POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI
Instytut Informatyki

Zaawansowane metody optymalizacji

ANALIZA WYBRANYCH ODMIAN PROBLEMU SZEREGOWANIA ZADAŃ

Adam Łangowski, 147896
Jakub Sudoł, 147906

POZNAŃ 2024

Spis treści

Rozdział 1

Wstęp

1.1 Wprowadzenie

W dzisiejszych czasach, w dynamicznym otoczeniu produkcyjnym, efektywne zarządzanie procesami produkcyjnymi staje się kluczowym elementem osiągania konkurencyjnej przewagi oraz generowania oszczędności. Jednym z istotnych obszarów w tym kontekście jest problem szeregowania zadań w systemach maszyn dedykowanych. Niniejsza praca skupia się na analizie problemów szeregowania zadań typu Flow Shop oraz Open Shop, które należą do przepływowych systemów obsługi. W kontekście tych zagadnień, możliwe jest opracowanie efektywnych algorytmów, zarówno dokładnych, jak i heurystycznych, mających na celu optymalizację procesów produkcyjnych.

1.2 Cel i zakres pracy

Celem niniejszej pracy jest analiza, sformalizowanie wybranych problemów oraz przeprowadzenie eksperymentów obliczeniowych dla zaimplementowanych algorytmów, rozwiązujących problemy szeregowania zadań typu Flow Shop oraz Open Shop, na dwóch dedykowanych maszynach.

W rozdziale 2. przedstawiono niezbędne informacje dotyczące ogólnego problemu szeregowania zadań oraz jego własności. Omówiono także najważniejsze aspekty złożoności obliczeniowej w kontekście problemów szeregowania zadań.

Rozdział 3. prezentuje analizę problemu typu Flow Shop. Na początku zdefiniowano wszystkie założenia dla problemu oraz warunki rozwiązania przy pomocy programowania matematycznego. Ponadto, przedstawiono zaimplementowane algorytmy - metodę podziału i ograniczeń oraz Tabu Search, popartych wcześniejszym formalnym zapisem. Dla obu metod rozwiązania przeprowadzono następnie eksperymenty obliczeniowe, celem zbadania efektywności tych podejść. Rozdział zamyka podsumowanie przeprowadzonych testów oraz wnioski dotyczące danego problemu jak i zastosowanych metod jego rozwiązania.

Rozdział 4. opisuje definicję analizowanego problemu typu Open Shop, przy pomocy programowania matematycznego. Podobnie jak w problemie typu Flow Shop, zdefiniowano konieczne ograniczenia odnośnie instancji oraz rozwiązania.

W rozdziale 5. przedstawiono podsumowanie pracy.

Rozdział 2

Podstawy teoretyczne

2.1 Problemy szeregowania zadań

Problemy szeregowania zadań polegają na ustaleniu optymalnego harmonogramu (sekwencji) wykonywania różnych zadań przy uwzględnieniu pewnych ograniczeń. Optymalizacja harmonogramu może opierać się na minimalizacji różnych kryteriów, takich jak czas ukończenia wszystkich zadań, koszty czy zużycie zasobów.

Wyróżniamy 3 podstawowe odmiany szeregowania zadań na maszynach dedykowanych:

1. Job shop

Job shop to wariant szeregowania ogólny lub szeregowania zadań w systemie gniazdowym. Jest to problem sformułowany dla n zadań i m maszyn, w którym kolejność wchodzenia zadań na maszyny jest dowolna, ale z góry określona. Jeśli wejście zadania na maszynę nazwiemy operacją, to kolejność operacji ma znaczenie, podczas gdy kolejność maszyn jest dowolna.

2. Flow shop

Flow shop, lub inaczej problem szeregowania przepływowego, to problem dla n zadań, m maszyn, w którym kolejne zadania muszą zostać przetworzone przez każdą z maszynę w tej samej kolejności. Operacje nie mogą być wykonywane na maszynach równolegle.

3. Open shop

Open shop, zwany problemem szeregowania zadań w systemie otwartym, to problem dla n zadań, m maszyn, przy czym w przeciwieństwie do Job shop'a zarówno kolejność wchodzenia zadań (operacji) na maszyny jest dowolna i kolejne odwiedzane maszyny są dowolne.

W kontekście problemów szeregowania zadań istnieje kilka reguł priorytetu lub rozdziału, które pomagają w ustalaniu kolejności wykonywania zadań:

1. Bazujące na czasie gotowości:

- SPT (ang. Shortest Processing Time) - ta reguła przyznaje priorytet zadaniom o najkrótszym czasie trwania, czyli wybiera te, które można zrealizować w najkrótszym czasie. Jest to strategia minimalizująca czas trwania całego procesu.
- LPT (ang. Longest Processing Time) - przeciwieństwo SPT, w tej metodzie zawsze wybierane jest zadanie, które ma najdłuższy przetwarzania.

- LWR (ang. Least Work Remaining) - kolejność wykonania zadań ustala się według rosnących sum czasów przetwarzania zadań pozostałych do wykonania w zleceniach, czyli w danym zleceniu liczymy sumę czasów pozostałych zadań potrzebnych do wykonania całego zlecenia i wybieramy to z sumą mniejszą.

- LST (ang. Longest Slack Time) - przydziela priorytet zadaniom na podstawie najdłuższego czasu pozostałego do zakończenia, czyli 'slack time'. Jest to różnica między terminem gotowości zadania a jego terminem wykonania. Zadanie o największym 'slack time' otrzymuje najwyższy priorytet i jest planowane jako pierwsze.

2. Oparte na terminie wykonania:

- EDD (ang. Earliest Due Date) - reguła ta przyznaje priorytet zadaniom na podstawie terminu wykonania, przy czym zadanie o najwcześniejszym terminie ma pierwszeństwo. Jest szczególnie istotna w przypadku problemów, gdzie istnieją surowe terminy dostarczenia.

- LDD (ang. Latest Due Date) - w przeciwności do reguły EDD, ta reguła przyznaje priorytet zadaniom według najpóźniejszych terminów dostarczenia. Zadanie o najdłuższym czasie do wykonania jest planowane jako pierwsze. Jest to szczególnie użyteczne, gdy celem jest uniknięcie opóźnień w dostarczeniach.

W wizualizacji problemów szeregowania zadań stosuje się najczęściej wykresy Gantta, przedstawiające w zbiorczy sposób listę dostępnych maszyn oraz czasy wykonania danych zadań. Kolumny (oś X) są w tym wypadku użyte jako skala czasu.

Jedną z pierwszych efektywnych metod rozwiązywania problemów szeregowania zadań jest algorytm S.M. Johnsona w 1954 roku. W pierwszym kroku algorytm identyfikuje zadania, które wymagają przypisania do dwóch różnych maszyn. Następnie określa pewną funkcję kosztu dla każdego zadania, uwzględniając czas wykonania na obu maszynach. Zadania są sortowane według rosnącego kosztu. W drugim etapie algorytm przypisuje posortowane zadania do maszyn, zaczynając od tych o najniższym koszcie. Ostateczne przypisanie jest dokonywane tak, aby minimalizować całkowity czas trwania. Algorytm Johnsona jest szczególnie skuteczny w przypadku problemów szeregowania zadań na dwóch maszynach, gdzie różnice w czasie wykonania na poszczególnych maszynach są istotne.

2.2 Wybrane własności problemów szeregowania

W problemach szeregowania wyszczególnia się kilka dodatkowych własności.

Po pierwsze zadania mogą być podzielne lub niepodzielne. W drugim przypadku przerwy w wykonywaniu poszczególnych operacji na jednej maszynie są niedopuszczalne. Natomiast dla wariantu z zadaniami podzielnymi wykonywanie można przerwać i rozpocząć ponownie w innym momencie.

Możliwe jest także wprowadzenie obowiązkowych momentów zakończenia danych zadań - tak zwane *deadlines*. Najczęściej określają one moment, w którym zadanie musi zostać zakończone, lub czas potencjalnych opóźnień chcemy minimalizować. Powiązany

z wartością deadlinu parametr *lateness* jest wyrażany jako różnica momentu zakończenia zadania i jego deadlinu.

Kolejnym opcjonalnym parametrem są tak zwane momenty gotowości (ang. ready time). Określają one momenty w jakich poszczególne zadania stają się dostępne i można je zacząć przetwarzać na maszynach.

Wyszczególnionym i bliżej opisanym w rozdziale 3. warunkiem jest zasada *no-wait*. Warunek ten określa przymus rozpoczęcia przetwarzania zadania na drugiej maszynie bezpośrednio w momencie zakończenia jego przetwarzania na maszynie pierwszej.

Często spotykane w problemach szeregowania okresy niedostępności stanowią kolejny ważny punkt. Okres niedostępności oznacza zakres czasu, w którym maszyna nie może wykonywać żadnego zadania - jest ona wyłączona z użytku na czas trwania okresu niedostępności. Takie obowiązkowe przerwy można nałożyć na wszystkie maszyny, bądź jedynie wybrane. Okresy niedostępności najczęściej wiąże się z parametrem określającym maksymalny czas, po którym musi wystąpić kolejny okres niedostępności na danej maszynie.

Ostatnią, wartą wspomnienia, własnością jest tak zwany efekt uczenia (ang. learning effect). Efekt uczenia polega na skróceniu wykonania kolejnych zadań zamieszczonych w uszeregowaniu. Tempo, z jakim maszyna skraca przetwarzanie danych zadań, kontrolowane jest przez parametr zwany współczynnikiem uczenia.

Problemy szeregowania zadań można zwięźle przedstawić przy pomocy notacji trójpolowej. Wyróżnia się przy tym 3 parametry: typ szeregowania, cechy zadań oraz kryterium oceny. Precyzyjnemu opisowi danych problemów służy programowanie matematyczne, w którym poprzez wprowadzenie odpowiednich oznaczeń oraz ograniczeń możemy dokładnie zapisać wszystkie warunki, które musi spełnić rozwiązanie problemu.

2.3 Złożoność obliczeniowa

Złożoność obliczeniowa w kontekście problemów szeregowania zadań na dwóch maszynach odgrywa kluczową rolę w ocenie wydajności algorytmów. Takie problemy szeregowania należą do klasy problemów optymalizacyjnych NP-trudnych, co oznacza, że na stan obecny nie istnieje efektywny algorytm wielomianowy rozwiązujący ten problem dla dowolnych danych wejściowych. Do znalezienia rozwiązania optymalnego dla dowolnie dużej instancji musimy zastosować algorytm dokładny, którego złożoność jest wykładnicza. Dla takich metod czas obliczeń rośnie zdecydowanie szybciej niż wielkość instancji (w kontekście problemu szeregowania chodzi głównie o liczbę zadań) i bardzo szybko algorytm może okazać się mało efektywny. Dlatego też dla omawianych problemów szeregowania stosuje się często metody takie jak metaheurystyka, aby skutecznie radzić sobie z trudnościami obliczeniowymi / czasowymi i znaleźć rozwiązanie zbliżone do optymalnego w akceptowalnym czasie.

Rozdział 3

Problem typu Flow Shop

3.1 Formalny zapis problemu

Problem: F2 h_{T1T2} | no-wait | $C_{\max}$
Oznaczenia:

- n - liczba zadań,
- h - długość okresów niedostępności,
- τ - maksymalny czas pomiędzy okresami niedostępności,
- t_{li} - moment rozpoczęcia zadania na pozycji l na maszynie i
- p_{ji} - długość zadania j na maszynie i,
- s_{ki} - moment rozpoczęcia k-tego okresu niedostępności na maszynie i,
- $z_{jli} = \begin{cases} 1, & \text{jeżeli zadanie } T_{li} \text{ występuje,} \\ 0, & \text{jeżeli nie występuje.} \end{cases}$

Minimalizujemy C_{max}, przy ograniczeniach:

1. $\sum_{l=1}^{n} z_{jli} = 1 \quad : j = 1, 2, \ldots, n; \;\; i = 1, 2,$
2. $\sum_{l=1}^{n} z_{jli} = 1 \quad : l = 1, 2, \ldots, n; \;\; i = 1, 2,$
3. $z_{jli} \in \{0, 1\} \quad : j = 1, 2, \ldots, n; \;\; l = 1, 2, \ldots, n; \;\; i = 1, 2,$
4. $t_{1i} \geq 0 \quad : i = 1, 2,$
5. $t_{li} + \sum_{j=1}^{n} p_{j1} z_{jli} \leq s_{ki} \quad \text{lub} \quad t_{li} \geq s_{ki} + h \quad : i = 1, 2, \;\; l = 1, 2, \ldots, n,$
6. $t_{l1} + \sum_{j=1}^{n} p_{j1} z_{jl1} = t_{l2} \quad : l = 1, 2, \ldots, n,$

7. $t_{l+1i} \geq t_{li} + \sum_{j=1}^{n} p_{ji} z_{jli} \quad : l = 1, 2, \ldots, n,$

8. $s_{k+1i} - s_{ki} \leq \tau \quad : i = 1, 2,$

9. $k \leq \lceil \frac{n}{\tau} \rceil \quad : \tau > 0,$

10. $C_{\max} = t_{n2} + \sum_{j=1}^{n} p_{j2} z_{jl2}$

3.2 Algorytm dokładny

Pierwszym zaimplementowanym i przetestowanym algorytmem rozwiązującym problem opisany w punkcie 3.1, jest metoda podziału i ograniczeń (ang. branch and bound). W tej sekcji omówiona zostanie szczegółowo zasada działania algorytmu oraz opisane zostaną przeprowadzone eksperymenty obliczeniowe.

3.2.1 Implementacja metody podziału i ograniczeń

Schemat działania można przedstawić w postaci drzewa poszukiwań. Każdy węzeł drzewa odpowiada podzbiorowi rozwiązań dopuszczalnych danego problemu. W procesie poszukiwań, przegląda się poszczególne węzły drzewa (poczynając od korzenia) i ocenia, czy w podzbiorze rozwiązań, odpowiadającym danemu węzłowi, może znajdować się rozwiązanie optymalne. Jeśli nie, to węzły poddrzewa, poczynając od węzła analizowanego, są odrzucane z dalszego procesu poszukiwań. W przeciwnym przypadku, z danego węzła generowane są kolejne węzły. Proces przeszukiwania drzewa trwa tak długo, aż dokonamy bezpośredniego lub pośredniego przeglądu wszystkich węzłów w drzewie (tzn., dopóki wszystkie możliwe rozwiązania problemu nie zostaną sprawdzone bądź odrzucone).

Działanie utworzonego algorytmu branch and bound można opisać w następujących krokach:

1. Inicjalizacja:

Tworzymy strukturę reprezentującą węzeł w drzewie przeszukiwań. Ustalamy poziom w drzewie, początkowy makespan, dolne ograniczenie, harmonogram oraz informację o odwiedzonych zadaniach.

2. Dodawanie do drzewa:

Dodajemy węzeł do drzewa przeszukiwań jako punkt startowy. Ustawiamy węzły w kolejce priorytetowej według ich dolnych ograniczeń.

3. Przeszukiwanie drzewa:

Iteracyjnie pobieramy węzły z kolejki priorytetowej. Dla każdego węzła sprawdzamy, czy jest liściem. Jeśli tak, aktualizujemy najlepsze uszeregowanie i makespan. Jeśli węzeł nie jest liściem, generujemy potomków przez dodanie kolejnych zadań do uszeregowania.

4. Obliczenia:

Dla każdego węzła obliczamy dolne ograniczenie, uwzględniając pozostałe czasy przetwarzania, czasy przerw i ilość przerw na obu maszynach. Makespan dla uszeregowania danego węzła obliczamy uwzględniając przerwy na obu maszynach.

5. Warunek no-wait:

Weryfikujemy, czy zadanie może być dodane bez oczekiwania (no-wait). Jeśli nie, to zwiększamy makespan o czas oczekiwania.

6. Przycinanie drzewa:

Przycinamy poddrzewa, które mają dolne ograniczenie gorsze niż aktualny najlepszy makespan. Optymalizacja ta pomaga w redukcji liczby rozważanych możliwości.

7. Aktualizacja najlepszego wyniku:

Jeśli makespan danego węzła jest lepszy niż aktualnie najlepszy, aktualizujemy najlepszy harmonogram i makespan.

8. Kontynuacja przeszukiwania:

Kontynuujemy przeszukiwanie drzewa, aż kolejka priorytetowa nie będzie pusta.

9. Wynik:

Po zakończeniu przeszukiwania drzewa, prezentujemy najlepszy harmonogram oraz makespan.

Algorytm ten wykorzystuje strategię podziału i ograniczeń, eliminując pewne gałęzie drzewa przeszukiwań w przypadku, gdy nie mają szans na uzyskanie lepszego wyniku niż już znalezione rozwiązanie. Kolejność przetwarzania węzłów jest determinowana ich dolnymi ograniczeniami, co pozwala na skoncentrowanie się na bardziej obiecujących obszarach przestrzeni rozwiązań. Przy obliczaniu dolnego ograniczenia uwzględnia się aktualny makespan, pozostałe czasy przetwarzania oraz maksymalny czas potrzebny na ukończenie pozostałych zadań na obu maszynach i minimalną liczbę przerw wymaganą dla tych zadań. Program umożliwia wprowadzenie danych przez użytkownika lub wykonanie serii testów z danymi zadaniami losowymi i określonymi wybranymi parametrami.

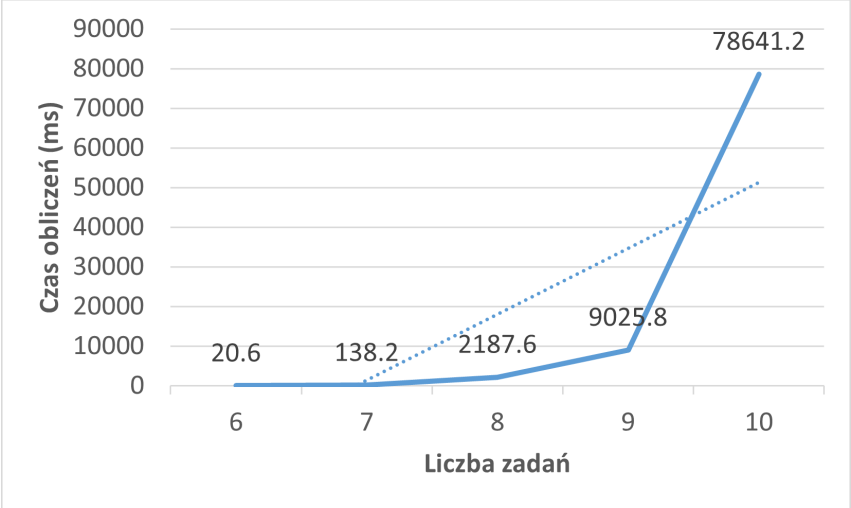
3.2.2 Eksperymenty obliczeniowe dla algorytmu dokładnego

Dla zaimplementowanego algorytmu dokładnego przeprowadzono szereg testów, sprawdzając przełożenie danych parametrów na czas obliczeń oraz osiągany C_{max}. Tabela 3.1 przedstawia założone na początku parametry domyślne. W każdej próbie (powtarzanej 5 razy) zmieniano wartość jednego z elementów, pozostawiając resztę domyślną.

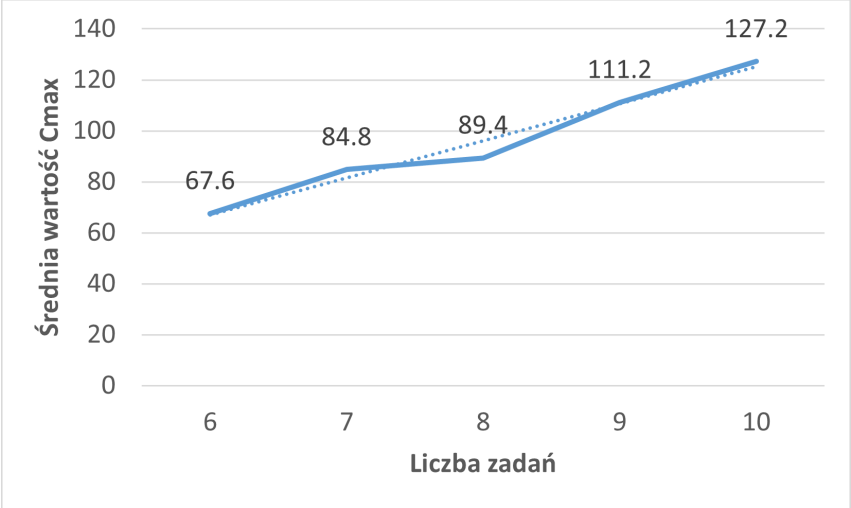
Liczba zadań	**Długość okresu niedostępności**	**Maksymalny czas pomiędzy okresami niedostępności**	**Maksymalna długość operacji**
7	3	20	15

Tabela 3.1: Parametry domyślne, ustawione podczas testów.

1. Liczba zadań



Rysunek 3.1: Zależność liczby zadań od czasu obliczeń.

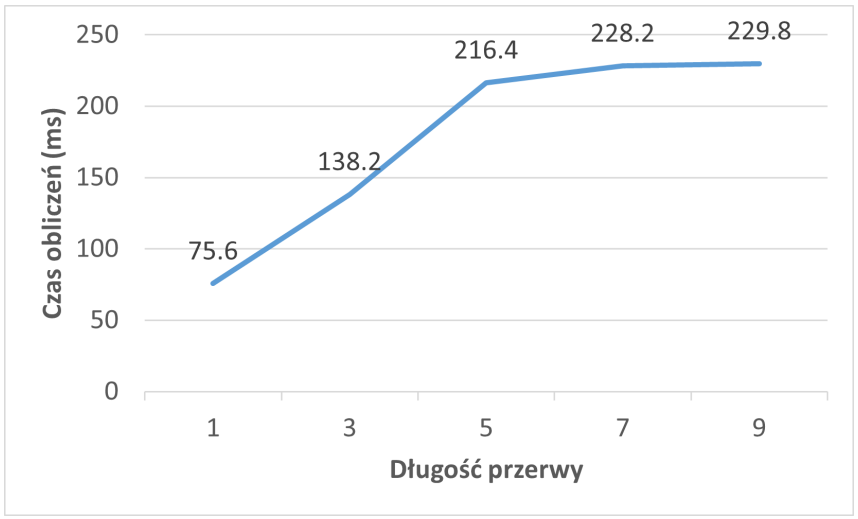


Rysunek 3.2: Zależność liczby zadań od uzyskiwanego C_{max}.

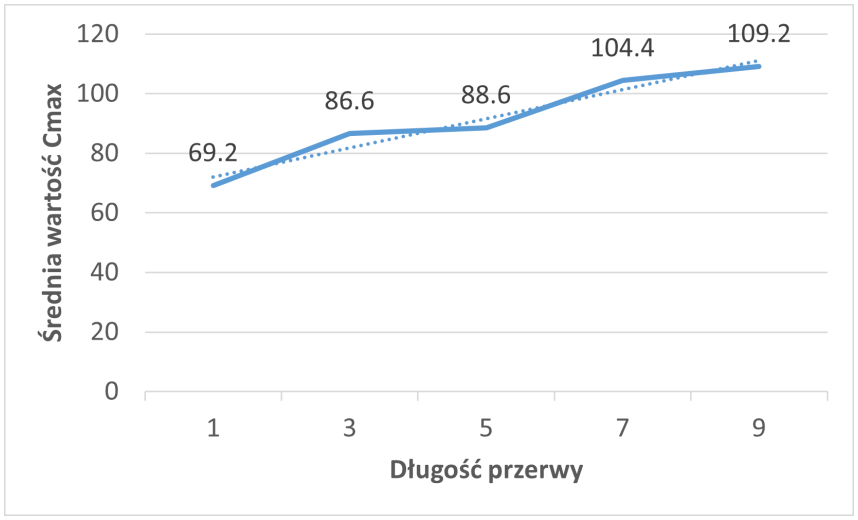
Jak widać na Rysunku 3.1 liczba zadań ma bardzo duże przełożenie na łączny czas obliczeń, potrzebny do znalezienia optymalnego rozwiązania. Analizując 9 zadań algorytm pracował średnio około 9 sekund, natomiast przy dziesięciu zadaniach średni czas eksperymentu wynosił już ponad jedną minutę. Jak można było zakładać, średnia wartość C_{max} rośnie niemal liniowo wraz z dodaniem kolejnych zadań, co ukazuje Rysunek 3.2.

2. Długość okresu niedostępności

Długość okresu niedostępności okazuje się mieć dość duży wpływ na czas obliczeń, co prezentuje Rysunek 3.3. Prawdopodobnie dłuższe okres niedostępności wprowadzają większą ilość możliwych kombinacji uszeregowań, które algorytm musi rozważyć, co z kolei zwiększa czas potrzebny na ich przeszukanie. Znaczenie może mieć także wa-



Rysunek 3.3: Zależność długości okresów niedostępności (przerw) od czasu obliczeń.

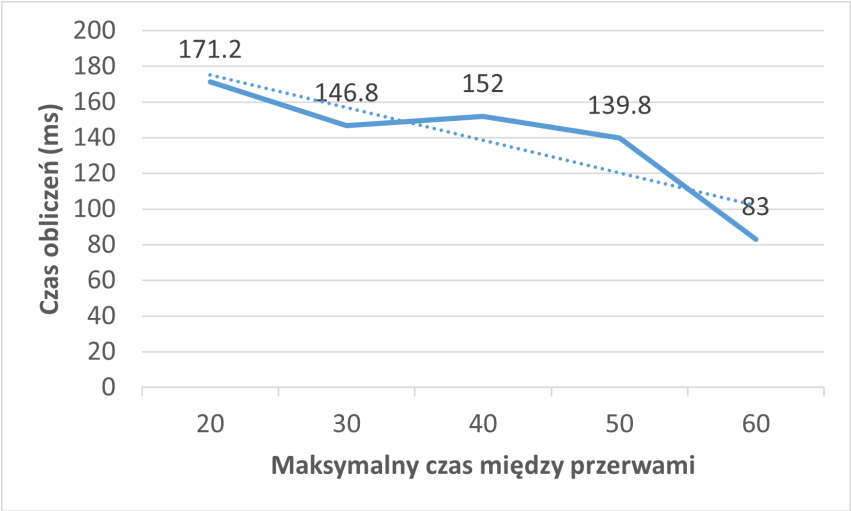


Rysunek 3.4: Zależność długości okresów niedostępności (przerw) od uzyskiwanego C_{max}.

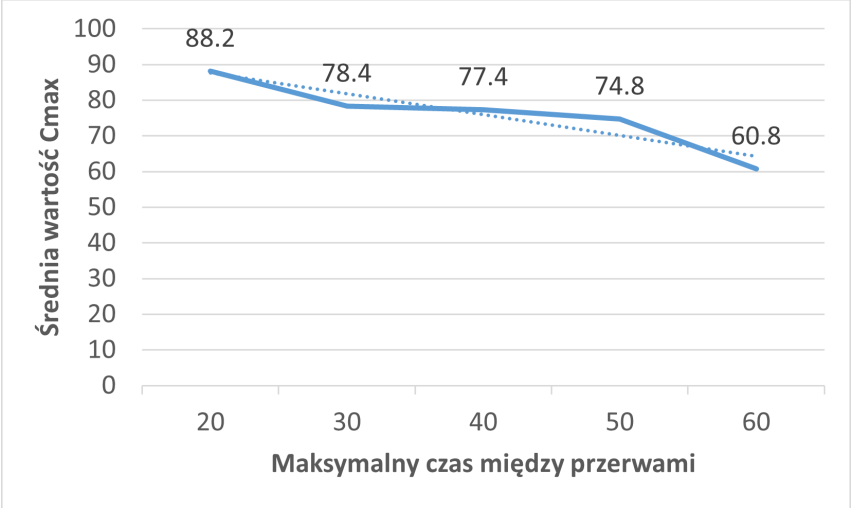
runek no-wait oraz funkcja dolnego ograniczenia, która może stać się bardziej złożona w przypadku dłuższego okresu niedostępności. Jak można się domyślać, dłuższe okresy niedostępności przekładają się także na otrzymywany całkowity makespan, co potwierdza Rysunek 3.4.

3. Maksymalny czas między okresami niedostępności

   Maksymalny czas pomiędzy kolejnymi okresami niedostępności w obrębie jednej maszyny (parametr *tau*) okazuje się mieć niemałe przełożenie na łączny czas obliczeń. Jak prezentuje Rysunek 3.5, im większy jest parametr *tau* tym mniej algorytm potrzebuje czasu na znalezienie rozwiązania dokładnego. Dzieje się tak ponieważ algorytm musi w uszeregowaniu umiejscowić mniejszą liczbę przerw (okresów niedostępności), co limituje możliwości potencjalnych uszeregowań, które metoda musi wziąć pod uwagę. Zwiększenie parametru *tau* przekłada się także pozytywnie na



Rysunek 3.5: Zależność maksymalnego czasu między okresami niedostępności od czasu obliczeń.

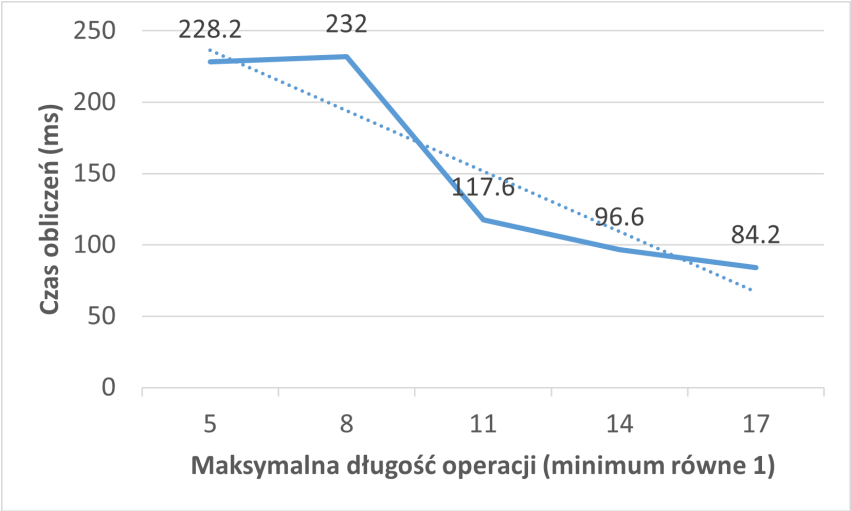


Rysunek 3.6: Zależność maksymalnego czasu między okresami niedostępności od uzyskiwanego C_{max} .

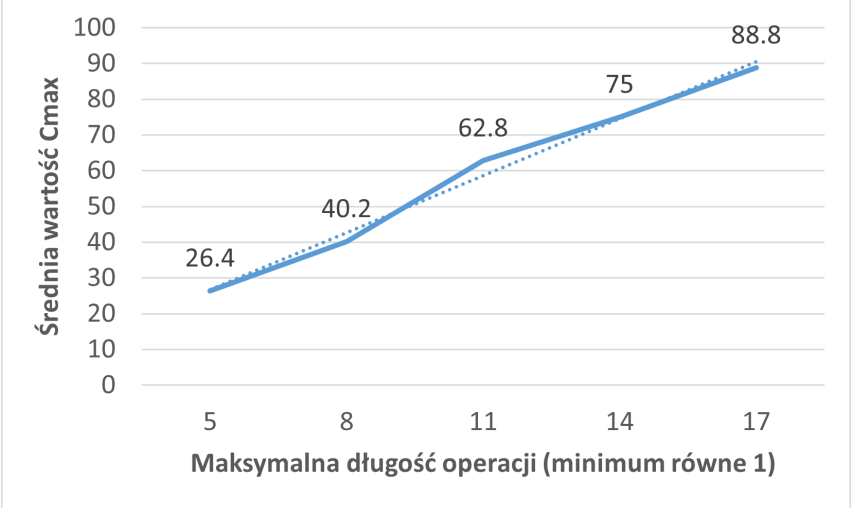
finalnie otrzymywany makespan, co ukazuje Rysunek 3.6. Występującą zależność można określić jako niemal liniową.

4. Maksymalna długość operacji

Maksymalna długość operacji (przy minimalnej ustawionej na 1) wywiera istotny wpływ na łączny czas pracy algorytmu dokładnego. Jak można zobaczyć po wynikach na Rysunku 3.7, wprowadzając dłuższe operacje do uszeregowania, które metoda podziału i ograniczeń musi połączyć z warunkiem no-wait oraz okresowymi przerwami, prawdopodobnie sprawiamy, że istnieje mniej kombinacji możliwych uszeregowań. Zgodnie z przewidywaniami, zwiększając zakres z jakich losowana jest długość poszczególnych operacji, wydłużamy finalny makespan, co prezentuje Rysunek 3.8.



Rysunek 3.7: Zależność długości operacji od czasu obliczeń. (Minimalny czas operacji równy 1).



Rysunek 3.8: Zależność długości operacji od uzyskiwanego C_{max}. (Minimalny czas operacji równy 1).

3.3 Algorytm heurystyczny

Drugim stworzonym algorytmem, rozwiązującym dany problem, jest rozwiązanie oparte o metodę tabu search, którego skuteczność również zbadano w eksperymentach obliczeniowych. W poniższym punkcie omówiona zostanie zasada działania zaimplementowanej metody tabu oraz opisane zostaną wyniki przeprowadzonych na niej testów.

3.3.1 Implementacja algorytmu Tabu Search

Działanie utworzonego algorytmu Tabu Search można opisać w następujących krokach:

1. Inicjalizacja:

Program zaczyna od zebrania od użytkownika danych wymaganych do przeprowadzenia obliczeń. Składają się na nie między innymi: długość okresów niedostępności, wartość

Tau ale także zmienne optymalizacyjne jak wielkość generowanego sąsiedztwa czy dane ile razy algorytm ma znaleźć inne rozwiązanie początkowe. Przede wszystkim jednak najważniejszym wejściem od użytkownika są konkretne zadania które mają być optymalizowane na maszynach M1 i M2. Odczytywanie tych danych odbywa się w osobnej metodzie. Na początku użytkownik podaje ilość zadań, a następnie przyjmuje po kolei ich wartości. Odczytywanie wartości polega na wypisaniu przez użytkownika dwóch wartości oddzielonych spacją. Pierwsza wartość dotyczy zadania na maszynie pierwszej a druga na maszynie drugiej. Następnie inicjowany jest algorytm.

2. Główna pętla Tabu Search

Uruchamiana jest główna pętla Tabu Search. Rozpoczyna się pomiar czasu oraz deklarowane są zmienne wymagane do poprawnego działania. Ustalana jest także wielkość listy tabu. Podczas testowania algorytmu wykazano, że średnio najlepsze wyniki otrzymuje się dla listy tabu wielkości 0.2 ilości zadań. Na początku każdego algorytmu tabu, generowane jest rozwiązanie początkowe. W tym programie rozwiązanie początkowe jest otrzymywane w sposób losowy. Za pomocą osobnej metody kolejność zadań jaka będzie rozważana w danej iteracji jest wybierana losowo a następnie przekazywana z powrotem. Rozwiązanie początkowe jest też automatycznie przypisywane jako najlepsze dotychczasowe rozwiązanie. Uruchomiona zostaje pętla odpowiedzialna za tworzenie sąsiedztwa. Sąsiedztwo jest tworzone poprzez losowe wybieranie i podmienienie miejsc w aktualnym rozwiązaniu. Jeśli dane sąsiedztwo nie zostało wpisane na listę tabu, przekazywane jest do metody addTaskToMachines która stara się opisać na maszynie zadania w wygenerowanej kolejności zgodnie z zasadami.

3. addTaskToMachines

Jest to główna metoda programu. To w tym miejscu zadania ale także okresy niedostępności są w odpowiedni sposób nakładane na maszynę by zminimalizować C_{max} jednocześnie zachowując zasady typu $NoWait$. Odbywa się to za pomocą wielu warunków if. Metoda za pomocą pętli przechodzi po kolei wszystkie przez elementy wygenerowanego sąsiedztwa i w jak najlepszy sposób stara się dopasować elementy. Metoda jest tak zaprojektowana, że przewiduje czy wymagane jest dołożenie okresu niedostępności czy nie. Jeśli wykryje, że okres bez przerwy jest dłuższy niż maksymalny czas określony przez użytkownika, wstawia przed opisywanym zadaniem okres niedostępności w jak najbardziej optymalny sposób. Bierze przy tym pod uwagę minimalizację C_{max}, warunek $NoWait$ ale też jak najlepsze wykorzystanie luk które naturalnie powstają. Po opisaniu wszystkich zadanych zadań na maszynach, metoda kończy pracę. Proces się powtarza do momentu działania głównej pętli algorytmu tabu.

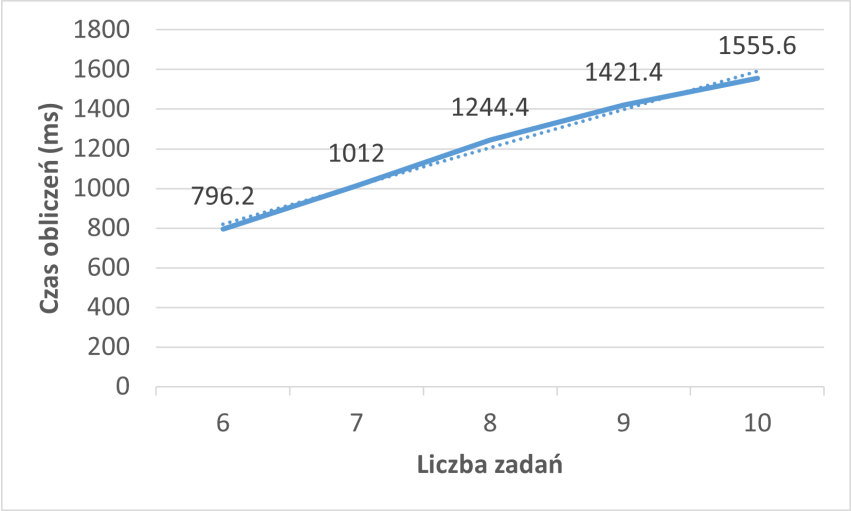
4. Koniec działania i wyświetlenie wyników

Program kończy swoje działanie w momencie skończenia wszystkich iteracji zdefiniowanych przez użytkownika zmiennych w pętlach. Zwraca na ekranie dokładne ułożenie zadań opisanych na maszynach M1 i M2, każdy blok ma podany zakres w jakim miejscu się znajduje. Zwracane jest także maksymalne C_{max} oraz czas jaki program potrzebował by uzyskać ten wynik.

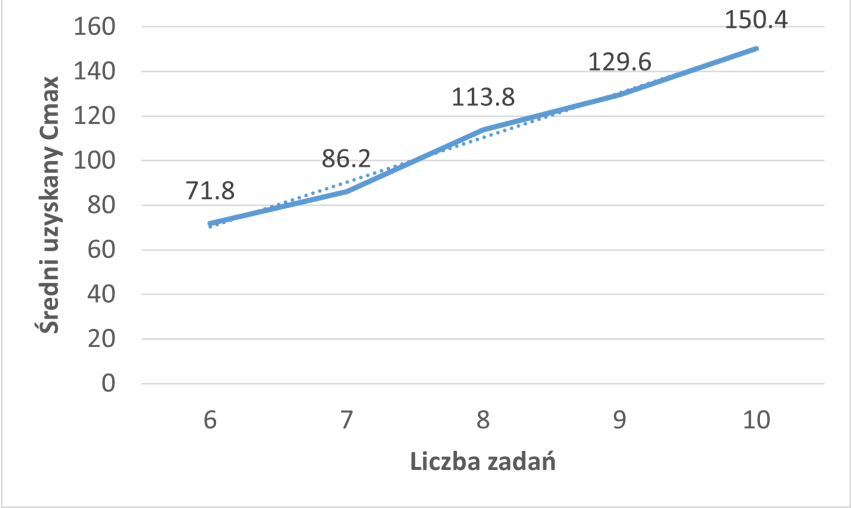
3.3.2 Eksperymenty obliczeniowe dla algorytmu heurystycznego

Poniżej znajdują się wyniki eksperymentów obliczeniowych przeprowadzonych na metodzie tabu search. Domyślne parametry ustawiono identycznie jak dla algorytmu dokładnego, ich wartości zostały zaprezentowane w Tabeli 3.1. Podobnie jak w przypadku algorytmu branch and bound, każdą próbę powtórzono 5-krotnie celem uśrednienia wyników.

1. Liczba zadań



Rysunek 3.9: Zależność liczby zadań od czasu obliczeń dla algorytmu tabu.

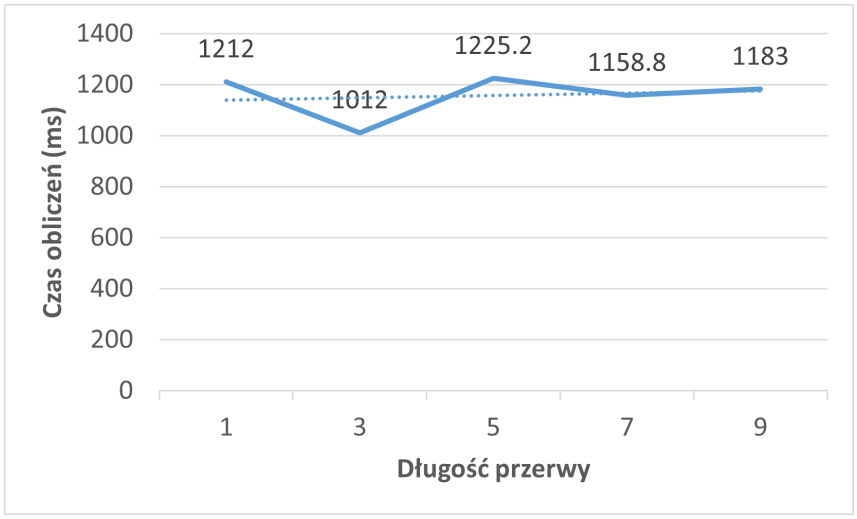


Rysunek 3.10: Zależność liczby zadań od uzyskiwanego C_{max} w algorytmie tabu.

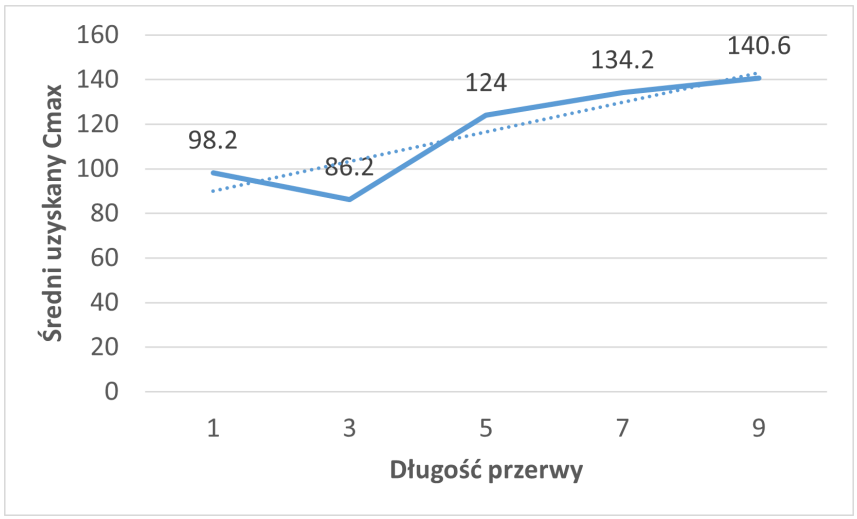
Jak widać na Rysunku 3.9 rosnąca liczba zadań powoduje proporcjonalny wzrost łącznego czasu obliczeń. Dla 10 zadań czas pracy algorytmu wynosi średni 1,5 sekundy, podczas gdy dla takiej samej ilości zadań algorytm dokładny potrzebował średnio blisko 79 sekund na znalezienie optymalnego rozwiązania. Jak pokazuje Rysunek 3.10 wzrost liczby zadań powoduje proporcjonalny wzrost uzyskiwanego C_{max}.

Należy w tym miejscu zaznaczyć, że średnia wartość C_{max} jest o 15-20% wyższa niż w przypadku algorytmu dokładnego. Oznacza to że tabu search nie znajduje rozwiązania optymalnego, natomiast rozwiązanie można uznać za całkiem zadowalające. Otrzymane uszeregowanie jest znacznie lepsze od rozwiązania początkowego lub całkowicie losowego.

2. Długość okresu niedostępności



Rysunek 3.11: Zależność długości okresu niedostępności od czasu obliczeń dla algorytmu tabu.

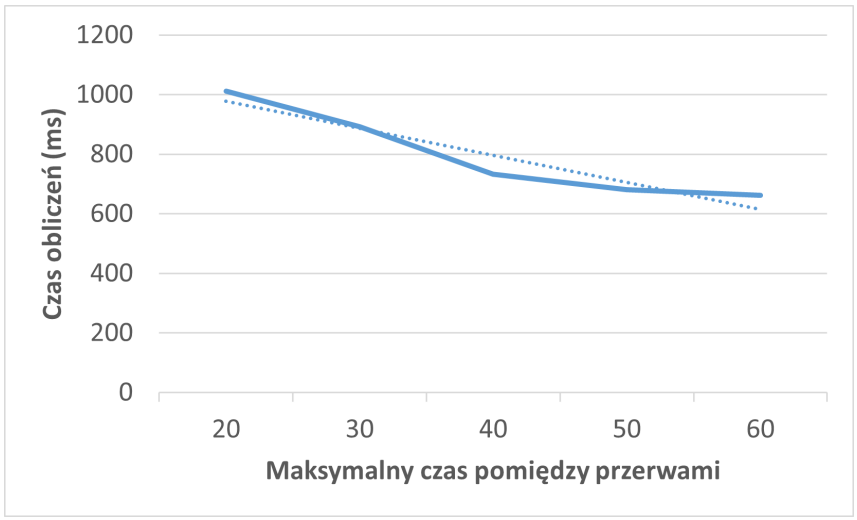


Rysunek 3.12: Zależność długości okresu niedostępności od uzyskiwanego C_{max} dla algorytmu tabu.

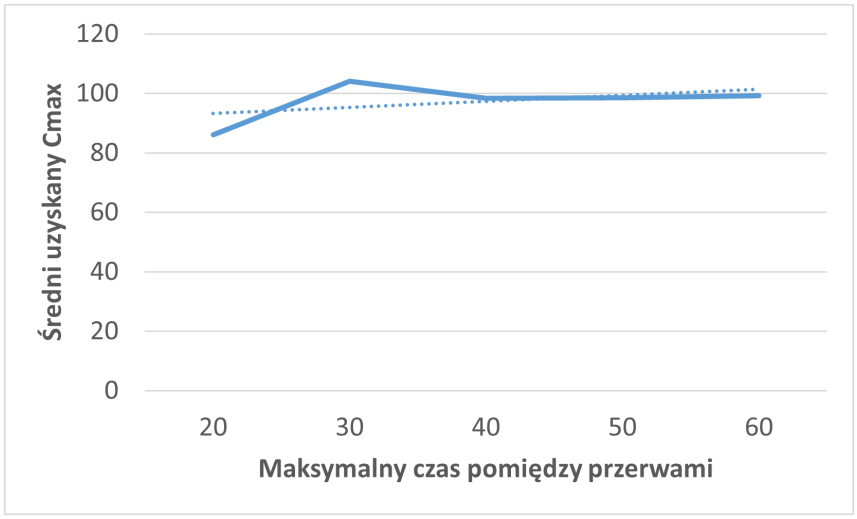
Jak widać na Rysunkach 3.11 oraz 3.12, wydłużanie liczby długości okresów niedostępności ma nieznaczne przełożenie na łączny czas pracy metaheurystyki. Zgodnie z

oczekiwaniami, podobnie jak w algorytmie dokładnym, zwiększona długość okresów niedostępności oznacza średnie powiększenie wartości osiąganego C_{max}.

3. Maksymalny czas między okresami niedostępności



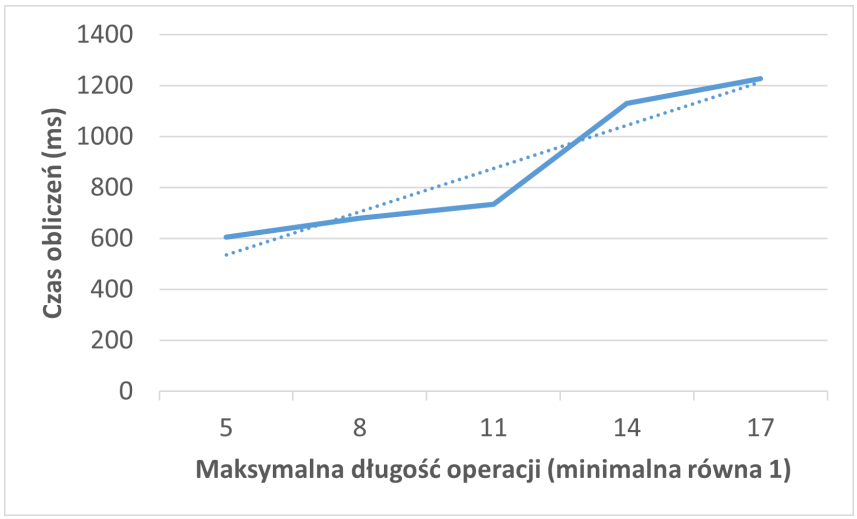
Rysunek 3.13: Zależność maksymalnego czasu pomiędzy okresami niedostępności od czasu obliczeń dla algorytmu tabu.



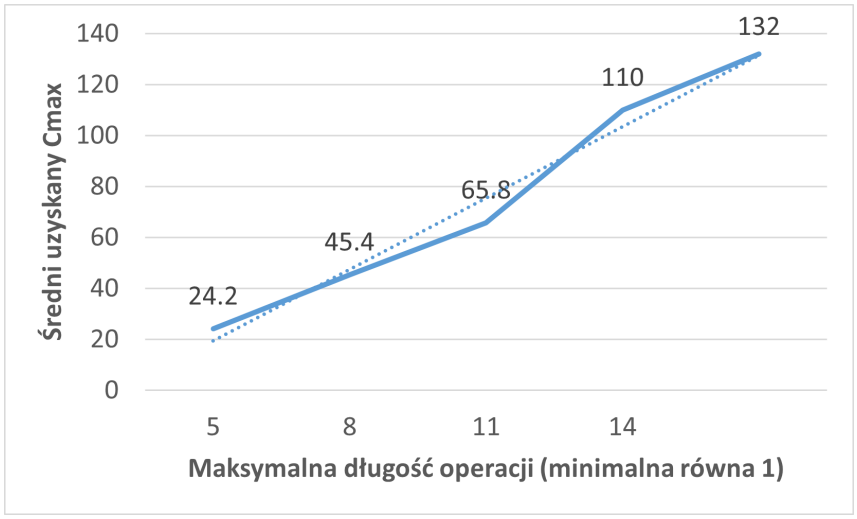
Rysunek 3.14: Zależność maksymalnego czasu pomiędzy okresami niedostępności od uzyskiwanego C_{max} dla algorytmu tabu.

Podobnie jak w algorytmie dokładnym, wydłużenie maksymalnego czasu pomiędzy kolejnymi okresami niedostępności skutkuje nieznaczną redukcją czasu obliczeń, co można zaobserwować na Rysunku 3.13. Jak widać na Rysunku 3.14 zwiększenie tego parametru ma również zauważalny wpływ na zwiększenie wartości uzyskiwanego C_{max}.

4. Maksymalna długość operacji



Rysunek 3.15: Zależność maksymalnej długości operacji od czasu obliczeń dla algorytmu tabu.



Rysunek 3.16: Zależność maksymalnej długości operacji uzyskiwanego C_{max} dla algorytmu tabu.

Zwiększanie maksymalnej długości operacji istotnie przekłada się na wydłużenie czasu obliczeń w przypadku algorytmu tabu, co uwidacznia Rysunek 3.15. Jest to zasadnicza różnica względem algorytmu dokładnego, gdzie zwiększenie długości operacji przełożyło się na skrócenie czasu obliczeń. Jak pokazuje Rysunek 3.16 wydłużenie poszczególnych zadań skutkuje powiększeniem średnich uzyskiwanych wartości C_{max}.

3.4 Porównanie obu metod i wnioski

Skuteczność obu algorytmów sprawdzono dodatkowo uruchamiając je na identycznych danych wejściowych. Przeprowadzono dwie próby w których podano 8 oraz 9 zadań różnej długości. Ponadto długość okresów niedostępności ustawiono na 3 oraz maksymalny czas pomiędzy kolejnymi okresami na 15. Tabele 3.2 oraz 3.3 przedstawiają zbiorcze wyniki dla obu prób.

Metryka	**Algorytm dokładny**	**Metaheurystyka tabu**
Czas obliczeń (ms)	4257	1254
Cmax	122	127

Tabela 3.2: Wyniki algorytmu dokładnego i tabu search dla 8 podanych zadań.

Metryka	**Algorytm dokładny**	**Metaheurystyka tabu**
Czas obliczeń (ms)	15952	1367
Cmax	133	137

Tabela 3.3: Wyniki algorytmu dokładnego i tabu search dla 9 podanych zadań.

Testy dały spodziewane rezultaty. Choć metaheurystyka radzi sobie bardzo dobrze to uzyskujemy za jej pomocą gorsze wyniki. Pod względem czasu wykonywania obliczeń widać ogromną różnicę. Jest to dobry przykład dlaczego metaheurystyki są szeroko stosowane tam gdzie nie jest wymagany dokładny wynik. Za ich pomocą możemy uzyskać zadowlające efekty w o wiele krótszym czasie.

Rozdział 4

Problem typu Open Shop

4.1 Formalny zapis problemu

Problem: O2 h_{ik} | LE | $C_{\max}$
Oznaczenia:

- n - liczba zadań,
- h_{ik} - długość k-tego okresu niedostępności na maszynie i,
- K_i - ilość okresów niedostępności na maszynie i,
- t_{li} - moment rozpoczęcia zadania na pozycji l na maszynie i
- p_{ji} - pierwotna długość zadania j na maszynie i,
- s_{ki} - moment rozpoczęcia k-tego okresu niedostępności na maszynie i,
- a - współczynnik uczenia,
- $p_{jli} = p_{ji} * l^a$ - długość wykonania zadania j na pozycji l na maszynie i
- $z_{jli} = \begin{cases} 1, & \text{jeżeli zadanie } T_{li} \text{ występuje,} \\ 0, & \text{jeżeli nie występuje.} \end{cases}$

Minimalizujemy C_{max}, przy ograniczeniach:

1. $\sum_{l=1}^{n} z_{jli} = 1 \quad : j = 1, 2, \ldots, n; \;\; i = 1, 2,$
2. $\sum_{l=1}^{n} z_{jli} = 1 \quad : l = 1, 2, \ldots, n; \;\; i = 1, 2,$
3. $z_{jli} \in \{0, 1\} \quad : j = 1, 2, \ldots, n; \;\; l = 1, 2, \ldots, n; \;\; i = 1, 2,$
4. $t_{1i} \geq 0 \quad : i = 1, 2,$
5. $a \leq 0,$

6. $K_i \geq 0 \quad : i = 1, 2,$

7. $1 \leq k \leq K_i \ \wedge k \in Z \quad : i = 1, 2,$

8. $h_{ik} > 0 \quad : i = 1, 2,$

9. $t_{li} + \sum_{j=1}^{n} p_{jl1} z_{jli} \leq s_{ki} \quad \text{lub} \quad t_{li} \geq s_{ki} + h_{ik} \quad : i = 1, 2; \ \ l = 1, 2, \ldots, n,$

10. $t_{l1} + \sum_{j=1}^{n} p_{jl1} z_{jl1} \leq t_{l2} \quad \text{lub} \quad t_{l2} + \sum_{j=1}^{n} p_{jl1} z_{jl2} \leq t_{l1} \quad : l = 1, 2, \ldots, n,$

11. $t_{l+1i} \geq t_{li} + \sum_{j=1}^{n} p_{jli} z_{jli} \quad : l = 1, 2, \ldots, n,$

12. $s_{k+1i} \geq s_{ki} + h_{ik} \quad : i = 1, 2,$

13. $C_{\max} = t_{n2} + \sum_{j=1}^{n} p_{jl2} z_{jl2}$

Rozdział 5

Podsumowanie

W niniejszej pracy omówiono szczegółowo wybrane problemy szeregowania zadań na maszynach dedykowanych typu Flow shop oraz Open shop. Dla obu problemów sporządzono ich dokładny zapis formalny, przy pomocy programowania matematycznego, nakreślając konieczne parametry i warunki, które musi spełnić rozwiązanie. Dla problemu Flow shop, z okresowymi przerwami na obu maszynach, warunkiem no-wait i miarą C_{max} zaimplementowano dwa algorytmy - dokładny i przybliżony. Następnie dla obu rozwiązań przeprowadzono szereg eksperymentów obliczeniowych i wyciągnięto wnioski na temat zastosowanych metod.

Utworzony algorytm dokładny został oparty o metodę podziału i ograniczeń (ang. branch and bound). Jak pokazują przeprowadzone testy, liczba możliwych kombinacji uszeregowania, które algorytm musi przeanalizować rośnie w sposób wykładniczy wraz ze wzrostem instancji, co sprawia, że złożoność czasowa staje się bardzo duża, nawet dla umiarkowanej ilości zadań. Odnotowane także istotne zależności pomiędzy czasem obliczeń a pozostałymi testowanymi parametrami: długością okresów niedostępności, maksymalnym czasem pomiędzy kolejnymi okresami niedostępności, a także maksymalnym czasem operacji.

Zaimplementowany algorytm metaheurystyczny, który cechuje wielomianowa złożoność obliczeniowa, osiąga w testach wyniki o 15-20% gorsze od optymalnych, generowanych przez metodę dokładną. Można to uznać za wartość zadowalającą. Należy zwrócić uwagę na fakt, że rozwiązanie oparte o metodę tabu search potrzebuje zdecydowanie mniej czasu na obliczenia, a różnica długości fazy obliczeń jest tym większa, im większa jest instancja wejściowa. Przy dziesięciu zadaniach czas obliczeń był już 50-krotnie mniejszy, gdy do rozwiązania instancji identycznej wielkości zastosowano metaheurystykę.

Podsumowując, w pracy podkreślono zalety oraz wady metody dokładnej i przybliżonej rozwiązującej dany problem szeregowania zadań typu Flow shop. Przeprowadzone testy dowodzą zasadnego stosowania metod heurystycznych, nawet dla instancji niewielkiej wielkości. Dodatkowe testy metody heurystycznej mogłyby wykazać dla jak dużych instancji zauważalnie spada jakość rozwiązania.

Literatura

Instytut Informatyki, Wydział Informatyki i Telekomunikacji
Politechnika Poznańska

Skład przy użyciu systemu LaTeX na platformie Overleaf.