Projektowanie efektywnych algorytmów - Projekt 3

Problem komiwojażera - Algorytm populacyjny.

.

Termin zajęć: Czwartek 09:15-11:00
Prowadzący: dr inż. Dariusz Banasiak
Adam Filipowicz 221713

1 Wstęp

W projecie zaimplemenotwany został algorytm populacyjny dla problemu komiwojażera. Stworzony algorytm uwzględnia:

- tworzenie losowych populacji początkowych i ich ocenę przystosowania,
- selekcję turniejową jako wybór populacji macierzystej,
- krzyżowanie chromosomów operatorem OX (ang. *order crossover*),
- mutację transpozycyjną,
- uaktualnienie ocen przystosowania organizmów,
- warunek zakończenia algorytmu.

W projekcie przedstawiony jest wstęp teoretyczny dotyczący problemu komiwojażera oraz algorytmów populacyjnych. Następnie przedstawiony jest sposób implementacji oraz przykładowy sposób wykonania algorytmu. Następnie podane zostały założenia projektowe. W końcu następuje podanie wyników testów oraz ich analiza w porównaniu do algorytmów wykorzystywanych w pozostałych projektach. Pod koniec pracy podane są wnioski oraz bibliografia.

Testy oraz porównanie następuje dla realnych problemów pobranych ze strony 'comopt.ifi.uni-heidelberg.de/software/TSPLIB95/' – na wartościach i czasach wykonywania programu z projektów poprzednich (programowania dynamicznego oraz Tabu Search).

Graf używany był do reprezentacji odległości między miastami. Do reprezentacji grafu użyta została macierzy, w której przechowywane są wartości poszczególnych krawędzi, np. wartość M[i][j] to wartość krawędzi między wierzchołkami (i j). Zużycie pamięci dla grafu to O(V^2), gdzie V to ilość wierzchołków. Dodanie, sprawdzenie istnienia i usunięcie krawędzi mają złożoność O(1). Sprawdzenie stopnia wierzchołka ma złożoność O(V).

Wynik działania algorytmu przechowywany jest w tablicy o długości V w postaci znalezionej permutacji oraz w zmiennej przechowującej długość całkowitą znalezionej ścieżki.

1.1 Problem komiwojażera

Problem komiwojażera polega na znalezieniu najkrótszego cyklu w grafie pełnym, gdzie każdy wierzchołek odwiedzamy dokładnie raz.

Algorytmy populacyjne

Zaimplementowany algorytm to algorytm populacyjny. Główne człony algorytmu to selekcja, krzyżowanie oraz mutacja. Na początku algorytmu generowana jest losowa populacja początkowa, a następnie dla każdego organizmu oceniane jest jego przystosowanie. Im przystosowanie organizmu jest większe, tym większe jest prawdopodobieństwo, że zostanie on wybrany w etapie selekcji.

Następnie następuje pętla która zawiera selekcję, krzyżowanie, mutację oraz ponowną ocenę przystosowania nowych organizmów z zadanym warunkiem zakończenia algorytmu. Selekcja polega na wybraniu populacji macierzystej z populacji wszystkich początkowych organizmów o najlepszej ocenie przystosowania. Nie zawsze jednak wybranie wszystkich najlepszych organizmów jest korzystne, dlatego powstało wiele różnych metod selekcji (np. turniejowa lub ruletka).

Krzyżowanie polega na stworzeniu nowych organizmów (potomstwa) z populacji macierzystej przy zachowaniu większości członów rodziców (zakładamy, że w etapie selekcji wybrano organizmy o wysokiej ocenie przystosowania). W przypadku problemu komiwojażera krzyżowanie nie jest tak oczywiste jak w przypadku problemów binarnych, dlatego powstało wiele metod i algorytmów do tego problemu (np. OX, PMX czy EX).

Mutacja polega na nieznacznej zmianie rozwiązania (genotypu) potomstwa przed zdefiniowanie prawdopodobieństwa mutacji. W dużej mierze poprawia to rozwiązania, przez stworzenie unikalnych rozwiązań które raczej nie powstałyby jedynie pod wpływem krzyżowania. Istnieje wiele metod mutacji, np. transpozycja, mutacja przez inwersję czy Scramble Mutation.

Implementacja

Populacja początkowa utworzona jest przez stworzenie losowych permutacji o zadanej liczności. Ocena przystosowania dla każdego 'organizmu' to długość ścieżki przejścia po wszystkich miastach.

Sposób selekcji został wybrany jako turniejowa. Jej działanie przedstawia następujący algorytm:

1. `wybierz losowo k (k < L, gdzie L to liczność populacji podstawowej P) osobników (bez zwracania) z populacji P,`

2. `zachowaj najlepszego i umieść go w populacji macierzystej S,`

3. `zwróć k osbników do populacji P,`

4. `wykonaj L razy, aby S = L`.

Sposób krzyżowania został wybrany jako OX. Sposób jego działania jest następujacy:

- losujemy indeksy k_1 oraz k_2, przy czym $k_1 < k_2$ oraz $k_1, k_2 \in [1, ..., w]$, gdzie w to liczba miast w problemie,
- do pierwszego potomka kopiujemy elementy z pierwszego rodzica o indeksach $[k_1, ..., k_2]$,
- dla drugiego rodzica przestawiamy wszystkie miasta tak, aby na pierwszym miejscu znalazło się miasto o indeksie $(k_2 + 1) \mathrm{mod} w$, na drugim miasto o indeksie $(k_2 + 2) \mathrm{mod} w$ itd., a następnie usuwamy z niego numery miast, które już występują w pierwszym potomku,
- zaczynając od indeksu $(k_2 + 1) \mathrm{mod} w$ przypisujemy wszystkie miasta pozostałe w drugim rodzicu od pierwszego indeksu,
- drugi potomek powstaje analogicznie przy zamianie rodziców.

Przykład pokazujący działanie sposobu krzyżowania OX według kroków przedstawionego powyżej algorytmu:

rodzic 1: $r_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$
rodzic 2: $r_2 = (5\ 3\ 6\ 7\ 8\ 1\ 2\ 9\ 4)$

- $k_1 = 3$, $k_2 = 6$
- rodzic 1: $r_1 = (1\ 2 \mid 3\ 4\ 5\ 6 \mid 7\ 8\ 9)$
  rodzic 2: $r_2 = (5\ 3 \mid 6\ 7\ 8\ 1 \mid 2\ 9\ 4)$
- potomek 1: $p_1 = (\mathrm{X}\ \mathrm{X} \mid 3\ 4\ 5\ 6 \mid \mathrm{X}\ \mathrm{X}\ \mathrm{X})$
- rodzic 2: $r_2 = (2\ 9\ 4\ 5\ 3\ 6\ 7\ 8\ 1)$ a następnie po usunięciu wierzchołków: $r_2 = (2\ 9\ \mathrm{X}\ \mathrm{X}\ \mathrm{X}\ \mathrm{X}\ 7\ 8\ 1)$
- potomek 1: $p_1 = (8\ 1 \mid 3\ 4\ 5\ 6 \mid 2\ 9\ 7)$
- potomek 2: $p_2 = (4\ 5 \mid 6\ 7\ 8\ 1 \mid 9\ 2\ 3)$

Mutacja została wybrana jako mutacja transpozycyjna. Polega ona na zamianie dwóch losowych miast miejscami.

2 Plan eksperymentu - założenia projektowe

- struktury danych są alokowane dynamicznie,
- program umożliwia wczytanie danych testowych z plików dla grafów symetrycznych i asymetrycznych,
- program umożliwia wprowadzenie kryterium stopu algorytmu jako liczba iteracji,
- implementacja algorytmu jest zgodna z obiektowym paradygmatem programowania,
- program testowany jest dla znanych wyników dla ilości miast równej 17, 21, 24, 20, 48 oraz 120 dla grafu symetrycznego oraz dla ilości miast równej 17, 43, 53 i 70 dla grafu asymetrycznego,
- Każdy pomiar losowy powtórzony jest 1000 razy za każdym razem dla tego samego grafu, a następnie liczona jest średnia

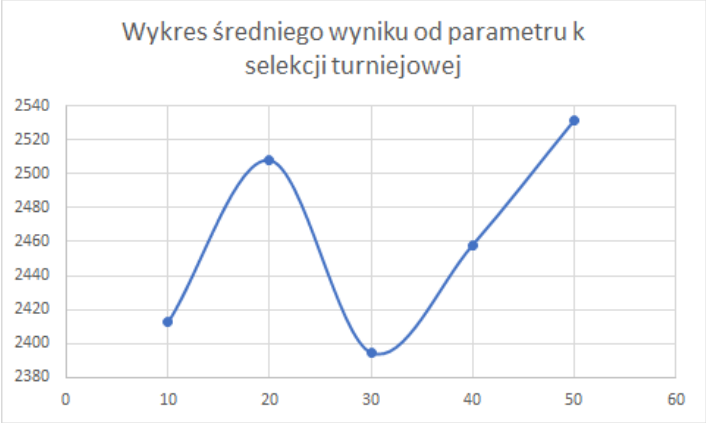
- do pomiarów czasu użyta została funkcji QueryPerformanceCounter. W pętli zapisysywanie zostały czasy dla kolejnych populacji, a z nich wyliczona została średnia,
- program napisany został w języku C++ w środowisku Microsoft Visual Studio 2017,
- waga krawędzi w przypadku losowanego grafu jest liczbą z przedziału (0,$RAND\_MAX$) wylosowaną za pomocą funkcji rand(),
- graf jest pełny zatem generuje wszystkie możliwe krawędzie (oprócz tych łączących wierzchołek z samym sobą). Graf może być symetryczny lub asymetryczny.

3 Zestawienie i analiza wyników

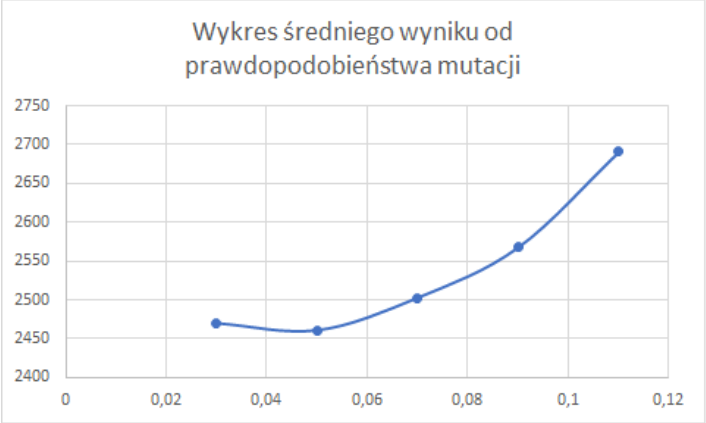
W poniższej analizie wyników testowany jest graf gr24.tsp z podanej wcześniej strony z grafami ze znanymi rozwiązaniami optymalnymi.

3.1 Analiza parametrów

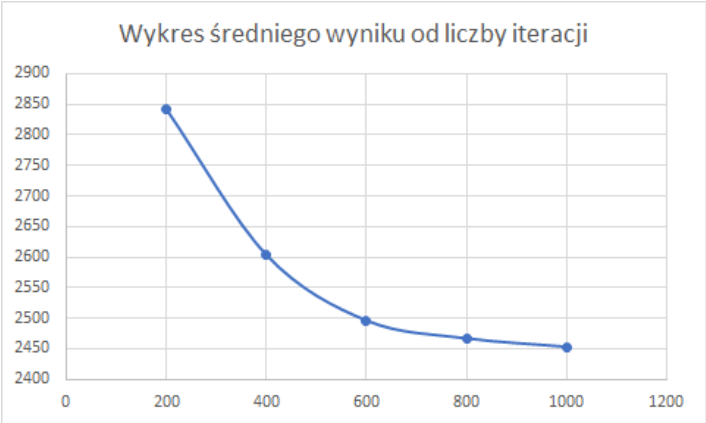
Zmiana parametru k – liczności grupy wybieranej przy selekcji turniejowej



Zmiana parametru prawdopodobieństwa mutacji

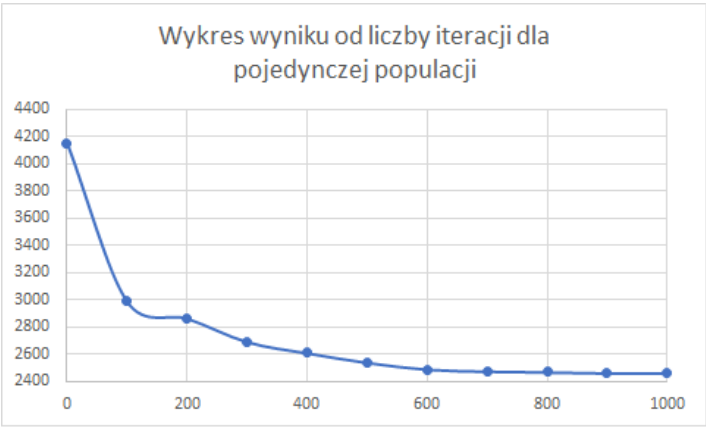


Zmiana liczby iteracji



3.2 Analiza jednej populacji dla znalezionych optymalnych parametrów

Na podstawie otrzymanych parametrów ($k = 30$, prawdopodobieństwo mutacji $= 0.05$ oraz liczba iteracji $= 1000$) stworzony został wykres dla pojedynczej populacji obrazujący zbieganie do optymalnego wyniku wraz ze wzrostem liczby iteracji.



3.3 Porównanie wyników i czasów wszystkich algorytmów

W tej części policzony zostanie błąd względny algorytmu populacyjnego, a następnie przedstawione zostanie porównanie błędu dwóch projektowych algorytmów aproksymacyjnych oraz czas wykonywania wszystkich trzech projektowych algorytmów.

Tabela 1: Błąd względny dla różnych grafów dla algorytmu populacyjnego

Nazwa pliku	Symetryczność	Wynik średni	Wynik optymalny	Błąd względny
gr17	symetryczny	2846	2085	0.36
gr21	symetryczny	4616	2707	0.70
gr24	symetryczny	2457	1272	0.93
bays29	symetryczny	3780	1610	1.34
hk48	symetryczny	37048	11461	2.23
gr120	symetryczny	43391	6942	5.25
br17	asymetryczny	63	39	0.61
p43	asymetryczny	6645	5620	0.18
ft53	asymetryczny	21614	6905	2.13
ft70	asymetryczny	64786	38673	0.67

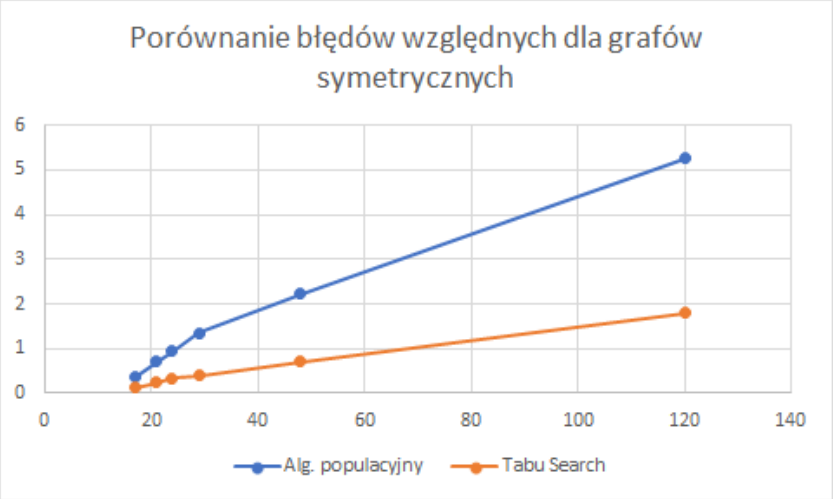
Tabela 2: Porównanie błędów względnych dla Tabu Search oraz algorytmu populacyjnego

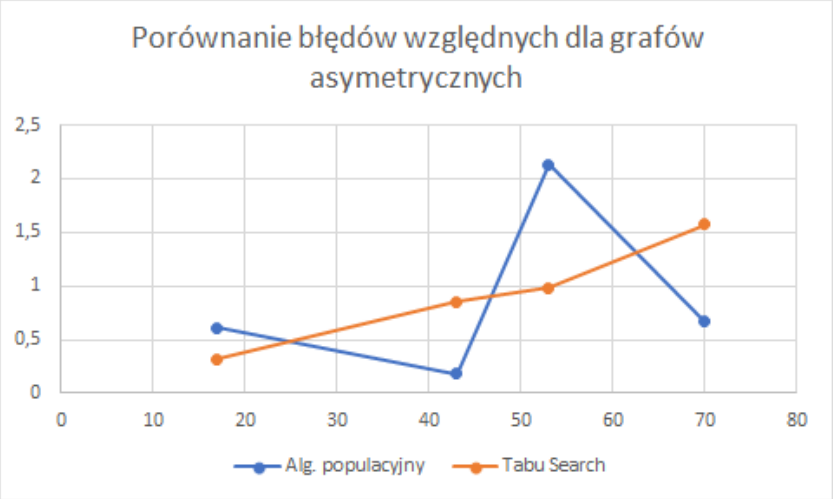
Nazwa pliku	Symetryczność	Tabu Search	Alg. populacyjny
gr17	symetryczny	0.13	0.36
gr21	symetryczny	0.24	0.70
gr24	symetryczny	0.34	0.93
bays29	symetryczny	0.39	1.34
hk48	symetryczny	0.70	2.23
gr120	symetryczny	1.80	5.25
br17	asymetryczny	0.32	0.61
p43	asymetryczny	0.85	0.18
ft53	asymetryczny	0.98	2.13
ft70	asymetryczny	1.57	0.67

Tabela 3: Porównanie czasu wykonywania dla algorytmu dynamicznego, Tabu Search oraz algorytmu populacyjnego

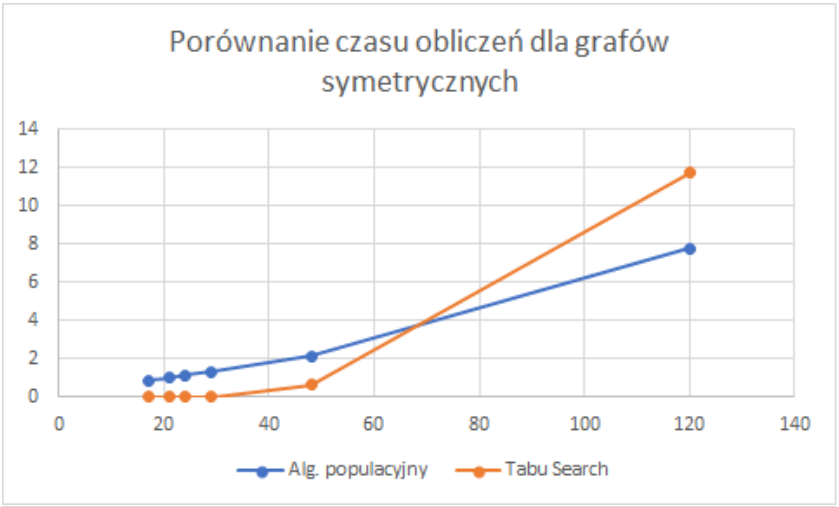
Nazwa pliku	Symetryczność	Alg. dynamiczny	Tabu Search	Alg. populacyjny
gr17	symetryczny	2.83	0.0025	0.89
gr21	symetryczny	86.30	0.0049	1.01
gr24	symetryczny	1018.76	0.0077	1.13
bays29	symetryczny	Brak danych	0.0137	1.35
hk48	symetryczny	Brak danych	0.6421	2.16
gr120	symetryczny	Brak danych	11.70	7.79
br17	asymetryczny	2.87	0.0025	0.88
p43	asymetryczny	Brak danych	0.537	2.03
ft53	asymetryczny	Brak danych	0.925	2.62
ft70	asymetryczny	Brak danych	2.31	3.96

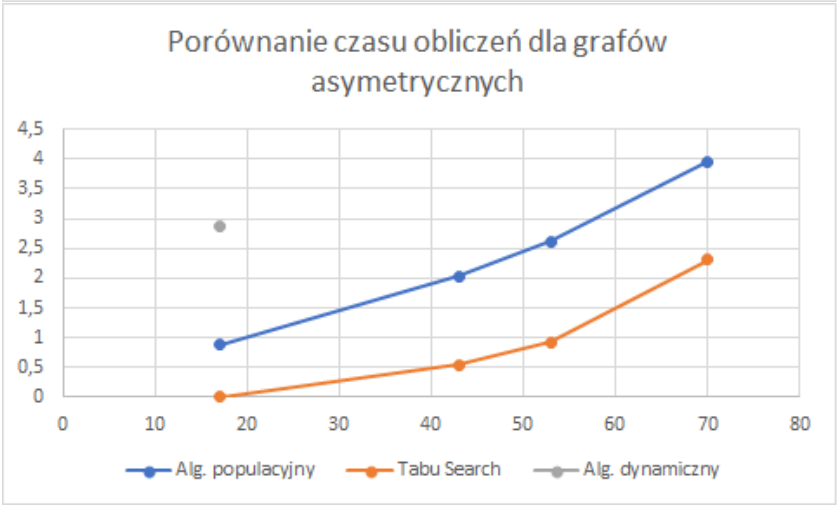
Porównanie błędów względnych





Porównanie czasów obliczeń





4 Wnioski

Wnioski

Zmiana parametru k nie wywołuje znacznej różnicy w poprawie wyników. W dalszej pracy przyjęta została $k = 30$ jako optymalna w testach.

Wraz z dużym wzrostem prawdopodobieństwa mutacji średni wynik znacznie się pogarsza. Dzieje się tak dlatego, że organizmy potomne stają się losowe przy zbyt częstych mutacjach.

Liczba iteracji ma duży wpływ na wynik końcowy, szczególnie dla małej liczby. Przy większych liczbach wpływ ten coraz bardziej maleje.

Błąd względny wyników zbadanych instancji są mniej więcej liniowe zatem zakładam że implementacja jest poprawna. Do rozwiązania optymalnego poszczególnych instancji algorytm powinien zostać specjalnie dostosowany do tej instancji. Algorytm populacyjny w większości przypadków ma większy błąd niż Tabu Search, jednak w niektórych przypadkach dla grafów symetrycznych błąd ten jest o wiele mniejszy (dla p43 i ft70).

Nieścisłości wyników mogą wynikać z losowości, niedokładności pomiaru czasu lub z tego, że niektóre instancje mają więcej minimów lokalnych od innych. Dodatkowo dla algorytmu populacyjnego istnieje o wiele większa losowość przy przeglądaniu wyników niż w przypadku Tabu Search.

Czas obliczeń dla małych instancji grafów jest mniejszy w przypadku Tabu Search, jednak wraz ze wzrostem liczby miast – szybszy staje się algorytm dynamiczny.

Z testów widać, że działanie dla grafu symetrycznego i asymetrycznego nie zmienia znacznie czasu działania algorytmu.

5 Bibliografia

1. Charles E. Leiserson. Ronald L. Rivest. Clifford Stein. *Introduction to Algorithms.* Third Edition. The MIT Press. Cambridge, Massachusetts London, England

2. A. Janiak, *Wybrane problemy i algorytmy szeregowania zadań i rozdziału zasobów*, PLJ 1999.

3. J. Knox, *Tabu search performance on the symmetric traveling salesman problem*, 1994