WYDZIAŁ ELEKTRONIKI
POLITECHNIKA WROCŁAWSKA

PROJEKTOWANIE EFEKTYWNYCH ALGORYTMÓW

Jednoprocesorowy problem szeregowania zadań przy kryterium minimalizacji ważonej sumy opóźnień zadań

Algorytm przeszukiwania z zakazami

Autor:
Łukasz Broll
225972

Prowadzący:
dr inż. Mariusz Makuchowski

Termin zajęć:
czwartek 11^{15}-13^{00}

WROCŁAW, 2017r.

Spis treści

1. Informacje wstępne

Należy zaimplementować algorytm metaheurystyczny dla jednego z wybranych problemów oraz wykonać testy polegające na pomiarze czasu działania algorytmu w zależności od wielkości instancji oraz jakości dostarczanych rozwiązań. Należy porównać rozwiązanie dostarczone przez algorytm z najlepszymi znanymi rozwiązaniami dla przykładów testowych.

1.1 Teoretyczny opis problemu szeregowania zadań

Problem ten może być opisany następująco: Danych jest n zadań (o numerach od 1 do n), które mają być wykonane bez przerwań przez pojedynczy procesor mogący wykonywać co najwyżej jedno zadanie jednocześnie. Każde zadanie j jest dostępne do wykonania w chwili zero, do wykonania wymaga pj > 0 jednostek czasu oraz ma określoną wagę (priorytet) wj > 0 i oczekiwany termin zakończenia wykonania dj > 0. Zadanie j jest spóźnione, jeżeli zakończy się wykonywać po swoim terminie dj , a miarą tego opóźnienia jest wielkość Tj = max(0, Cj , −dj), gdzie Cj jest terminem zakończenia wykonywania zadania j. Zadanie polega na znalezieniu takiej kolejności 1 wykonywania zadań (permutacji), aby zminimalizować kryterium TWT.

Źródło: http://www.zio.iiar.pwr.wroc.pl/pea/pea_projekt_1718.pdf

1.2 Teoretyczny opis algorytmu przeszukiwania z zakazami

Metaheurystyka Tabu Search może być określona jako metoda dynamicznej zmiany sąsiedztwa danego rozwiązania. Oznacza to, ze zbiór rozwiązań nie jest z góry ustalony dla każdego, lecz może zmieniać się w zależności od informacji zebranych w poprzednich etapach przeszukiwania. Za twórcę uznawany jest Fred W. Glover, który w latach 70-tych opublikował kilka prac związanych z tym sposobem poszukiwania rozwiązania.

Metoda Tabu Search posiada następujące charakterystyczne elementy:

- Funkcje oceny wartości ruchu,
- Listę ruchów zakazanych (Tabu List),
- Kryterium aspiracji,
- Strategie wyboru rozwiązań,
- Strategię dywersyfikacji,
- Warunek zakończenia.

Tabu Search jest metodą bardzo ogólną, a szczegóły jej implementacji różnią się znacznie w zależności od problemu, do którego została zastosowana.

Źródło: http://cs.pwr.edu.pl/zielinski/lectures/om/localsearch.pdf

2. Implementacja

Aplikacja konsolowa zaprojektowana do rozwiązania problemu została wykonana w języku C++ za pomocą Microsoft Visual Studio 2015. Projekt oparty jest o 3 klasy oraz bazową metodę main():

2.1 Klasa **tasks**

Podstawowa struktura do zarządzania pojedynczym zadaniem. Przechowuje zmienne prywatne `_id`, `_time`, `_weight` oraz `_date` każdego zadania oraz ich gettery. Umożliwia dodawanie kolejnych zadań oraz ich drukowanie. Zadania przechowywane są w tablicy `tasksT.`

2.2 Klasa **tabu**

Klasa dziedzicząca po klasie `tasks` i `stack`, odpowiedzialna za główne funkcjonalności aplikacji. Zawiera m.in. metodę generującą listę sąsiedztwa, sprawdzającą kryterium aspiracji czy liczenie całkowitego opóźnienia. Dodatkowo system pomiaru czasu został zaimplementowany wewnątrz klasy. Znajdują się w niej wszystkie niezbędne tablice, metody oraz zmienne, które szczegółowo zostały opisane w nagłówku klasy (`tabu.h`).

2.3 Klasa **stack**

Klasa również dziedziczy po klasie `tasks.` Jest odpowiedzialna za przechowywanie listy ruchów zakazanych. Wewnątrz klasy znajduj się struktura `simpleTurn,` w której przechowywane jest pojedyncze przejście między zadaniami. Oprócz podstawowych metod związanych z dodawaniem i zdejmowaniem ruchów ze stosu, zawiera metodę sprawdzającą czy dany ruch nie znajduje się już na liście tabu. Kadencja pojedynczego ruchu zakazanego jest równa rozmiarowi listy tabu.

2.4 Metoda **main()**

Zadaniem metody jest jedynie komunikacja z użytkownikiem. Znajduję się w niej nieskończona pętla menu funkcji dostępnych dla użytkownika oraz krótka informacja o programie.

2.5 Algorytm przeszukiwania z zakazami

Działanie algorytmu oparte jest i kilka metod podrzędnych, w których sprawdzane są kolejne parametry całego algorytmu oraz o nadrzędną metodę `tabuSolve()`, która łączy i sprawdza wszystkie warunki oraz parametry, a na ich podstawie zwraca optymalne rozwiązanie.

Pierwszym etapem działania algorytmu jest inicjacja parametrów statycznych, tj. rozmiaru listy tabu, listy sąsiedztwa oraz limitu iteracji wykonywania algorytmu. Rozwiązanie, od którego algorytm rozpoczyna działanie jest ułożone rosnąco, zgodnie z kolejnością wczytania przez aplikację.

Następnie wykonuje się wewnętrzna pętla, w której sprawdzana jest lista sąsiedztwa danego rozwiązania. Wewnątrz pętli sprawdzana jest jakość przejścia – aby uniknąć wpisania na listę ruchów zakazanych, przejście musi być chociaż 20% lepsze od dotychczasowego lub spełniać kryterium aspiracji (ważona suma opóźnień zadań musi być mniejsza od dotychczasowej). Po wykonaniu pętli, sprawdzana jest jakość ostatniego (najlepszego), rozwiązania z pętli, ze znanym ogólnie najlepszym rozwiązaniem. Jeśli tyle samo razy, co rozmiar listy sąsiedztwa, algorytm nie uzyska lepszego rozwiązania, następuje dywersyfikacja – losowane jest nowe rozwiązanie, na podstawie którego algorytm będzie szukał lepszych rozwiązań. Czas działania głównej pętli algorytmu uzależniony jest od warunku zakończenia, którym jest ilość dywersyfikacji.

2.6 Algorytm sprawdzania listy sąsiedztwa

Działanie algorytmu przeszukiwania z zakazami opiera się na liście sąsiedztwa. Na jej podstawie generowana jest lista ruchów zakazanych. W połączeniu z dywersyfikacją oraz kryterium aspiracji, algorytm pozwala skutecznie rozwiązywać problemy dużych instancji w sensownym czasie. Warunkiem szybkiego i poprawnego działania algorytmu sprawdzania listy sąsiedztwa jest nałożenie ograniczenia przy szukaniu najlepszych rozwiązań oraz rozmiaru samej listy. Jeśli algorytm określoną ilość razy (np. dla 100. instancji 8. razy) nie znajdzie lepszego rozwiązania, akceptuje te które otrzymał. Natomiast ustalenie zbyt dużego lub zbyt małego rozmiaru listy, zwiększy ryzyko pominięcia dobrych rozwiązań.

Proces wyznaczania najlepszego sąsiada zaczyna się od pobrania listy zadań, które są aktualnie rozpatrywane. Permutacja listy sąsiedztwa polega na wylosowaniu dwóch zadań i zamienieniu ich miejscami. Zanim to nastąpi, algorytm sprawdza czy ruch nie znajduje się przypadkiem na liście ruchów zakazanych, czy kryterium aspiracji pozwala złamać listę tabu lub czy nie został przekroczony limit szukania nowych rozwiązań. Jeśli nowy układ zadań okazuje się lepszy, od dotychczasowego najlepszego, to jest zapamiętywany, a licznik gorszych rozwiązań jest resetowany. Limitem losowania nowych kombinacji listy sąsiedztwa jest jej rozmiar.

3. Testy wydajności

Przeprowadzone zostały testy wydajności aplikacji, w zależności od rozmiaru instancji (ilości zadań) oraz parametrów algorytmu. Przetestowane zostały przykłady z plików *wt40.txt* oraz *wt100.txt*, ze źródeł z instrukcji[3]. W obu plikach znajduje się po 125 przypadków szeregowania zadań, zawierających odpowiednio po 40 i 100 zadań. Natomiast w plikach *wtopt40.txt* oraz *wtopt100.txt*, ich najlepsze rozwiązania, do których zostały przyrównane wyniki działania testów.

W związku z tym, że jakość algorytmu tabu search, zależy głównie od zaimplementowanych właściwości, przetestowane zostały następujące parametry:

- Liczba dywersyfikacji (warunek zakończenia)
- Rozmiar listy sąsiedztwa
- Rozmiar listy tabu

Dla każdego zestawu zadań zmierzony został czas działania algorytmu i jego wynik. Następnie obliczony został błąd względny wyniku w stosunku do najlepszych wyników z plików *wtopt.* Dla każdego zestawu zadań czasy oraz błędy względne uśredniono.

Przed przystąpieniem do pomiarów w aplikacji wyłączone zostały wszystkie komendy drukujące napisy na ekran. Program został uruchomiony w wersji Release x64 przy minimalnych obciążeniu systemu. Specyfikacja sprzętu, na którym przeprowadzone testy jest następująca:

- Procesor: Intel® Core™ i5-2410M CPU @ 2.30Ghz
- RAM: 8.00 GB
- System: Windows 10 x64
- Dysk: SSD GOODRAM IRIDIUM PRO 240GB 560MB/S SATA III

3.1 Testy na pliku wt40.txt

LP	Liczba dywersyfikacji	Rozmiar listy sąsiedztwa	Rozmiar listy tabu	Średni czas[ms]	Średni błąd względny
1	10	10	20	582,31	2,000
2	20	12	6	1084,71	0,697
3	**20**	**15**	**7**	**1606,27**	**0,024**
4	10	20	10	1609,70	1,937
5	10	15	8	835,75	4,206
6	10	30	10	5579,13	0,435
7	20	30	8	12349,45	0,668
8	20	30	12	11606,65	1,145

Tabela 1. Tabela rozwiązań dla pliku wt40.txt

Na podstawie uzyskanych wyników, można wywnioskować, że najlepszym zestawem parametrów, do uzyskania najmniejszych błędów w sensownym czasie, jest zestaw 3., którego średni czas wykonania wynosi 1,6 sekundy. Zwiększenie liczby dywersyfikacji oraz rozmiaru listy sąsiedztwa ma wpływ na poprawę jakości rozwiązań, natomiast rozmiar listy tabu nie może być, ani za dużo, ani za mały, żeby uzyskać lepsze rozwiązanie.

3.2 Testy na pliku wt100.txt

LP	Liczba dywersyfikacji	Rozmiar listy sąsiedztwa	Rozmiar listy tabu	Średni Czas [ms]	Średni błąd względny
1	40	15	10	1412,96	119,92
2	100	10	6	3038,22	309,19
3	10	20	10	5275,72	3,52
4	20	30	6	24066,47	2,64
5	25	30	7	31535,96	2,25
6	30	30	12	25646,25	2,41
7	**20**	**25**	**8**	**13814,66**	**0,14**
8	10	30	15	26324,84	2,53
9	4	40	8	24469,64	0,80
10	20	30	12	56435,71	0,70
11	50	50	8	364037,00	0,11

Tabela 2. Tabela rozwiązań dla pliku wt100.txt

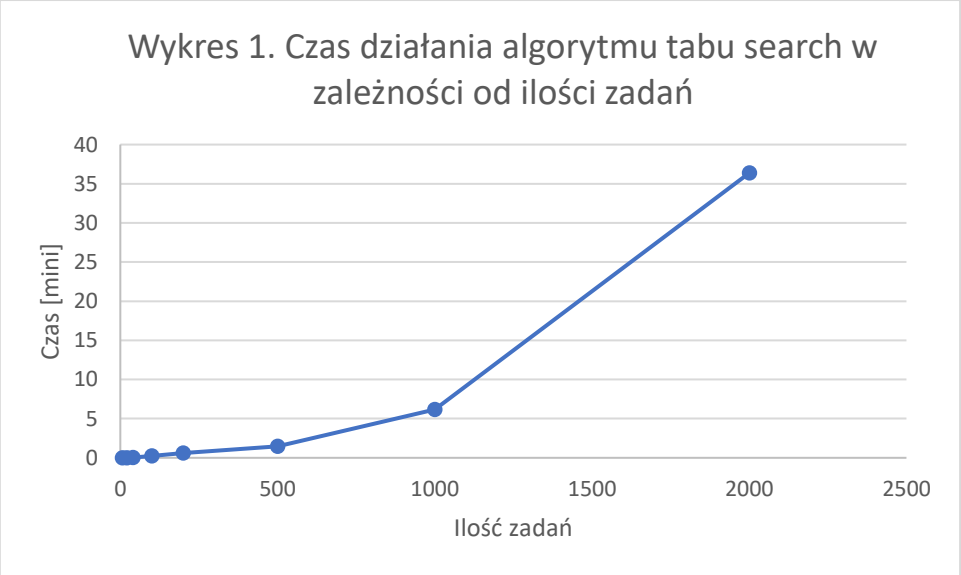
Na podstawie uzyskanych wyników, można wywnioskować, że najlepszym zestawem parametrów, do uzyskania najmniejszych błędów w sensownym czasie, jest zestaw 7., którego średni czas wykonania wynosi ok. 14 sekund. W tym przypadku duży wpływ na pogorszenie wyników ma zbyt duży rozmiar liczby dywersyfikacji, ale za to algorytm wykonuje się bardzo szybko. Zwiększenie rozmiaru listy sąsiedztwa znacząco zwiększa czas wykonania algorytmu, ale w tym przypadku wpływa na poprawę wyników, natomiast rozmiar listy tabu nie może być ani za dużo, ani za mały, tak samo jak w poprzednim przypadku.

3.3 Test wydajności czasowej od ilości zadań

W związku z brakiem plików testowych dla instancji większych niż 100, test wydajności czasowej względem ilości zadań został wykonany na losowo wygenerowanych instancjach zadań. Na podstawie testów pliku wt40.txt oraz wt100.txt, proporcjonalnie zwiększano parametry algorytmu, żeby zasymulować mniejszy błąd wyniku.

Ilość zadań	Liczba dywersyfikacji	Rozmiar listy sąsiedztwa	Rozmiar listy tabu	Średni Czas[ms]	Średni błąd względny
6	15	10	7	0.07116	-
20	15	12	7	0.17404	-
40	20	15	7	1606,27	0,024
100	20	25	8	13814,66	0,14
200	20	28	10	36614,1	-
500	20	30	12	88605,8 (ok. 1,5 minuty)	-
1000	25	32	14	369434 (ok. 6 minut)	-
2000	30	35	16	2182145 (ok. 35 min)	-

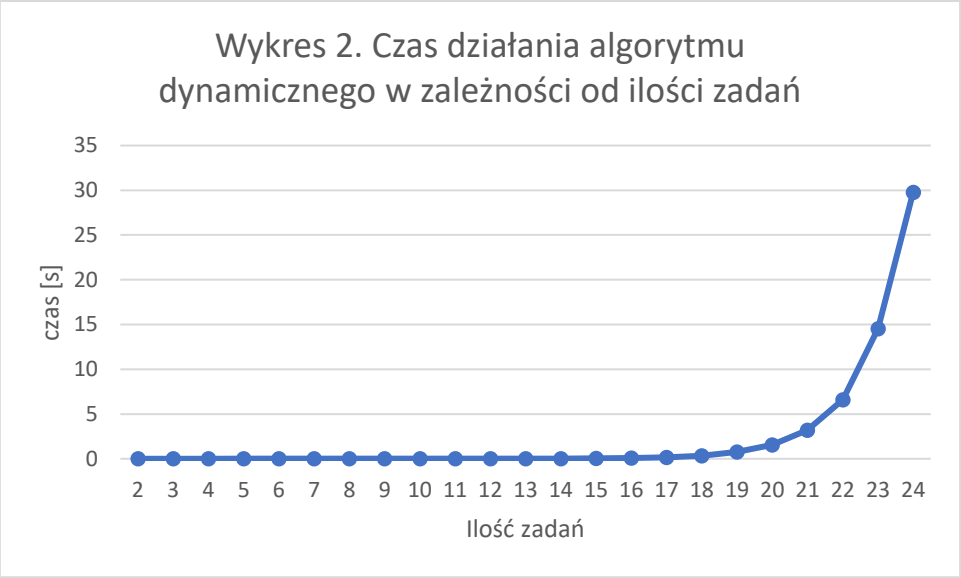
Tabela 3. Tabela rozwiązań dla losowo wygenerowanych instancji bez porównania do optymalnych wyników.



3.4 Porównanie do algorytmu dynamicznego

Cechy algorytmu przeszukiwania z zakazami względem algorytmu dynamicznego są trudne do jednoznacznego porównania, ponieważ występuje między nimi wiele różnic. Algorytm dynamiczny, sprawdzając wszystkie możliwe rozwiązania jest bardzo dokładny, kosztem nieporównywalnie większego czasu i zasobów pamięci, potrzebnych do jego działania. Zastosowanie algorytmu tabu search, dzięki wykorzystaniu wielu parametrów, daje możliwość rozsądnego wyboru miedzy czasem działania algorytmu, a jakością, oraz nie obciąża tak bardzo zasobów sprzętu. Bardzo ważny jest również fakt, że algorytm tabu search jest w stanie opracować rozwiązanie dla problemu instancji rzędu setek i tysięcy. Oczywiście jakość rozwiązania, będzie zależała od odpowiedniej modyfikacji parametrów oraz czasu jaki użytkownik jest w stanie poświęcić na otrzymanie wyniku.

Porównując Wykres 1 i Wykres 2 od razu widać nieporównywalny większy czas na wykonanie algorytmu dynamicznego dla 24 zadań (ok. 30s), gdzie algorytm tabu search dla 100 zadań potrzebuje zaledwie 14. sekund, przy bardzo niskim ryzku popełnienia błędu.



4. Wnioski

Podsumowując, przygotowanie projektu pozwoliło na zapoznanie się z tematyką algorytmiki metaheurystycznej. Po przeprowadzeniu analizy wyników testów, jednoznacznie widać, że ilość zadań ma wpływ na czas działania programu. Częste sprawdzanie kryterium aspiracji tj, przeliczanie ważonej sumy opóźnień zadań, dodatkowo wydłuża czas działania programu. Algorytm pozwala rozwiązywać znacznie większe instancje, niż programowanie dynamiczne, jednak odpowiednie dobranie parametrów metodą eksperymentalną wymaga czasu oraz dobrej znajomości problemu. Niewątpliwą zaletą algorytmu jest możliwość kompromisu właśnie między czasem działania a jakością rozwiązania.

5. Bibliografia

1. Metody przeszukiwania lokalnego, http://cs.pwr.edu.pl/zielinski/lectures/om/localsearch.pdf
2. Instrukcja projektu, http://www.zio.iiar.pwr.wroc.pl/pea/pea_projekt_1718.pdf
3. Przykładowe dane testowe http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html
4. Wykład z kursu http://www.zio.iiar.pwr.wroc.pl/pea/w5_ts.pdf