Prowadzący: dr inż. Tomasz Kapłon
termin projektu: wt TP, 15.15 – 16.55

Wrocław, 20.12.2016
Weronika Luźna, 209879

# Projektowanie efektywnych algorytmów
# Zadanie 2.

## Algorytm przeszukiwania tabu
## dla problemu komiwojażera.

# 1. Wstęp.

Celem zadania było zaimplementowanie i przetestowanie algorytmu metaheurystycznego (tabu search) dla problemu komiwojażera oraz porównanie wyników z algorytmem symulowanego wyżarzania realizowanego w zadaniu poprzednim.

Problem komiwojażera polega na odnalezieniu minimalnego cyklu Hamiltona w pełnym grafie – przejściu przez n miast jak najkrótszą drogą, odwiedzając każde z nich dokładnie raz. Istnieją dwa rodzaje tego problemu: symetryczny (odległość z punktu A do B jest identyczna jak z B do A) oraz niesymetryczny (odległości są różne w każdą stronę).

Algorytm wyszukiwania z zakazami (TS – tabu search) polega na nakładaniu ograniczeń i wykorzystywaniu ich w celu unikania minimów lokalnych i cykli, dla zwiększenia możliwości odnalezienia globalnie optymalnych rozwiązań podczas przeszukiwania przestrzeni rozwiązań. Sprowadza się do dynamicznej zmiany aktualnie znalezionego rozwiązania na najlepsze znajdujące się w jego sąsiedztwie (pod warunkiem, że nie ma go na liście rozwiązań zakazanych).

# 2. Implementacja.

Program powstał przy użyciu języka C++ w środowisku Microsoft Visual Studio 2015. Oparty jest o programowanie zorientowane obiektowo; składa się z pięciu klas. Pierwsza z nich: *Matrix* zawiera macierz, będącą reprezentacją grafu oraz metody potrzebne do parsowania i wczytywania danych. Druga klasa: *Solution* służy do przechowywania i obsługi rozwiązań (losowanie x0, zamienianie miejscami losowych elementów danego rozwiązania, operator przypisania). Trzecia klasa: *Move* definiuje obiekty agregowane w czwartej klasie: *TabuList,* realizującej kolejkę FIFO o ograniczonym maksymalnym rozmiarze (kadencji elementu). Ostatnia klasa *TabuSearch* realizuje algorytm od początku do końca, czyli zawiera:

- inicjalizację parametrów (rozmiar sąsiedztwa, rozmiar listy tabu, warunek zatrzymania – ilość iteracji)

- obliczanie długości ścieżki dla danej permutacji (kryterium aspiracji: ścieżka jak najkrótsza)

- opcjonalną dywersyfikację (Jeśli przez 2*n przejść nie zmieniono najlepszego rozwiązania losujemy 10 nowych rozwiązań i wybieramy z niego najlepsze)

- algorytm, który przedstawić można poniższym pseudokodem:

```
s ← s0
sBest ← s

tabuList ← []

while (not stoppingCondition())

        candidateList ← []

        bestCandidate ← null

        for (sCandidate in sNeighborhood)

                if ( (not tabuList.contains(sCandidate)) and
(fitness(sCandidate) > fitness(bestCandidate)) )

                        bestCandidate ← sCandidate

                end

        end

        s ← bestCandidate

        if (fitness(bestCandidate) > fitness(sBest))

                sBest ← bestCandidate

        end

        tabuList.push(bestCandidate);

        if (tabuList.size > maxTabuSize)

                tabuList.removeFirst()

        end

end

return sBest
```

Przy czym ostatni warunek sprawdzający rozmiar listy tabu został przeniesiony do metody klasy ją realizującej (TabuList) – podczas dodawania nowego zakazu ( Add(move) ) ostatni na liście zostaje automatycznie usuwany, jeżeli wychodzi poza zadeklarowany rozmiar tabu.

## 3. Testowanie.

Do testów wybrano trzy instancje symetryczne i trzy asymetryczne o znanych rozwiązaniach optymalnych: (gr21, dantzig42, gr120). Dla każdej z nich zmierzono czas działania algorytmu przy zmiennych parametrach wejściowych. Pomiaru czasu dokonano przy pomocy gotowych funkcji z biblioteki <windows.h>.

Ustalono stały warunek przerwania: 1000 iteracji oraz stały rozmiar listy tabu: n (ilość miast).

Wykonano pomiary dla czterech wartości rozmiaru sąsiedztwa: S ={n, 2n, 10n}.

I. Instancja gr21.tsp, n=21, wynik optymalny: 2707.

II. Instancja dantzig42.tsp, n=42, wynik optymalny: 699.

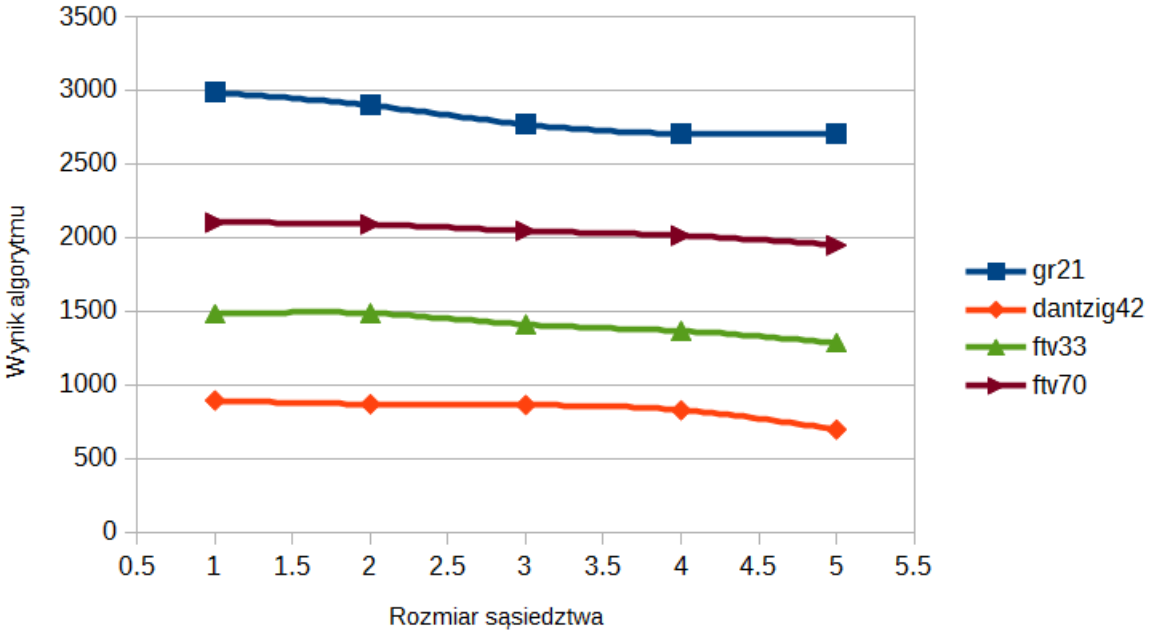
III. Instancja gr120.tsp, n=120, wynik optymalny: 6942.

IV. Instancja ftv33.atsp, n=34, wynik optymalny: 1286.

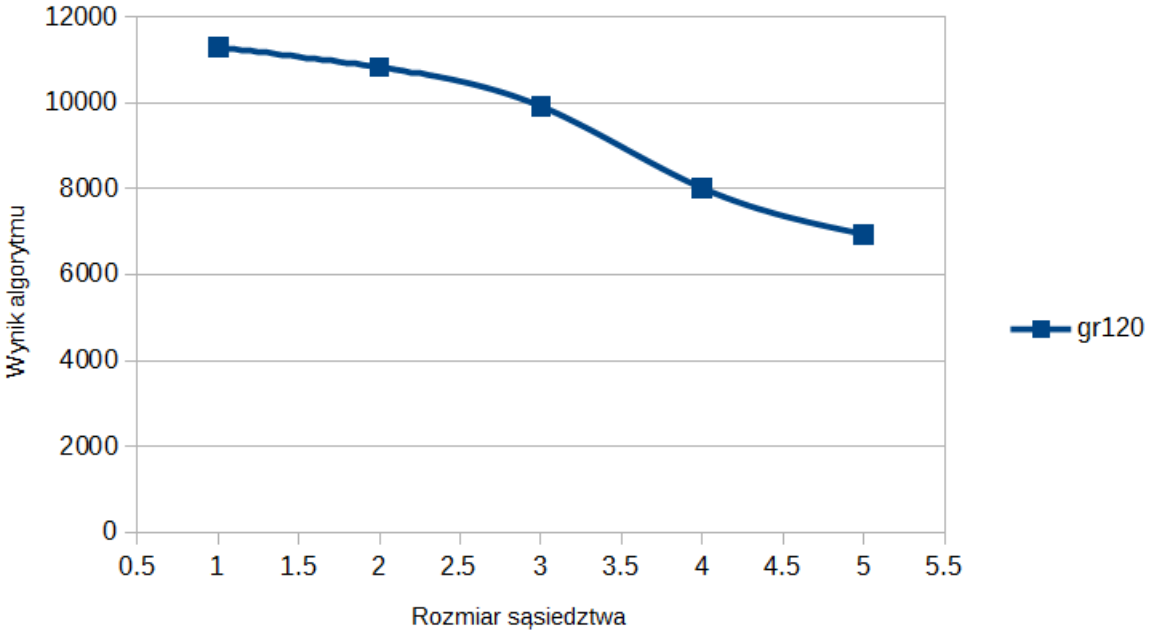
V. Instancja ftv70.atsp, n=71, wynik optymalny: 1950.

Tabela 1. Zestawienie wyników algorytmu i pomiarów czasu dla 5 instancji.

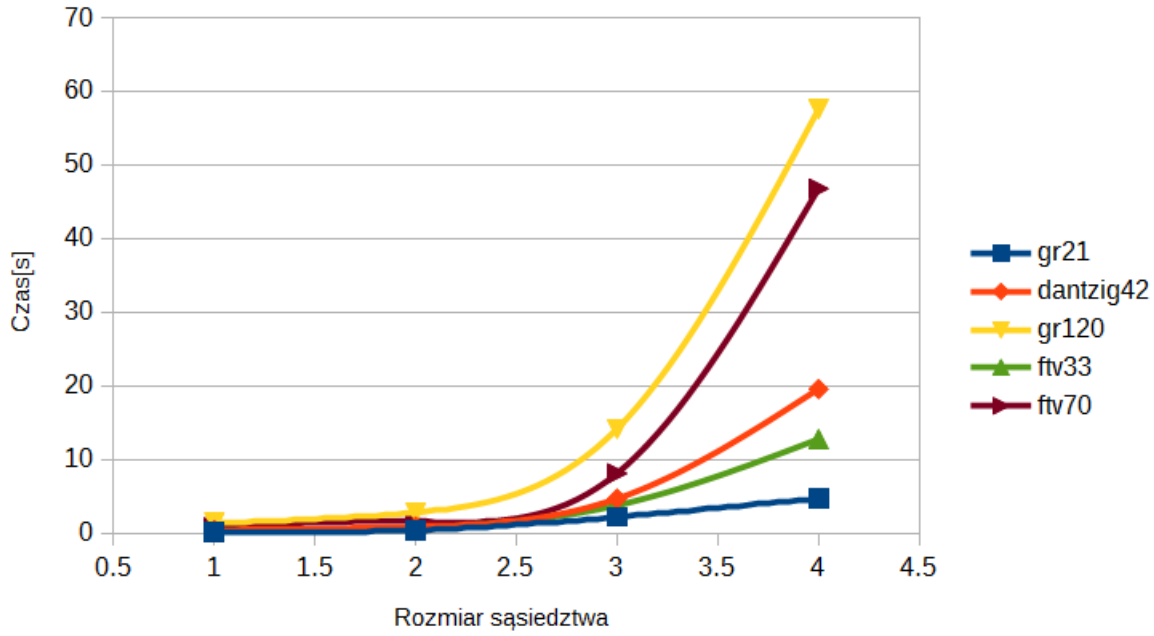
| **S** / **instancja** | **n** | **2n** | **10n** | **$n^2$** |
|---|---|---|---|---|
| | **WYNIK** | | | |
| **gr21** | 2987 | 2898 | 2769 | 2707 |
| **dantzig42** | 896 | 871 | 865 | 830 |
| **gr120** | 11295 | 10826 | 9918 | 7014 |
| **ftv33** | 1482 | 1488 | 1412 | 1368 |
| **ftv70** | 2104 | 2092 | 2048 | 2016 |
| | **CZAS [s]** | | | |
| **gr21** | 0.239 | 0.461 | 2.262 | 4.769 |
| **dantzig42** | 0.482 | 0.938 | 4.726 | 19.618 |
| **gr120** | 1.452 | 2.877 | 14.215 | 57.593 |
| **ftv33** | 0.393 | 0.757 | 3.830 | 12.818 |
| **ftv70** | 0.818 | 1.637 | 8.167 | 46.823 |




Wykres 1. Zależność optymalności wyniku od rozmiaru sąsiedztwa.




Wykres 2. Zależność optymalności wyniku od rozmiaru sąsiedztwa dla dużej instancji.



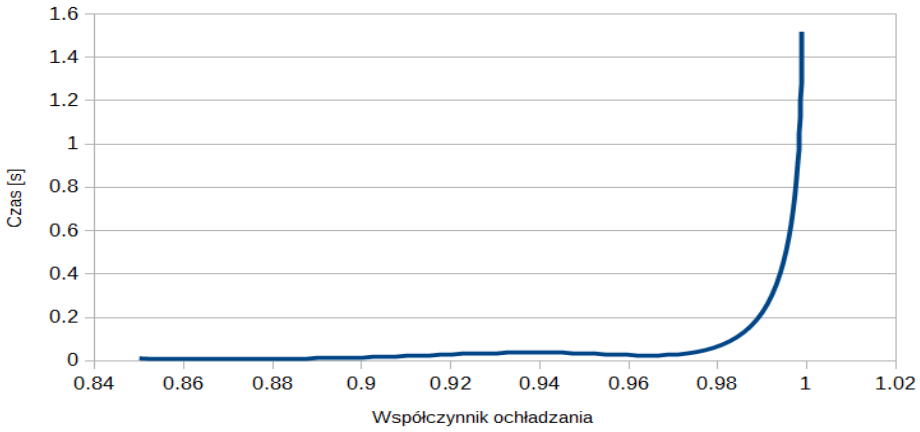

Wykres 3. Zależność czasu od rozmiaru sąsiedztwa.

## 4. Porównanie przeszukiwania tabu (TS) i symulowanego wyżarzania (SA).

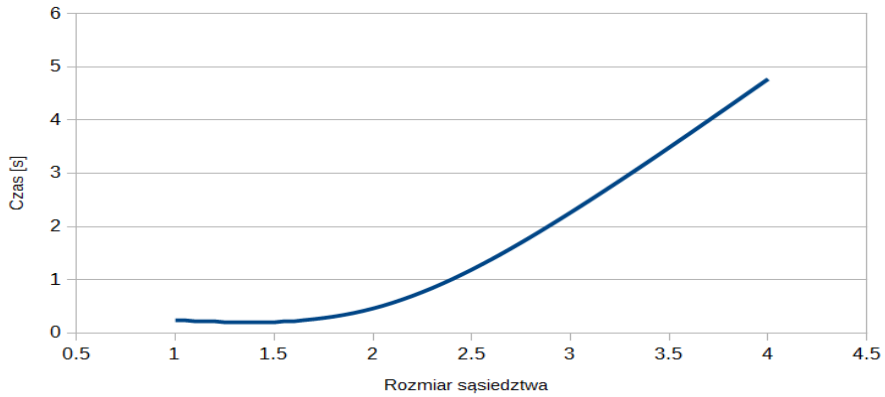
Zebarano najbardziej optymalne wyniki, jakie udało się uzyskać przy pomocy jednego i drugiego algorytmu i obliczono dla nich średni błąd.

Tabela 2. Zestawienie najlepszych wyników.

| instancja | TS | Błąd TS [%] | SA | Błąd SA [%] | optimum |
|---|---|---|---|---|---|
| **gr21** | 2707 | 0 | 2707 | 0 | 2707 |
| **dantzig42** | 830 | 18.74 | 701 | 2.86 | 699 |
| **gr120** | 7014 | 1.03 | 7291 | 5.02 | 6942 |
| **ftv33** | 1368 | 6.37 | 1548 | 20.37 | 1286 |
| **ftv70** | 2016 | 3.38 | 2640 | 35.38 | 1950 |
| | | **5.90** | | **12.73** | |




Wykres 4. Zależność czasu od parametryzacji dla SA.




Wykres 5. Zależność czasu od parametryzacji dla TS.

## 5. Wnioski.

W trakcie realizacji projektu zauważono, że dobór parametrów wejściowych dla algorytmu ma wpływ na jakość zwracanego rozwiązania, ale w mniejszym stopniu niż przy symulowanym wyżarzaniu. Zależność tą widać na wykresach 1. i 2. Wraz ze zwiększaniem przeglądanego sąsiedztwa, wynik zbliża się powoli do optimum. Im większa instancja problemu, tym ostrzejsza jest zmiana.

Na wykresie 3. widać wzrost czasu dla wzrostu rozmiaru sąsiedztwa. Im większa instacja, tym większa rozbieżność między czasem początkowym a końcowym.

Z danych zebranych w tabeli 2. wynika, iż implementacja przeszukiwania z zakazami okazała

się być efektywniejsza  - średni błąd dla 5 przetwarzanych instancji wyniósł 5.90%, natomiast dla symulowanego wyżarzania jest to 12.73 %.

Na wykresach 4. 5. nakreślono krzywe czasowe, odrębnie dla dwóch algorytmów. Widać, że dla parametryzacji symulowanego wyżarzania, czas ostro rośnie w bardzo małym zakresie, natomiast dla tabu search, jest to wzrost mniej stromy.