**POLITECHNIKA WROCŁAWSKA**

**Zespół Inżynierii Oprogramowania i Inteligencji Obliczeniowej**

**Katedra Informatyki Technicznej**

**Projektowanie efektywnych algorytmów** **- projekt**

**Kurs: INEK032**

**Sprawozdanie z projektu**

|  |  |
| --- | --- |
| **Wykonał:** | **Michał Madarasz, 238903** |
| **Termin:** | **Czwartek 17:05** |
| **Prowadzący:** | **Mgr inż. Antoni Sterna** |
| **Data oddania sprawozdania:** | **14.01.2019** |
| **Ocena:** |  |

|  |
| --- |
| **Uwagi prowadzącego:** |

# Wstęp teoretyczny

Problem komiwojażera jest jednym z najpopularniejszych i najbardziej znanych problemów optymalizacyjnych dla informatyka. Problem polega na znalezieniu najkrótszej ścieżki prowadzącej z wierzchołka startowego, która przechodzi dokładnie jeden raz przez każdy wierzchołek i wraca do wierzchołka startowego. Innymi słowy jest to problem znalezienia minimalnego cyklu Hamiltona w grafie pełnym ważonym. W języku angielskim problem ten nazywa się *Travelling Salesman Problem* (zwany dalej *TSP*).

Jest to problem należący do grupy problemów NP-trudnych, to oznacza, że złożoność obliczeniowa rozwiązania problemu jest wykładnicza i nie da się rozwiązać problemu w normalnym czasie. Zagadnienie TSP jest skomplikowane, ponieważ ogólnie komiwojażer przy wyborze trasy przez *N* miast ma do rozpatrzenia (N-1)! (silnia) przypadków. W problemie tym rozróżniamy także podział na dwa przypadki:

* Symetryczny – gdzie graf jest nieskierowany (ang. *Symmetric Travelling Salesman Problem*),
* Asymetryczny – gdzie graf jest skierowany (ang. *Asymetric Travelling Salesman Problem*);

Alternatywnym sposobem rozwiązania TSP jest skorzystanie z algorytmów metaheurystycznych zajmujących się tym zagadnieniem. Algorytmów metaheurystycznych można używać do rozwiązania dowolnego problemu, ponieważ algorytmy te proponują jedynie przybliżony sposób rozwiązania, więc rozwiązanie może nie być prawidłowe. Jednakże złożoność obliczeniowa użycia takich algorytmów jest znacznie mniejsza niż przy normalnym rozwiązaniu.

W tym projekcie zostaje przedstawione rozwiązanie ATSP za pomocą trzech algorytmów:

* Symulowane wyżarzanie (ang. Simulated Anealing) – Algorytm symulowanego wyżarzania, to algorytm metaheurystyczny opierający swoje działanie na losowości. Nie znajduje on zawsze najlepszego rozwiązania problemu, w taki sposób jak przegląd zupełny, jednak jego działanie prowadzi do znalezienia rozwiązania optymalnego – trasy, która będzie krótka, ale nie koniecznie najkrótsza z możliwych. Jego działanie oraz nazwa biorą się z analogii do procesu wyżarzania w metalurgii. W trakcie działania algorytmu, stopniowo, wraz z kolejnymi iteracjami pętli, obniżana jest temperatura. Algorytm zaczyna od pewnego rozwiązania początkowego i w kolejnych iteracjach zamienia miejscami losowe elementy trasy (miasta/wierzchołki). Jeśli po zamianie, trasa jest lepsza – krótsza, zostaje ona zapisana jako dotychczasowo najlepsza, jeśli nie, odrzucamy ją i zamieniamy inne elementy. Aby podczas tych zamian, algorytm nie utknął w minimum lokalnym, dopuszczalne są także zmiany pogarszające obecnie uzyskany wynik, ale pozwalające dojść do lepszego rozwiązania w kolejnych iteracjach.
* Przeszukiwanie Tabu (ang. Tabu Search) - metahuerystyka używana do rozwiązywania problemów optymalizacyjnych badając sąsiedztwo rozwiązania znajdującego się w przestrzeni rozwiązań problemu oraz zapamiętując wykonywane ostatnio ruchy. Podstawą tej metody jest zapamiętywanie ruchów na **liście tabu** blokuje jego wykonywanie w pewnej ilości następnych kroków w zależności od przyznanej im **kadencji**. Zapisanie ruchów niedozwolonych w tabu pozwala na odrzucenie rozwiązań niedawno sprawdzane zwiększając obszar przeszukiwania, co skutkuje większą możliwością wyjścia z minimum lokalnego, ale kosztem dokładności algorytmu.

## 1.1 Opis działania algorytmu



### 1.1.2 Symulowane wyżarzanie

Wygeneruj punkt startowy x0 ∈ X

T ← Tmax

xopt ← x0

repeat

Wybierz w sposób losowy x ∈ N(x0)

if random[0, 1) < P(x0, x) then

x0 ← x

if f(x0) < f(xopt) then

xopt ← x0

end if

end if

T ← g(T) {Obniz temperature}

until warunek konca=true

### 1.1.2 Tabu Search

Wygeneruj losowo lub heurystycznie punkt startowy x0 ∈ X

xopt ← x0

TABU ← ∅

repeat

x0 ← AspirationPlus(x0) {Lub inna strategia wyboru rozwiązania}

if f(x0) < f(xopt) then

xopt ← x0

end if

Dodaj nowe elementy do listy TABU

for all (atrybuti, kadencjai) ∈ TABU do

kadencjai ← kadencjai − 1

if kadencjai = 0 then

usun\_element (atrybuti, kadencjai) z T ABU

end if

end for

if CriticalEvent() = true then

x0 ← Restart() {Dywersyfikacja}

if f(x0) < f(xopt) then

xopt ← x0

end if

end if

until warunek zakonczenia = true

# Opis projektu

Założenia projektowe:

* Struktury przechowujące dane alokowane są dynamicznie, zależnie od rozmiaru problemu;
* Program posiada możliwość wczytywania danych z pliku, w celu weryfikacji poprawności;
* Algorytm został zaprojektowany zgodnie z paradygmatami programowania obiektowego;
* Program umożliwia wprowadzenie kryterium stopu jako czas podany w sekundach;
* Program umożliwia włączenie/wyłączenie dywersyfikacji w algorytmie Tabu Search;

W programie zostały wykorzystane następujące klasy:

* Menu – klasa obsługująca interfejs użytkownika,
* SA – klasa implementująca algorytm symulowanego wyżarzania,
* Tabu – klasa implementująca algorytm przeszukiwania tabu;

# Plan testów

Do pomiarów zostały wykorzystane macierze br17, ftv70, ftv170 ze strony <http://elib.zib.de/pub/mp-testdata/tsp/tsplib/atsp/index.html>. Zostały one przerobione na pliki testowe, gdzie w pierwszej linii był podany rozmiar macierzy oraz najlepszy znany wynik dla tej macierzy.

Mierzenie czasu odbywało się przy pomocy funkcji *QueryPerformanceCounter()* oraz *QueryPerformanceFrequency()* z biblioteki <Windows.h>. Aby uzyskać czas należało zliczoną liczbę impulsów licznika podzielić przez częstotliwość. Zastosowane rozwiązanie pozwalało na odpowiednie badanie czasu wykonania algorytmu.

Parametry uruchamiania algorytmu Symulowanego wyżarzania:

* Czas wykonywania algorytmu – 30 sekund, 60 sekund;
* Temperatura początkowa:

;

* Zmiana temperatury:

, gdzie , i *maxit* maksymalna liczba iteracji;

Parametry uruchamiania Tabu Search:

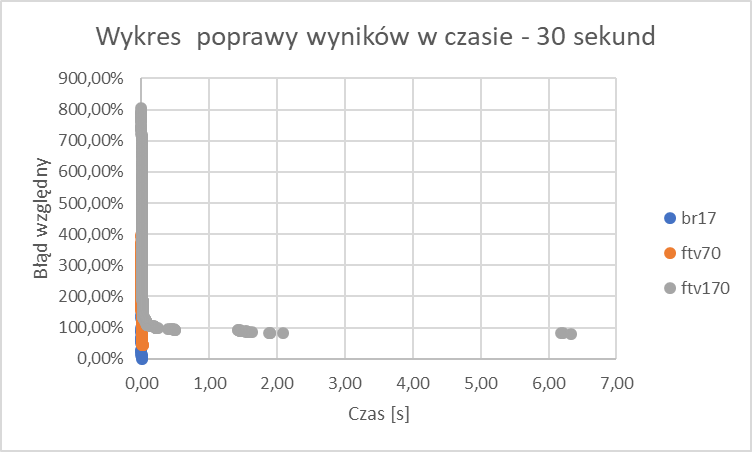
* Czas wykonywania algorytmu – 30 sekund, 60 sekund;
* Dywersyfikacja następuje po 10 powtórzeniach;
* Kryterium aspiracji następuje, jeżeli rozwiązanie na liście Tabu jest lepsze o 40%;
* Stała kadencja – 10;

# Wyniki testów

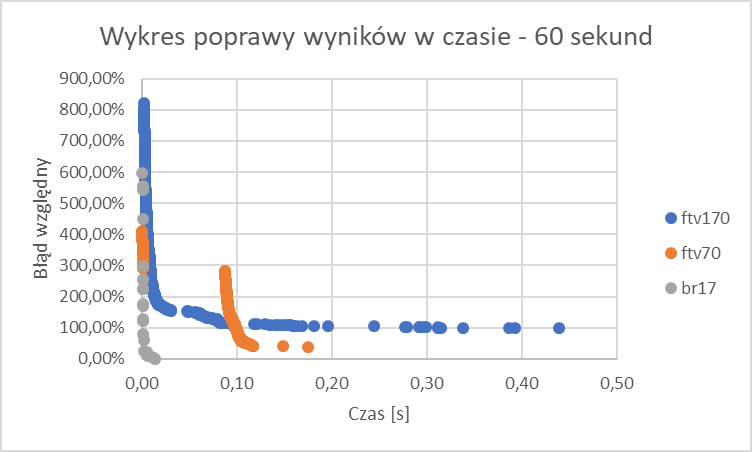
## 5.1 Symulowane wyżarzanie

|  |  |  |  |
| --- | --- | --- | --- |
|  | **30 sekund** | | |
|  | **br17** | **ftv70** | **ftv170** |
| **Optymalna długość ścieżki** | **39** | **1950** | **2755** |
| **Znaleziona długość ścieżki** | **39** | **2549** | **3215** |
| **Czas wykonania** | **29,708** | **29,2403** | **29,5545** |
| **Błąd** | **0** | **30,7179** | **16,6969** |

## 

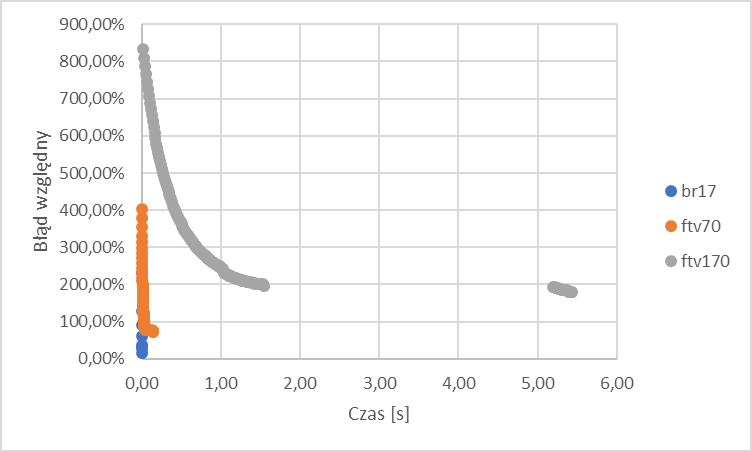


|  |  |  |  |
| --- | --- | --- | --- |
|  | **60 sekund** | | |
|  | **br17** | **ftv70** | **ftv170** |
| **Optymalna długość ścieżki** | **39** | **1950** | **2755** |
| **Znaleziona długość ścieżki** | **39** | **2551** | **5306** |
| **Czas wykonania** | **30,857** | **36,6472** | **59,6324** |
| **Błąd** | **0** | **30,8205** | **92,5953** |

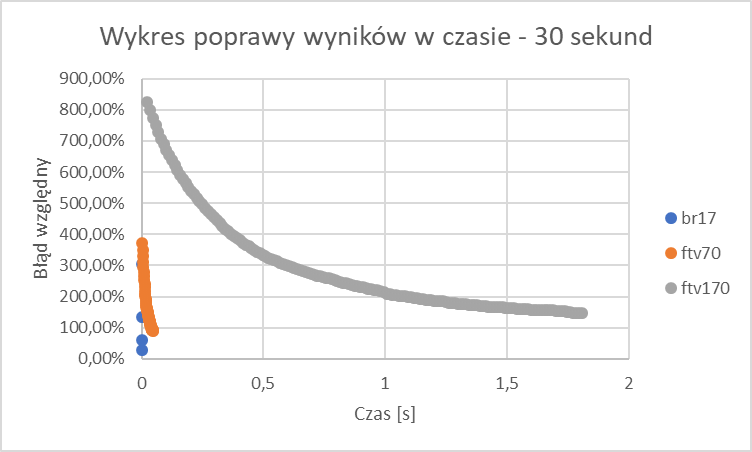


## 5.2 Tabu Search

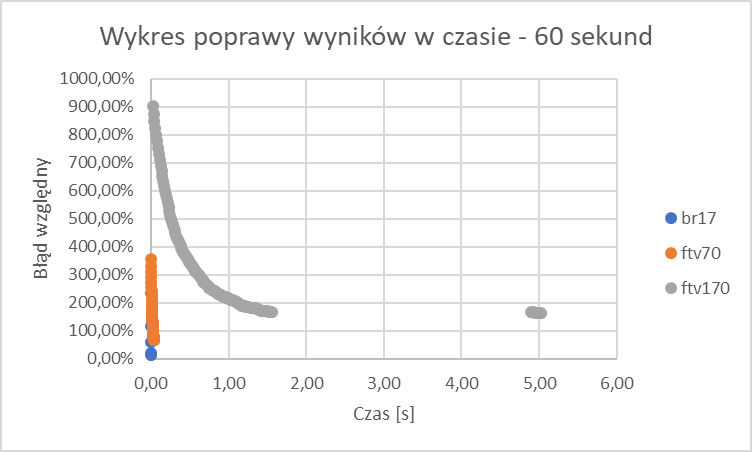
|  |  |  |  |
| --- | --- | --- | --- |
|  | **30 sekund - dywersyfikacja** | | |
|  | **br17** | **ftv70** | **ftv170** |
| **Optymalna długość ścieżki** | **39** | **1950** | **2755** |
| **Znaleziona długość ścieżki** | **39** | **3385** | **7628** |
| **Czas wykonania** | **29,7517** | **29,7035** | **29,8722** |
| **Błąd** | **0** | **73,5897** | **176,878** |



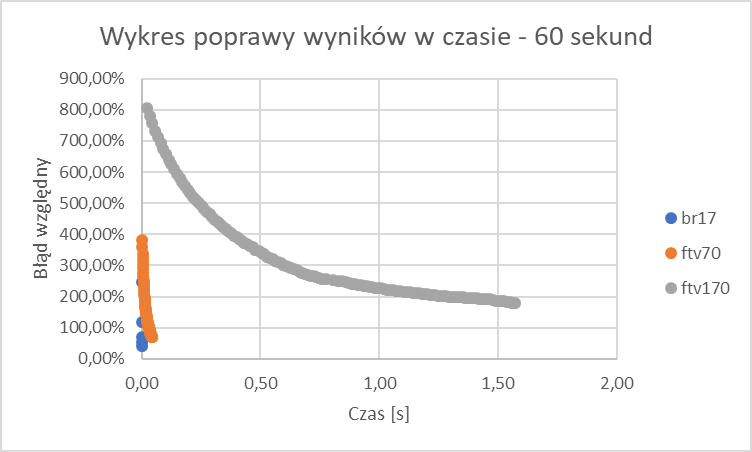
|  |  |  |  |
| --- | --- | --- | --- |
|  | **30 sekund – bez dywersyfikacji** | | |
|  | **br17** | **ftv70** | **ftv170** |
| **Optymalna długość ścieżki** | **39** | **1950** | **2755** |
| **Znaleziona długość ścieżki** | **42** | **3215** | **8222** |
| **Czas wykonania** | **29,5761** | **29,867** | **29,3294** |
| **Błąd** | **7,69231** | **64,8718** | **198,439** |



|  |  |  |  |
| --- | --- | --- | --- |
|  | **60 sekund - dywersyfikacja** | | |
|  | **br17** | **ftv70** | **ftv170** |
| **Optymalna długość ścieżki** | **39** | **1950** | **2755** |
| **Znaleziona długość ścieżki** | **44** | **3316** | **7231** |
| **Czas wykonania** | **59,8122** | **59,907** | **59,5299** |
| **Błąd** | **12,8205** | **70,0513** | **162,468** |



|  |  |  |  |
| --- | --- | --- | --- |
|  | **60 sekund – bez dywersyfikacji** | | |
|  | **br17** | **ftv70** | **ftv170** |
| **Optymalna długość ścieżki** | **39** | **1950** | **2755** |
| **Znaleziona długość ścieżki** | **49** | **3223** | **7772** |
| **Czas wykonania** | **59,3355** | **59,9751** | **59,9719** |
| **Błąd** | **25,641** | **65,2821** | **182,105** |



# Wnioski

* Algorytmy poszukiwania lokalnego są niedokładne, to znaczy, że nie zawsze znajdują najlepsze rozwiązanie. W zamian znajdują rozwiązania bliskie optymalnemu w krótkim czasie (w porównaniu do algorytmów dokładnych);
* Implementacja algorytmu to jedynie połowa sukcesu. Oprócz napisania algorytmu bardzo ważnym elementem jest dobór parametrów najlepszych dla problemu. W zależności od rozmiaru grafu jak i odchylenia wartości średniej przejść można wnioskować jakie parametry będą dawały najlepsze wyniki;
* Strategia wykorzystania tych algorytmów opiera się na ich szybkości. Są na tyle szybkie, że najlepiej uruchamiać je wiele razy dla tego samego problemu i wyciągać optymalne rozwiązanie.

Spis treści

[1. Wstęp teoretyczny 2](#_Toc535504781)

[1.1 Opis działania algorytmu 3](#_Toc535504782)

[1.1.2 Symulowane wyżarzanie 3](#_Toc535504785)

[1.1.2 Tabu Search 3](#_Toc535504786)

[3. Opis projektu 4](#_Toc535504787)

[4. Plan testów 4](#_Toc535504788)

[5. Wyniki testów 4](#_Toc535504789)

[5.1 Symulowane wyżarzanie 4](#_Toc535504790)

[5.2 Tabu Search 6](#_Toc535504791)

[6. Wnioski 8](#_Toc535504792)