WYDZIAŁ ELEKTRONIKI
POLITECHNIKA WROCŁAWSKA

PROJEKTOWANIE EFEKTYWNYCH ALGORYTMÓW

Jednoprocesorowy problem szeregowania zadań przy kryterium minimalizacji ważonej sumy opóźnień zadań

Algorytm genetyczny

Autor:
Łukasz Broll
225972

Prowadzący:
dr inż. Mariusz Makuchowski

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

WROCŁAW, 2018r.

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 genetycznego

Algorytm genetyczny – rodzaj heurystyki przeszukującej przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.

Sposób działania algorytmów genetycznych nieprzypadkowo przypomina zjawisko ewolucji biologicznej, ponieważ ich twórca John Henry Holland właśnie z biologii czerpał inspiracje do swoich prac. Obecnie zalicza się go do grupy algorytmów ewolucyjnych.

Problem definiuje środowisko, w którym istnieje pewna populacja osobników. Każdy z osobników ma przypisany pewien zbiór informacji stanowiących jego genotyp, a będących podstawą do utworzenia fenotypu. Fenotyp to zbiór cech podlegających ocenie funkcji przystosowania modelującej środowisko. Innymi słowy - genotyp opisuje proponowane rozwiązanie problemu, a funkcja przystosowania ocenia, jak dobre jest to rozwiązanie.

Źródło: https://pl.wikipedia.org/wiki/Algorytm_genetyczny

2. Implementacja

Aplikacja konsolowa zaprojektowana do rozwiązania problemu została wykonana w języku C++ za pomocą Microsoft Visual Studio 2015. Projekt oparty jest o 2 klasy oraz bazową metodę main():

2.1 Klasa **`tasks`**

Podstawowa struktura do zarządzania pojedynczym zadaniem. Przechowuje zmienne prywatne `_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 **`genetic`**

Klasa dziedzicząca po klasie `tasks`, odpowiedzialna za główne funkcjonalności aplikacji. Zawiera m.in. główną metodę algorytmu, metody wczytujące dane, krzyżowanie, mutacje, reprodukcje 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 (`genetic.h`).

2.3 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.

2.4 Algorytm genetyczny – implementacja

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

Pierwszym etapem działania algorytmu jest losowe wygenerowanie populacji początkowej. Selekcja populacji opiera się o uproszczoną selekcję rangową. Populacja sortowana jest malejąco, według całkowitej sumy opóźnień zadań, a najgorsze genotypy są odrzucane. Mimo zaimplementowania selekcji turniejowej, nie została ona uwzględniona w testach, ponieważ bardzo duży błąd wyniku sugeruje jej nieprawidłowe działanie. Krzyżowanie osobników zostało zrealizowane metodą OX oraz OrderOne (krzyżowanie jednopunktowe). Od współczynnika reprodukcji zależy ile genotypów weźmie udział w krzyżowaniu. Mutacja została zrealizowana metodą transposition – zamienia dwa losowo wybrane zadania. Może zajść losowo dla każdego z osobników, zgodnie ze prawdopodobieństwem mutacji. Warunkiem zakończenia działania algorytmu jest ilość generacji, czyli rozmiar pętli.

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 genetyczny, zależy głównie od zaimplementowanych właściwości, przetestowane zostały następujące parametry:

- Rozmiar populacji
- Liczba generacji

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:

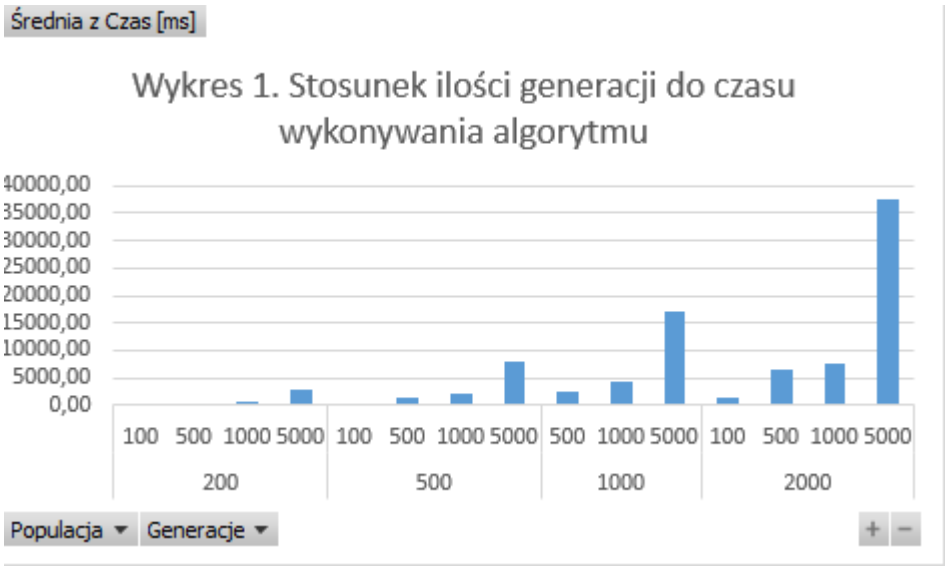
- Procesor: Intel® Core™ i5-2410M CPU @ 2.30Ghz
- RAM: 8.00 GB
- System: Windows 10 x64
- Dysk: SSD GOODRAM IRIDIUM PRO 240GB 560MB/S SATA III

3.1 Testy na pliku wt100.txt

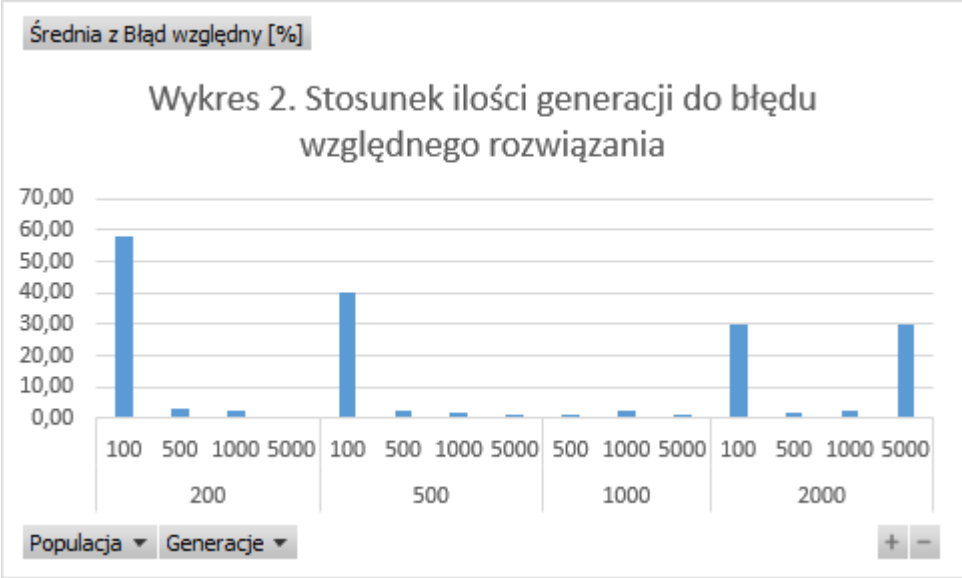
Populacja/Generacje	Średnia z Czas [ms]	Średnia z Błąd względny [%]
200	**800,38**	**18,33**
100	101,15	57,95
500	453,71	3,17
1000	751,74	2,72
5000	2989,47	0,65
500	**2256,24**	**12,65**
100	347,51	39,82
500	1288,98	2,26
1000	2213,97	1,69
5000	8092,75	1,02
1000	**6247,36**	**1,69**
500	2631,74	1,47
1000	4445,60	2,24
5000	17082,14	1,02
2000	**10108,61**	**15,80**
100	1513,99	29,65
500	6351,04	1,77
1000	7603,44	2,26
5000	37318,17	29,70

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 200/5000., którego średni czas wykonania wynosi ok. 3 sekund. W tym przypadku duży wpływ na pogorszenie wyników ma zbyt mały rozmiar generacji w stosunku do populacji.



Na wykresie 1. można zauważyć, że proporcjonalnie do ilości generacji i rozmiaru populacji, wydłuża się czas działania algorytmu.



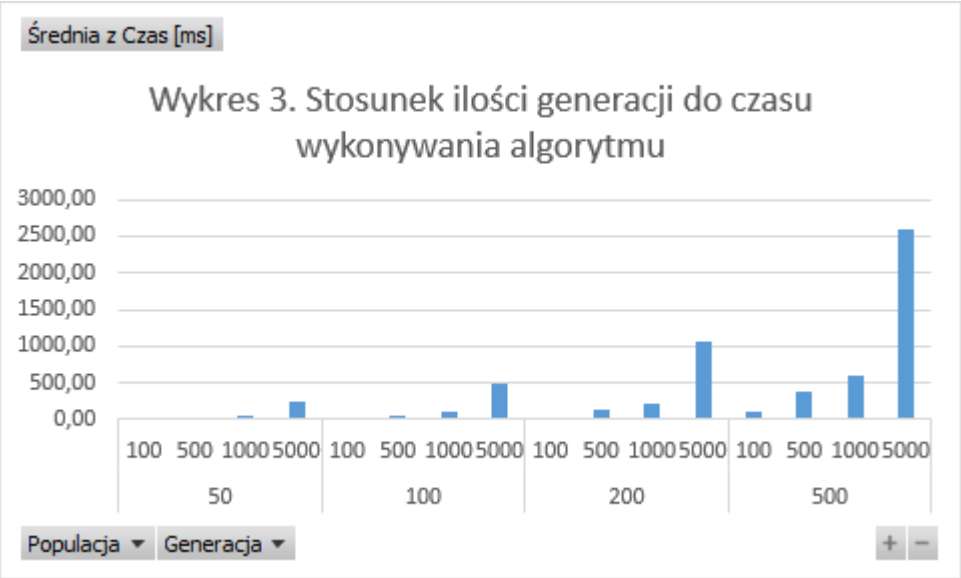
Natomiast na wykresie 2. widać, że zbyt mały rozmiar populacji oraz za mała liczba generacji, ma negatywny wpływ na jakość rozwiązania.

3.2 Testy na pliku wt40.txt

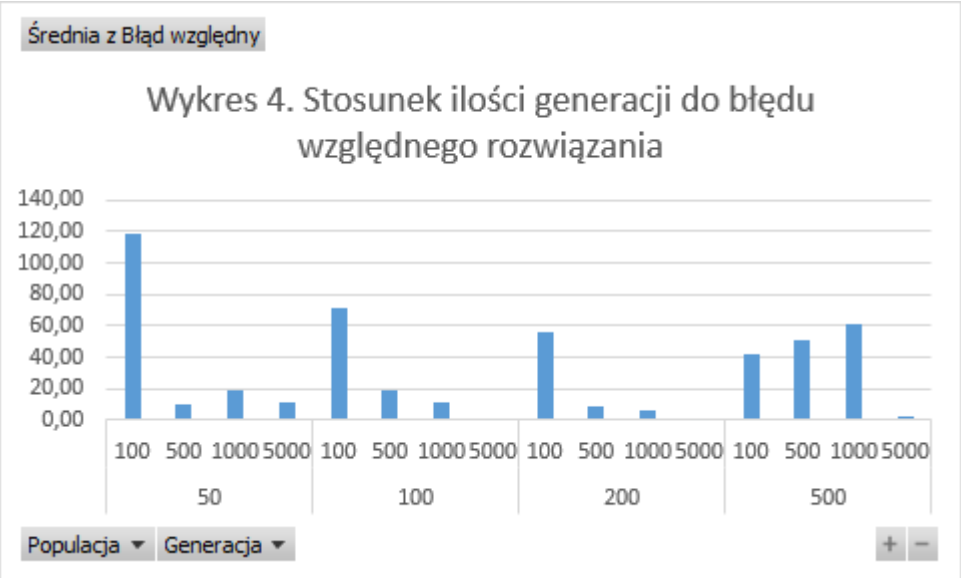
Populacja/Generacje	Średnia z Czas [ms]	Średnia z Błąd względny [%]
50	**80,83**	**40,02**
100	6,49	119,07
500	25,64	9,74
1000	51,77	19,48
5000	238,96	11,56
100	**118,05**	**29,35**
100	15,11	71,27
500	58,13	18,53
1000	104,68	11,08
5000	497,83	1,73
200	**239,66**	**20,98**
100	34,88	56,05
500	125,56	8,50
1000	224,18	6,41
5000	1072,62	0,94
500	**626,88**	**45,47**
100	108,38	41,79
500	374,41	51,10
1000	602,20	60,83
5000	2605,62	2,17

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 200/5000., którego średni czas wykonania wynosi mniej niż sekundę. Zwiększenie rozmiaru populacji oraz ilości generacji ma wpływ na poprawę jakości rozwiązań. Nieodpowiednio dobrane proporcje obu parametrów, jednak pogarszają rozwiązanie.



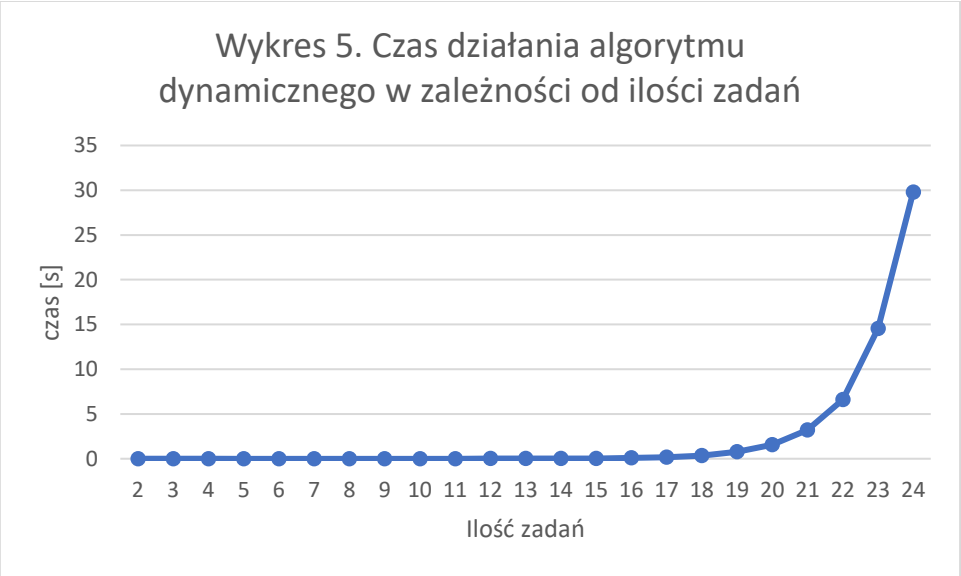
Na wykresie 3. można zauważyć, że proporcjonalnie do ilości generacji i rozmiaru populacji, wydłuża się czas działania algorytmu.



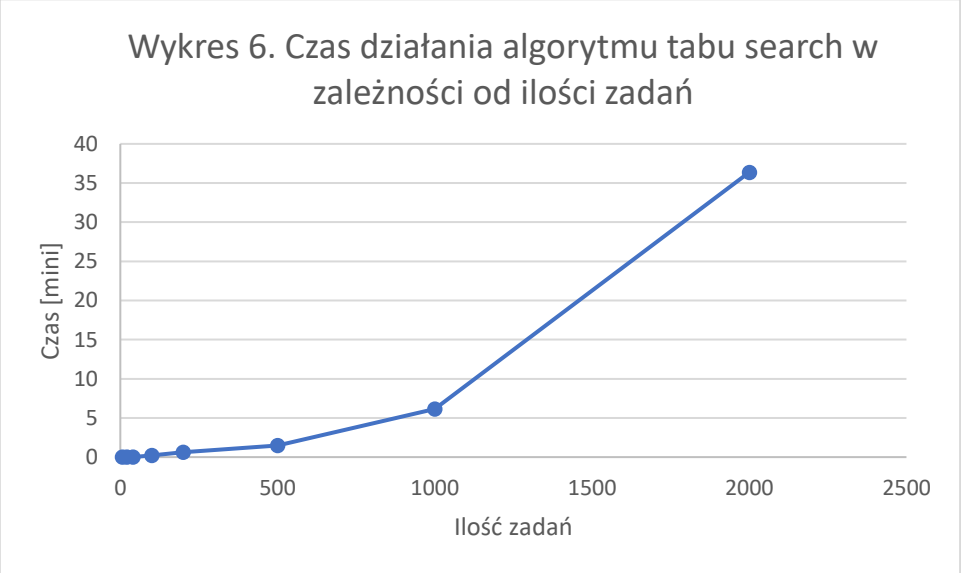
Natomiast na wykresie 4. widać, że zbyt mały rozmiar populacji oraz za mała liczba generacji, ma negatywny wpływ na jakość rozwiązania, jednak przy większym rozmiarze populacji, jest ryzyko, że mimo wszystko rozwiązania mogą nie być lepsze.

3.3 Porównanie do algorytmu dynamicznego

Cechy algorytmu genetycznego 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 genetycznego, dzięki wykorzystaniu wielu parametrów, daje możliwość rozsądnego wyboru miedzy czasem działania algorytmu, a jakością, oraz nie obciąża tak bardzo zasobów sprzętu. Porównując Wykres 5 do Wykresu 1. i 3., od razu widać nieporównywalny większy czas na wykonanie algorytmu dynamicznego dla 24 zadań (ok. 30s), gdzie algorytm genetyczny dla 40 zadań potrzebuje mniej niż sekundę, przy bardzo niskim błędzie wyniku.



Niestety, w związku z nieodpowiednią implementacją metody wyboru zadań w algorytmie tabu search, niemożliwe jest bezpośrednie porównanie obu algorytmów. Porównując ogólnie czasy działania i błędy względne rozwiązań algorytmów, te uzyskane za pomocą algorytmu tabu search są wyraźnie szybsze i lepsze, nawet dla dużo większego rozmiaru instancji.



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. Rozmiar populacji oraz ilość generacji tj, również 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. Algorytm genetyczny, wykład z kursu, http://www.zio.iiar.pwr.wroc.pl/pea/w9_ga_tsp.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
5. Modyfikacje i ulepszenia standardowego algorytmu genetycznego
   http://aragorn.pb.bialystok.pl/~wkwedlo/EA3.pdf