowych było zbadanie wpływu sposobu defuzyfikacji liczby rozmytej na stabilność rozwiązań. Okazało się, że różnice są niewielkie (mniej niż 1.5%). Jednak najbardziej odporne są rozwiązaniami, gdy do defuzyfikacji zastosowano zasadę „ostatnie z maksimum" ([C6]).

instance			*pTSGPU*			*pFzTSGPU*		
name	$n \times m$	*flex*	Δ_{min}	Δ_{aprd}	Δ_{max}	Δ_{min}	Δ_{aprd}	Δ_{max}
Mk01	10×6	2.09	0.315	0.618	0.861	0.268	0.528	0.775
Mk02	10×6	4.10	0.052	0.292	0.468	0.138	0.298	0.441
Mk03	15×8	3.01	0.086	0.271	0.598	0.165	0.396	0.554
Mk04	15×8	1.91	-0.039	0.350	0.625	0.210	0.396	0.584
Mk05	15×4	1.71	-0.037	0.006	0.069	-0.053	0.012	0.075
Mk06	10×15	3.27	0.725	0.860	1.052	0.506	0.942	1.164
Mk07	20×5	2.83	0.080	0.283	0.418	0.026	0.283	0.490
Mk08	20×10	1.43	0.103	0.161	0.232	0.120	0.230	0.350
Mk09	20×10	2.53	0.159	0.451	0.865	0.047	0.216	0.360
Mk10	20×15	2.98	-0.087	0.011	0.314	0.008	0.148	0.286

Tabela 12: Średni błąd względny dla zaburzonych danych.

8.6 Chemia obliczeniowa – dynamika molekularna

W ramach współpracy z dr hab. inż. Bartłomiejem Szyją z Katedry Chemii i Technologii Paliw (Wydział Chemiczny Politechniki Wrocławskiej) prowadzone były badania w obszarze automatyzacji i optymalizacji narzędzi do parametryzacji reaktywnego pola siłowego w obliczeniach z dziedziny chemii.

W ramach badań opracowana została metody dopasowywania parametrów ReaxFF dla układu składającego się z mieszanych tlenków metali przejściowych. Przetestowane zostało kilka metod pozwalających na obliczenie różnic między wektorami sił działających na atomy otrzymanymi z referencyjnej symulacji DFT, a zależnym od parametrów ReaxFF. Przeprowadzone eksperymenty numeryczne wskazały, że wykorzystanie miary odległość bezwzględna między dwoma wektorami (*ang. footrule distance*) daje najlepsze parametry spośród badanych opcji. Weryfikacja otrzymanych parametrów zrealizowana została przy użyciu systemu składającego się z obsługiwanych przez hematyt klastrów $(TiO_2)_n$. Wyniki wskazują, że przebudowane parametry pozwalają uzyskać akceptowalne geometrie klastrów po symulacji MD na poziomie ReaxFF i pomimo krótkiej skali czasu prowadzą do stabilnych struktur.

Wyniki badań zostały zawarte w artykule „Mixing ReaxFF parameters for transition metal oxides using force-matching method”[13] oraz zaakceptowane do publikacji w czasopiśmie „Journal of Molecular Modeling”.

8.7 Cyberbezpieczeństwo – wykrywanie anomalii w ruchu sieciowym

W ramach współpracy z Instytutem Technik Innowacyjnych EMAG (Sieć Badawcza Łukasiewicz) w ramach realizacji projektu RegSOC prowadzone były badania w zakresie wykrywania anomalii i ataków w monitorowaniu ruchu sieciowego za pomocą metod klasyfikacji i wytłumaczalnej sztucznej inteligencji (XAI).

[13]Włodarczyk A., Uchroński M., Podsiadły-Paszkowska A., Irek J., Szyja B. M., Mixing ReaxFF parameters for transition metal oxides using force-matching method, Journal of Molecular Modeling

Zapewnienie bezpieczeństwa ruchu w sieci jest bardzo ważną kwestią w wielu dzisiejszych branżach. Dlatego celem analiz był rozwój metod wykrywania anomalii i ataków. W wyniku przeprowadzonych badań zaproponowano podejście do monitorowania bezpieczeństwa ruchu w sieci oparte na klasyfikacji binarnej. Jako wspólną klasę potraktowano wiele typów ataków i przetestowano cztery klasyfikatory binarne. Po testach walidacji krzyżowej do dalszej analizy wybrano tylko dwie z czterech metod: regresji logistycznej i wzmacniania gradientu. Aby upewnić się, że tak zbudowane modele nie opierają się zbytnio na zmiennych liniowo zależnych, przeprowadzono również analizę korelacji 10 najważniejszych zmiennych. Wreszcie analiza XAI dotycząca natury znalezionych anomalii potwierdziła, że możliwe jest uzyskanie bardziej szczegółowych informacji o wewnętrznym typie anomalii.

Wyniki badań zostały zebrane w artykule „Detecting anomalies and attacks in network traffic monitoring with classification methods and XAI-based explainability"[14] zaprezentowanym na konferencji „Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 25th International Conference KES2021" oraz opublikowanym w czasopiśmie „Procedia Computer Science".

8.8 Inżynieria środowiska – badanie jakości powietrza wewnętrznego

W ramach współpracy z dr hab. Andrzejem Szczurkiem oraz dr hab. inż. Moniką Maciejewską z Katedry Klimatyzacji, Ogrzewnictwa, Gazownictwa i Ochrony Powietrza (Wydział Inżynierii Środowiska Politechniki Wrocławskiej) prowadzone były badania naukowe w obszarze zastosowania metod sztucznej inteligencji na potrzeby określania preferencji cieplnych na podstawie analizy zdarzeń. Badania zostały oparte o rzeczywiste dane pomiarowe jakości powietrza wewnętrznego - temperatura, wilgotność, stężenie CO_2 w pomieszczeniu biurowym WCSS.

Określenie optymalnych parametrów środowiska wewnętrznego jest niezbędne z punktu widzenia komfortu cieplnego użytkowników i oszczędności energii. Celem badań było opracowanie metody realizującej powyższy cel. Metoda opiera się na ciągłych pomiarach temperatury (T) i wilgotności względnej (RH) powietrza w pomieszczeniach oraz ich klasyfikacji. Klasyfikacja jest stosowana do wykrywania wzorców zmienności T i/lub RH, które wynikają ze stosowania środków powodujących celową zmianę warunków wewnętrznych. Występowanie tych wzorców wskazuje na okresy, w których warunki termiczne były pod kontrolą, tj. zdarzenia związane z ogrzewaniem lub chłodzeniem. Założono, że preferencje grupowe dotyczące warunków termicznych mogą być reprezentowane przez rozkład prawdopodobieństwa temperatury oraz rozkład prawdopodobieństwa wilgotności względnej w całkowitym czasie wykrytych zdarzeń. Badania skupione zostały na preferowanych warunkach chłodzenia pomieszczeń. Metoda została opracowana i zwalidowana na podstawie wyników monitoringu jakości powietrza w pomieszczeniach w okresie trzech sezonów letnich, tj. w latach 2016, 2017 i 2018 w biurze inżynierów IT. Porównano kilka wariantów klasyfikacji. Różniły się one: 1. parametrem powietrza w pomieszczeniu używanym do wykrywania zdarzeń (T, RH lub T

[14] Wawrowski Ł., Michalak M., Andrzej Białas A., Kurianowicz R. ,Sikora M., Uchroński M., Kajzer A. Detecting anomalies and attacks in network traffic monitoring with classification methods and XAI-based explainability, Procedia Computer Science, Vol. 192, 2021, 2259–2268

i RH), 2. wielkością fragmentów szeregów czasowych wykorzystywanych do ekstrakcji cech metodą szybkiej transformacji Fouriera (8, 16 i 32). min); 3. wielkością zbiorów danych wykorzystywanych do trenowania modelu klasyfikacyjnego (jeden sezon, dwa sezony); 4. rodzajem klasyfikatora (regresja logistyczna, autoenkoder, losowy las decyzyjny, maszyna wektorów nośnych (SVM)). Skuteczność metody oceniono na podstawie odległości Jeffreya między rozkładami prawdopodobieństwa T i RH związanymi z prawdziwymi i przewidywanymi zdarzeniami. Najlepsze wyniki osiągnięto dla wariantu klasyfikacji obejmującego: 1. wilgotność względną, 2. 16 min, 3. dwa sezony danych i 4. SVM. Metoda była bardzo dokładna dla rozkładów prawdopodobieństwa dotyczących temperatury i wilgotności względnej w okresach testowych. Dla poszczególnych sezonów letnich odległość Jeffreya dla rozkładów prawdopodobieństwa z temperaturą wynosiła od 0,0037 do 0,0321, a odległość Jeffreya dla rozkładów prawdopodobieństwa dla wilgotności względnej wynosiła od 0,0107 do 0,0158.

9 Analiza cytowań

Baza danych	H-index	Cytowania bez autocytowań	Cytowania łącznie
Web of Science Core Collection	6	128	140
Scopus	7	202	226
Google Scholar	8	–	321

Bibliografia

[1] Bożejko W., Chaczko Z., Uchroński M., Wodecki M., Parallel patterns determination in solving cyclic flow shop problem with setups, in: Special issue on Discrete Systems: Theory and Applications (Bozejko W., Wodecki M., Eds.), Archives of Control Sciences, Vol. 27, No. 2, 183–196 (2017)

[2] Bożejko W., Hejducki Z., Rogalska M., Wodecki M., Metody sprzężeń czasowych w inżynierii przedsięwzięć budowlanych, Materiały Budowlane nr 6, 11–12 (2016)

[3] Bożejko W., Rajba P., Uchroński M., Wodecki M., A Job Shop Scheduling Problem with Due Dates Under Conditions of Uncertainty, Proceedings of the International Conference on Computational Science, in: M. Paszynski et al. (Eds.): ICCS 2021, LNCS 12742, Springer, pp. 198-205 (2021)

[4] Bożejko W., Grabowski J., Wodecki M., Block approach-tabu search algorithm for single machine total weighted tardiness problem, Computers & Industrial Engineering 50, 1/2, 1–14 (2006)

[5] Bożejko W., Wodecki M., On the theoretical properties of swap multimoves. Oper. Res. Lett. 35(2), 227–231 (2007)

[6] Bożejko W., Uchroński M., Wodecki M., The new golf neighborhood for the flexible job shop problem. In: Proceedings of the ICCS 2010, Procedia Computer Science 1, pp. 289–296. Elsevier (2010)

[7] Bożejko W., Uchroński M., A Neuro-Tabu Search Algorithmfor the Job Shop Problem, L. Rutkowski et al (Eds.): Proceedings of the ICAISC 2010, Lecture Notes in Artificial Intellignece No. 6114, Springer, 387–394 (2010)

[8] Bożejko, W., Pempera J., Wodecki M., Parallel Simulated Annealing Algorithm for Cyclic Flexible Job Shop Scheduling Problem. Lecture Notes in Artificial Intelligence No. 9120, Springer, 603–612 (2015)

[9] Bożejko W., Uchroński M., Wodecki M., Block approach to the cyclic flow shop scheduling. Computers & Industrial Engineering, 81, 158–166 (2015)

[10] Brucker P., Burke E. K., Groenemeyer S., A branch and bound algorithm for the cyclic job-shop problem with transportation. Computers & Operations Research, 39, 12, 3200–3214 (2012)

[11] Dubois D., Prade H., Theorie des Possibilites. Applications a la representation des connaissances en informatique. Masson, Paris (1988)

[12] Flynn M. J., Very high-speed computing systems, Proceedings of the IEEE 54 (12), 1901–1909 (1966)

[13] Grabowski J., Wodecki M., A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Computers & Operations Research, 31, 1891–1909 (2004)

[14] Nowicki E., Smutnicki C., A Fast tabu serach algorithm for permutation flow shop problem, European Journal of Operational Research 91, 160–175 (1996)

[15] Nowicki E., Smutnicki C., A Fast tabu serach algorithm for the job shop problem, Management Scence 42, 797–813 (1996)

[16] Nowicki E., Smutnicki C., A Fast tabu serach algorithm for the job shop problem, Management Scence, 42, 797–813 (1996)

[17] OR-Library http://people.brunel.ac.uk/ mastjjb/jeb/info.html

[18] Potts C.N., Van Wassenhove L.N., Single Machine Tardiness Sequencing Heuristics, IIE Transactions, 23, 346–354 (1991)

[19] Ruiz R., Stutzle T., An iterated Greedy heuristic for the sequence dependet setup times flowshop problem with makespan and weighted tardiness objectives, EJOR, 187(3), 1143–1159 (2008)

[20] Smith W.E. Various Optimizers for Single-Stage Production, Naval Research Logist Quartely, 3, 59–66 (1956)

[21] Taillard E., Benchmarks for basic scheduling problems, European Journal of Operational Research, 64, 278–285 (1993)

[22] Uchroński M., Test instances for a single-machine total weighted tardiness scheduling problem, https://zasobynauki.pl/zasoby/51561

[23] Nawaz, M., Enscore, Jr, E.E., Ham, I. A heuristic algorithm for the m–machine, n–job flow–shop sequencing problem. OMEGA International Journal of Management Science, 11, 91–95 (1983)

[24] Ruiz, R., Stützle, T. An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European Journal of Operational Research, 187(3), 1143–1159 (2008)

[25] Voß, S. Tabu search: Applications and prospects. In Network Optimization Problems; Du, D.Z., Pardalos, P.M., Eds.; World Scientific Publishing Co.: Singapore, 333–353 (1993)

[26] Wolpert D.H., Macready W.G., No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, 67–82 (1997)