**Politechnika Wrocławska**

**Wydział Informatyki i Telekomunikacji**

|  |
| --- |
| **Projektowanie efektywnych algorytmów**  Projekt nr 3 - Implementacja i analiza efektywności algorytmu Tabu Search i Symulowanego Wyżarzania dla problemu komiwojażera |

semestr zimowy 2023/2024

Prowadzący:  
Dr inż. Marcin Łopuszyński

Autor:  
Eryk Mika 264451

Spis treści

[1. Wstęp teoretyczny 1](#_Toc153294545)

[1.1. *Tabu Search* 1](#_Toc153294546)

[1.2. Symulowane wyżarzanie 2](#_Toc153294547)

[1.3. Definicje sąsiedztwa 2](#_Toc153294548)

[2. Opis implementacji algorytmów 3](#_Toc153294549)

[2.1. Klasa *Graph* 3](#_Toc153294550)

[2.2. Klasa Graph – metoda *generateInitialSolution()* 3](#_Toc153294551)

[2.3. Klasa *Graph* – metoda *solveSimulatedAnnealing()* 3](#_Toc153294552)

[2.1. Klasa *Graph* – metoda *generateInitialTemp()* 6](#_Toc153294553)

[2.2. Klasa Graph – metoda *solveTabuSearch()* 6](#_Toc153294554)

[2.3. Klasa *Route* 8](#_Toc153294555)

[3. Sposób przeprowadzenia badania 10](#_Toc153294556)

[4. Wyniki przeprowadzonego badania 10](#_Toc153294557)

[4.1. Wszystkie wyniki 10](#_Toc153294558)

[4.2. Najlepsze rozwiązania 14](#_Toc153294559)

[5. Wnioski 15](#_Toc153294560)

[6. Bibliografia 16](#_Toc153294561)

# Wstęp teoretyczny

Zgodnie z informacjami przedstawionymi w poprzednich sprawozdaniach, problem komiwojażera jest problemem trudnym pod względem obliczeniowym. W tym opracowaniu zostanie omówione rozwiązanie tego problemu z wykorzystaniem algorytmów przeszukiwania z tabu (ang. *Tabu Search*, TS) oraz symulowanego wyżarzania (ang. *Simulated Annealing*, SA). Są to algorytmy przeszukiwania lokalnego (ang. *local search*) – **zazwyczaj** pozwalają one na znalezienie rozwiązania **zbliżonego** do optimum danego problemu optymalizacyjnego (globalnego maksimum lub minimum), jednak **nie muszą** zapewnić znalezienia tego optimum[[1]](#footnote-1).

## *Tabu Search*

*Tabu Search* (TS) to heurystyczna (taka, która nie gwarantuje znalezienia rozwiązania optymalnego lub nawet prawidłowego)[[2]](#footnote-2) metoda optymalizacyjna używana do rozwiązywania problemów optymalizacji kombinatorycznej. Została wprowadzona przez Freda Glovera w latach 80. i opiera się na koncepcji przeszukiwania przestrzeni rozwiązań przy użyciu mechanizmu tabu (ruchów zakazanych), który zapobiega powtarzaniu tych samych ruchów lub rozwiązań w trakcie przeszukiwania.

Algorytm zaczyna od wygenerowania początkowego rozwiązania, które jest następnie oceniane pod kątem funkcji celu. Tworzone są sąsiednie rozwiązania poprzez wprowadzanie określonych zmian (ruchów) do obecnie rozpatrywanego rozwiązania. To sąsiedztwo jest kluczowe dla algorytmu, ponieważ eksplorowana jest przestrzeń rozwiązań, szukając lepszych wartości funkcji celu. Algorytm ocenia każde nowe rozwiązanie za pomocą funkcji celu, która przypisuje wartość jakości danego rozwiązania. Celem jest minimalizacja lub maksymalizacja tej funkcji, zależnie od charakterystyki problemu. Utrzymywana jest lista tabu, która zawiera informacje na temat ostatnio odwiedzonych rozwiązań lub ruchów. Mechanizm tabu uniemożliwia powrót do niedawno odwiedzonych stanów, co pomaga uniknąć cykli i skuteczniej przeszukiwać przestrzeń rozwiązań. Pomimo tego, że ruch może być oznaczony jako zakazany na liście tabu, kryterium aspiracji może zezwalać na jego wykonanie, jeśli jest on korzystny w danej sytuacji. Algorytm składa się z faz intensyfikacji (skupiającej się na lokalnym przeszukiwaniu wokół obecnie najlepszego rozwiązania) i dywersyfikacji (poszerzającej przeszukiwanie w poszukiwaniu nowych obszarów rozwiązań). Algorytm działa do momentu spełnienia określonego warunku stopu, na przykład osiągnięcia maksymalnej liczby iteracji lub uzyskania satysfakcjonującego rozwiązania[[3]](#footnote-3).

## Symulowane wyżarzanie

Symulowane wyżarzanie to algorytm heurystyczny, którego nazwa nawiązuje do procesu wyżarzania metali. Algorytm został zainspirowany procesem chłodzenia stopionego metalu, w którym stopiony metal jest stopniowo schładzany, co pozwala na osiągnięcie bardziej stabilnej struktury krystalicznej. W przypadku symulowanego wyżarzania, proces chłodzenia jest odwzorowany w celu znalezienia globalnego optimum w przestrzeni rozwiązań.

Algorytm rozpoczyna od losowego rozwiązania początkowego. Podobnie jak w przypadku *tabu search*, symulowane wyżarzanie korzysta z funkcji celu do oceny jakości rozwiązania. Algorytm wprowadza pojęcie temperatury, która kontroluje prawdopodobieństwo akceptacji gorszych rozwiązań. W początkowej fazie temperatura jest wysoka, co pozwala na akceptowanie większej liczby gorszych rozwiązań. W miarę postępu algorytmu temperatura maleje, co sprawia, że akceptacja gorszych rozwiązań staje się coraz mniej prawdopodobna. Prawdopodobieństwo zaakceptowania gorszego rozwiązania jest związane z różnicą wartości funkcji celu oraz aktualnej temperatury. Im wyższa temperatura, tym większa szansa na akceptację gorszego rozwiązania. Algorytm działa do momentu spełnienia określonego warunku stopu, na przykład osiągnięcia maksymalnej liczby iteracji lub uzyskania satysfakcjonującego rozwiązania. Symulowane wyżarzanie jest stosowane w przypadkach, gdzie przestrzeń rozwiązań jest duża i skomplikowana, a funkcja celu może zawierać wiele lokalnych optimów. Algorytm pozwala na unikanie zatrzymywania się w lokalnych optimum poprzez akceptację czasami gorszych rozwiązań na początku procesu optymalizacji, a następnie stopniowe zaostrzanie kryteriów akceptacji w miarę postępu algorytmu. To podejście pozwala na przeszukiwanie przestrzeni rozwiązań w sposób bardziej elastyczny i bardziej zrównoważony[[4]](#footnote-4).

## Definicje sąsiedztwa[[5]](#footnote-5)

Sąsiedztwo dla *tabu search* i symulowanego wyżarzania jest kluczowym elementem, ponieważ wpływa na sposób generowania sąsiednich rozwiązań podczas przeszukiwania przestrzeni rozwiązań. Dla wielu problemów optymalizacyjnych, takich jak problem komiwojażera, stosuje się trzy popularne operatory sąsiedztwa: *swap* (zamiana miejscami), *insert* (wstawianie) i *inverse* (odwracanie):

* *swap* - polega na zamianie dwóch elementów w rozwiązaniu. Dla problemu trasowania, swap może oznaczać zamianę dwóch miast w trasie,
* i*nsert* - polega na przeniesieniu jednego elementu z jednego miejsca w rozwiązaniu i wstawieniu go w inne miejsce. Dla problemu trasowania, insert może oznaczać przeniesienie miasta z jednej pozycji w trasie i wstawienie go w inne miejsce,
* *inverse* - polega na odwróceniu kolejności pewnego fragmentu rozwiązania. Dla problemu trasowania, *inverse* może oznaczać odwrócenie kolejności miast pomiędzy dwoma wybranymi punktami w trasie.

# Opis implementacji algorytmów

W celu analizy efektywności omawianych algorytmów został napisany program w języku C++ z wykorzystaniem obiektowego paradygmatu programowania. Najistotniejszymi komponentami aplikacji są klasy *Graph* oraz *Route*, których pola (struktury danych) oraz metody są odpowiedzialne za realizację algorytmu. Wiele istotnych kwestii związanych z implementacją zostało wyjaśnionych w komentarzach w plikach źródłowych.

## Klasa *Graph*

Klasa Graph jest główną klasą programu, która jest odpowiedzialna za przechowywanie struktury i metod grafu, na którym wykonywane są badane algorytmy. Pola prywatne klasy – dwuwymiarowa tablica *std::vector matrix* oraz *size* są użyte do przechowywania długości krawędzi w postaci macierzy kwadratowej – kosztów - *matrix* stopnia *size*. Oba pola przechowują liczby stałoprzecinkowe typu *int*. W macierzy komórka o współrzędnych *i*, *j* zawiera odległość pomiędzy wierzchołkami *i* *i j*.

Zaimplementowano konstruktor wczytujący instancję z pliku tekstowego, przeładowany operator przypisania oraz metodę wypisującą graf (macierz) na ekran. Została zaimplementowana metoda *calculateRouteCost()*, która służy do obliczania kosztu danej trasy komiwojażera w grafie – poprzez iterowanie po krawędziach w ścieżce i dodanie ich długości do zwracanej sumy.

## Klasa Graph – metoda *generateInitialSolution()*

Metoda ta użyta jest w obu algorytmach do wyznaczenia rozwiązania początkowego w sposób **zachłanny**.

Algorytm:

1. Utworzenie listy *visited*, zainicjowanej odwiedzeniem korzenia (wierzchołka 0).
2. Rozpoczyna się pętla, która wykonuje się *size-1* razy, ponieważ trasa musi odwiedzić wszystkie wierzchołki oprócz korzenia.
3. Dla aktualnego wierzchołka *curIndex* na trasie, znajdowany jest najbliższy nieodwiedzony wierzchołek *dst* z najmniejszą wagą krawędzi. Wartość minimalnej wagi przechowywana jest w zmiennej *dstMin*.
4. Znaleziony wierzchołek *dst* jest dodany do listy visited i do wynikowej trasy *res* na odpowiedniej pozycji.
5. Po zakończeniu pętli, waga krawędzi powrotnej do korzenia, czyli od ostatnio odwiedzonego wierzchołka do korzenia (wierzchołek 0), jest dodawana do kosztu. Wygenerowana trasa jest zwracana z metody.

## Klasa *Graph* – metoda *solveSimulatedAnnealing()*

Algorytm zaczyna się od inicjalizacji zmiennych, takich jak czas, liczba elementów trasy, i temperatura początkowa – wykorzystana jest w tym celu metoda *generateInitialTemp()*. Następnie początkowa trasa jest generowana w sposób zachłanny i obliczany jest jej koszt. Algorytm przechodzi do głównej pętli, która będzie trwała do momentu spełnienia warunku stopu. W każdej iteracji pętli następują następujące operacje:

1. Losowane są dwie różne pozycje w trasie, a następnie tworzona jest nowa trasa poprzez zastosowanie operatora *insert*.
2. Obliczane są koszty obecnej trasy i nowo wygenerowanej trasy. Następnie obliczana jest różnica kosztów między nimi.
3. Jeśli nowa trasa jest lepsza (o niższym koszcie), to zostaje zaakceptowana jako obecna trasa. W przeciwnym razie, jest ona akceptowana z pewnym prawdopodobieństwem, zależnym od różnicy kosztów i temperatury. Wyraża się ono wzorem Rysunek 2.1, gdzie lewa strona nierówności to losowa liczba z przedziału [0,1), *diff* to różnica kosztów pomiędzy nową a obecną trasą, natomiast *T* to temperatura w obecnej iteracji. Prawdopodobieństwo akceptacji gorszego rozwiązania maleje wraz ze spadkiem temperatury.

Rysunek .

Źródło: opracowanie własne na podstawie[[6]](#footnote-6)

1. Jeśli obecna trasa jest lepsza niż trasa optymalna, to aktualizowana jest trasa optymalna oraz jej koszt.
2. Temperatura jest zmniejszana, co odpowiada procesowi schładzania. W algorytmie symulowanego wyżarzania schładzanie ma na celu zwiększenie prawdopodobieństwa akceptacji lepszych rozwiązań na początku, a następnie stopniowe zaostrzanie kryteriów akceptacji. Schładzanie odbywa się według wzoru Rysunek 2.2. We wzorze tym *T(i)* to temperatura w obecnej iteracji, *α* to współczynnik schładzania – w implementacji określony przez parametr *delta*, natomiast *T(i-1)* to temperatura w poprzedniej iteracji.

Rysunek .

Źródło: opracowanie na podstawie założeń dot. projektu

1. Po każdej 100000. iteracji pętli sprawdzane jest, czy upłynął określony limit czasu. Jeśli tak, to wykonywanie pętli jest przerywane. Sprawdzanie czasu co określoną liczbę iteracji ma na celu ograniczenie wpływu pomiaru czasu na czas wykonywania właściwego algorytmu.

Po zakończeniu głównej pętli algorytm następuje wypisanie czasu wykonania, optymalnego kosztu oraz ścieżki. Zwracany jest koszt trasy. Według przyjętych założeń projektowych główna pętla algorytmu jest wykonywana przez sztywnie narzucony limit czasu określony przez parametr *timeLimit*. Omówiony algorytm przedstawia Rysunek 2.3.

Rysunek . Główna pętla algorytmu symulowanego wyżarzania



Źródło: opracowanie własne

## Klasa *Graph* – metoda *generateInitialTemp()*

Metoda ta użyta jest do generowania temperatury początkowej dla danego problemu w oparciu o przetwarzane dane. W tym celu oblicza się **odchylenie przeciętne**[[7]](#footnote-7) dla długości krawędzi w grafie reprezentującym dany problem. Następuje sumowanie wartości krawędzi dla wszystkich par różnych wierzchołków grafu, pomijając krawędzie pętlowe (krawędzie prowadzące do tego samego wierzchołka). Oblicza się średnią wartości krawędzi jako ilorazu sumy wartości krawędzi przez liczbę wszystkich krawędzi. Szukane odchylenie wyznacza się jako stosunek sumy wartości bezwzględnej różnicy między średnią a wartością krawędzi dla wszystkich par różnych wierzchołków i liczby krawędzi niebędących pętlami. Temperaturę początkową stanowi wartość odchylenia pomnożona przez 105.

## Klasa Graph – metoda *solveTabuSearch()*

Metoda *solveTabuSearch()* użyta jest do rozwiązywania omawianego problemu komiwojażera metodą *Tabu Search*.

Inicjalizowane są zmienne trasy, takie jak trasa bieżąca (*currentRoute*) oraz trasa optymalna (*optimalRoute*), wraz z obliczeniem ich kosztów. Wybierana jest operacja sąsiedztwa: Na podstawie przekazanego znaku (*neighbourFunction*), wybierana jest operacja sąsiedztwa: '*n*' (*insert*), '*i*' (*inverse*), lub domyślnie '*s*' (*swap*). Wybrana operacja przypisywana jest do wskaźnika na metodę ruchu (*movePtr*). Ustalana jest wielkość listy tabu. Rozmiar listy tabu (*tabuListSize*) jest ustawiany na wartość równą liczbie wierzchołków w grafie. Inicjalizowane są zmienne czasowe. Ustalane są zmienne związane z pomiarami czasu. Rozpoczyna się główna pętla algorytmu, która będzie działać do momentu spełnienia warunku stopu – podobnie jak w przypadku metody *SA* jest to przekroczenie limitu czasu *timeLimit*. Iteruje się po sąsiedztwie aktualnej trasy. Dla każdej pary różnych wierzchołków w trasie iteruje się po wszystkich możliwych ruchach sąsiedztwa, wykonując operację wskazywaną przez *movePtr*. Dla każdego ruchu sprawdzane są kryteria: czy ruch jest na liście tabu, czy różnica kosztów jest większa od poprzednio zarejestrowanego najlepszego ruchu, czy też nowy koszt jest lepszy niż dotychczasowy optymalny koszt (kryterium aspiracji). Trasa aktualizowana jest na podstawie najlepszego ruchu, a zakazany ruch dodawany jest do listy tabu. Sprawdzane są kryteria stopu i dywersyfikacji. Dywersyfikacja następuje w przypadku przekroczenia granicy krytycznej – *criticalBound* – równej *105/rozmiar problemu*. Lista tabu jest wtedy czyszczona i generowane jest nowe losowe rozwiązanie. Po każdej iteracji sprawdzane jest kryterium stopu, czyli czy nie został przekroczony limit czasu. Dodatkowo, zwiększany jest licznik krytyczny (*criticalCounter*).

Po zakończeniu pętli, obliczany jest łączny czas wykonania algorytmu. Następnie, zwracany jest optymalny koszt trasy. Wypisywane są wyniki. Omówiony algorytm przedstawia Rysunek 2.4.

Rysunek . Główna pętla algorytmu Tabu Search w postaci pseudokodu



Źródło: opracowanie własne

Istotnym zagadnieniem jest sposób, w jaki **atrybuty** wykonywanych ruchów są dodawane do **listy tabu**. Przyjęto, że dla każdego wykonywanego ruchu *(i, j)* do listy tabu dodawane są pary *(indeks, wierzchołek)*, gdzie *indeks* to odpowiednio *i* oraz *j* , a *wierzchołek* to element pod tym indeksem w wewnętrznej tablicy obiektu trasy.

## Klasa *Route*

Klasa ta jest użyta do reprezentowania ścieżki – trasy komiwojażera bez pierwszego i ostatniego przystanku na trasie, który jest przyjęty jako 0. Oznacza to, że przykładowo ciąg wierzchołków 0-1-2-3-0 jest w tej klasie reprezentowany jako ciąg 1-2-3. Ciąg wierzchołków jest przechowywany w postaci tablicy *std::vector<int> route*.

Zostały zaimplementowane następujące komponenty klasy:

* Konstruktor *Route(int n)* - tworzy obiekt trasy o rozmiarze *n*. Inicjalizuje wektor *route* o zadanej wielkości,
* Metoda *randomize()* generuje losową permutację trasy, reprezentującą trasę komiwojażera bez pierwszego i ostatniego przystanku na trasie (0),
* Metoda *toString()* zwraca tekstową reprezentację trasy, gdzie kolejne liczby są oddzielone spacją.
* Operator przypisania *operator=* przypisuje zawartość jednej trasy do drugiej.
* Operator dostępu do elementu *operator[]* umożliwia odczyt i modyfikację elementów trasy.
* Operator porównania *operator==* porównuje dwie trasy i zwraca *true*, jeśli są identyczne.
* Metoda *procedureSwap()* wykonuje operację swap (zamiana miejscami) dla dwóch wierzchołków na trasie.
* Metoda *procedureInverse()* wykonuje operację odwracania kolejności elementów trasy między dwoma wskazanymi indeksami.
* Metoda *procedureInsert()* wykonuje operację wstawiania elementu na trasie między dwoma wskazanymi indeksami.
* Metoda *getSize()* zwraca rozmiar trasy.
* Pomocnicza metoda *swap()* zamienia miejscami dwa elementy trasy na podstawie ich indeksów.

Implementację operatorów sąsiedztwa przedstawia Rysunek 2.5.



Źródło: opracowanie własne

Rysunek . Implementacje operatorów sąsiedztwa i metody pomocniczej swap()

# Sposób przeprowadzenia badania

W celu realizacji badania – eksperymentu zostały wykorzystane omówione wcześniej klasy. Podobnie jak w przypadku realizacji wcześniejszych projektów, w celu oceny efektywności badanych algorytmów został wykorzystany pomiar czasu przy wykorzystaniu funkcjonalności biblioteki *<chrono>* i zawartej w niej klasy *steady\_clock*, która reprezentuje zegar monotoniczny, dla którego gwarantowane jest, że różnica czasu (przykładowo przekonwertowanego do milisekund – tak jak w programie) będzie większa od zera dla dwóch momentów czasu badania, z których drugi występuje później niż pierwszy. Po wykonaniu każdej z metod realizujących badane algorytmy wypisywany jest czas uzyskania wyniku w milisekundach, koszt ścieżki, ścieżka – bez początkowego i ostatniego wierzchołka na trasie (0) oraz, dla algorytmu *SA*, temperatura końcowa.

Do badań zostały wykorzystane problemy wskazane w wytycznych dot. projektu. Są one reprezentowane przez pliki:

* *ftv47.atsp* (1776),
* *ftv170.atsp* (2755) ,
* *rgb403.atsp* (2465).

W nawiasach zostały podane najlepsze znane rozwiązania – koszty tras – dla tych problemów[[8]](#footnote-8).

Przeanalizowano rezultaty działania algorytmów TS iSA dla podanych trzech plików przy wykorzystaniu podanych parametrów:

* *Tabu Search* – sąsiedztwo typu *swap*,
* *Simulated Annealing* –współczynnik schładzania *α* = 0,9999995 i sąsiedztwo typu *insert*.

Ustalono limity czasu wykonywania algorytmów na: 2 min dla problemu *ftv47.atsp*, 4 min dla problemu *ftv170.atsp* oraz 6 min dla problemu *rgb403.atsp*.

# Wyniki przeprowadzonego badania

Rezultaty przeprowadzonego badania zostały podsumowane w tabelach oraz na wykresie.

## Wszystkie wyniki

Tabela . Plik ftv47.atsp – algorytm Tabu Search

|  |
| --- |
| **ftv47.atsp TS** |
| **uruchomienie** | **czas [ms]** | **dł. ścieżki** | **błąd %** |
| **1** | 6.13E+04 | 2018 | 14% |
| **2** | 4.63E+04 | 2011 | 13% |
| **3** | 5.60E+04 | 1949 | 10% |
| **4** | 4.00E+04 | 1998 | 13% |
| **5** | 9.21E+04 | 1949 | 10% |
| **6** | 9.40E+04 | 2021 | 14% |
| **7** | 3.01E+04 | 2063 | 16% |
| **8** | 7.52E+03 | 2060 | 16% |
| **9** | 9.16E+04 | 2043 | 15% |
| **10** | 1.03E+05 | 2082 | 17% |
| **średnia** | **6.22E+04** | **2019** | **14%** |
| **mediana** | **5.86E+04** | **2020** | **14%** |

Źródło: opracowanie własne

Tabela . Plik ftv170.atsp – algorytm Tabu Search

|  |
| --- |
| **ftv170.atsp TS** |
| **uruchomienie** | **czas [ms]** | **dł. ścieżki** | **błąd %** |
| **1** | 2.87E+03 | 3632 | 32% |
| **2** | 2.87E+03 | 3632 | 32% |
| **3** | 2.80E+03 | 3632 | 32% |
| **4** | 2.82E+03 | 3632 | 32% |
| **5** | 2.81E+03 | 3632 | 32% |
| **6** | 2.92E+03 | 3632 | 32% |
| **7** | 2.91E+03 | 3632 | 32% |
| **8** | 2.86E+03 | 3632 | 32% |
| **9** | 2.81E+03 | 3632 | 32% |
| **10** | 2.89E+03 | 3632 | 32% |
| **średnia** | **2.86E+03** | **3632** | **32%** |
| **mediana** | **2.86E+03** | **3632** | **32%** |

Źródło: opracowanie własne

Tabela . Plik rbg403.atsp – algorytm Tabu Search

|  |
| --- |
| **rbg403.atsp TS** |
| **uruchomienie** | **czas [ms]** | **dł. ścieżki** | **błąd %** |
| **1** | 2.57E+05 | 2597 | 5% |
| **2** | 2.56E+05 | 2597 | 5% |
| **3** | 2.56E+05 | 2597 | 5% |
| **4** | 2.65E+05 | 2597 | 5% |
| **5** | 2.78E+05 | 2597 | 5% |
| **6** | 2.62E+05 | 2597 | 5% |
| **7** | 2.61E+05 | 2597 | 5% |
| **8** | 2.71E+05 | 2597 | 5% |
| **9** | 2.68E+05 | 2597 | 5% |
| **10** | 2.60E+05 | 2597 | 5% |
| **średnia** | **2.63E+05** | **2597** | **5%** |
| **mediana** | **2.62E+05** | **2597** | **5%** |

Źródło: opracowanie własne

Tabela . ftv47.atsp – algorytm Symulowanego Wyżarzania

|  |
| --- |
| **ftv47.atsp SA** |
| **uruchomienie** | **czas [ms]** | **dł. ścieżki** | **błąd %** | **temp. końcowa** |
| **1** | 4.21E+04 | 1935 | 9% | 8.46E-12 |
| **2** | 4.06E+04 | 1911 | 8% | 2.09E-12 |
| **3** | 9.40E+04 | 1987 | 12% | 3.44E-12 |
| **4** | 1.11E+05 | 2001 | 13% | 2.19E-12 |
| **5** | 3.85E+04 | 1978 | 11% | 1.55E-12 |
| **6** | 3.79E+04 | 1980 | 11% | 2.42E-12 |
| **7** | 8.50E+04 | 1951 | 10% | 1.20E-12 |
| **8** | 3.99E+04 | 1959 | 10% | 1.04E-12 |
| **9** | 3.89E+04 | 2013 | 13% | 2.09E-12 |
| **10** | 3.81E+04 | 1962 | 10% | 1.89E-12 |
| **średnia** | **5.66E+04** | **1968** | **11%** | **2.64E-12** |
| **mediana** | **4.03E+04** | **1970** | **11%** | **2.08E-12** |

Źródło: opracowanie własne

Tabela . Plik ftv170.atsp - algorytm Symulowanego Wyżarzania

|  |
| --- |
| **ftv170.atsp SA** |
| **uruchomienie** | **czas [ms]** | **dł. ścieżki** | **błąd %** | **temp. końcowa** |
| **1** | 6.18E+00 | 3923 | 42% | 3.88E-05 |
| **2** | 6.20E+00 | 3923 | 42% | 2.02E-05 |
| **3** | 6.25E+00 | 3923 | 42% | 6.74E-06 |
| **4** | 6.28E+00 | 3923 | 42% | 4.98E-05 |
| **5** | 6.92E+00 | 3923 | 42% | 2.24E-05 |
| **6** | 6.20E+00 | 3923 | 42% | 1.93E-05 |
| **7** | 6.49E+00 | 3923 | 42% | 6.08E-05 |
| **8** | 6.26E+00 | 3923 | 42% | 5.23E-05 |
| **9** | 6.41E+00 | 3923 | 42% | 6.08E-05 |
| **10** | 6.29E+00 | 3923 | 42% | 7.81E-05 |
| **średnia** | **6.35E+00** | **3923** | **42%** | **4.09E-05** |
| **mediana** | **6.27E+00** | **3923** | **42%** | **4.43E-05** |

Źródło: opracowanie własne

Tabela . Plik rbg403.atsp - algorytm Symulowanego Wyżarzania

|  |
| --- |
| **rbg403.atsp SA** |
| **uruchomienie** | **czas [ms]** | **dł. ścieżki** | **błąd %** | **temp. końcowa** |
| **1** | 3.29E+05 | 2739 | 11% | 9.79E-02 |
| **2** | 3.37E+05 | 2702 | 10% | 8.86E-02 |
| **3** | 3.24E+05 | 2702 | 10% | 8.43E-02 |
| **4** | 3.17E+05 | 2700 | 10% | 1.56E-01 |
| **5** | 3.43E+05 | 2775 | 13% | 8.43E-02 |
| **6** | 3.36E+05 | 2739 | 11% | 7.63E-02 |
| **7** | 3.26E+05 | 2731 | 11% | 9.79E-02 |
| **8** | 3.18E+05 | 2728 | 11% | 1.20E-01 |
| **9** | 3.42E+05 | 2700 | 10% | 1.39E-01 |
| **10** | 3.20E+05 | 2681 | 9% | 1.20E-01 |
| **średnia** | **3.29E+05** | **2720** | **10%** | **1.06E-01** |
| **mediana** | **3.27E+05** | **2715** | **10%** | **9.79E-02** |

Źródło: opracowanie własne

Wykres .

Źródło: opracowanie własne

## Najlepsze rozwiązania

Plik *ftv47.atsp* – koszt 1911 – algorytm SA – ścieżka:

Rysunek .

*0 25 1 9 33 27 28 2 41 43 47 26 42 22 40 20 38 18 17 46 36 35 14 23 34 13 12 32 7 31 30 5 24 4 29 3 6 10 8 11 37 19 44 15 16 45 39 21 0*

Źródło: opracowanie własne

Plik *ftv170.atsp* – koszt 3632 – algorytm TS – ścieżka:

Rysunek .

*0 1 2 77 73 170 49 50 51 52 53 43 55 54 58 59 60 61 68 67 167 70 87 85 86 83 84 69 66 63 64 56 57 62 65 88 153 154 89 90 91 94 96 97 99 98 95 92 93 166 108 107 106 105 165 163 101 100 102 117 118 119 120 121 122 123 162 103 104 114 109 113 164 127 126 125 124 129 128 130 131 132 112 133 134 6 7 8 9 10 76 74 75 11 12 18 19 20 21 22 23 26 27 28 29 30 31 33 34 156 40 39 38 37 35 36 157 41 155 42 45 44 46 47 48 168 72 78 82 79 80 81 3 4 5 169 111 110 71 158 32 16 17 24 25 150 161 160 151 152 144 143 142 149 148 147 137 136 138 135 139 140 14 15 159 13 115 116 146 145 141 0*

Źródło: opracowanie własne

Plik *rbg403.atsp* – koszt 2597 – algorytm TS – ścieżka:

Rysunek .

*0 314 59 23 14 62 13 205 379 248 348 241 113 395 33 376 88 38 270 19 377 402 287 107 61 370 104 145 304 260 308 394 225 58 8 6 164 3 2 386 47 112 322 272 28 151 11 152 76 340 367 310 29 397 5 56 65 387 84 46 77 31 52 40 39 78 226 147 360 69 245 81 22 21 82 247 232 172 26 182 131 167 64 15 169 27 96 66 60 44 221 67 94 79 353 263 90 72 57 163 25 246 92 51 49 37 36 108 120 392 351 83 218 349 206 30 133 339 364 303 18 253 301 256 318 99 389 333 267 373 355 115 114 393 312 35 209 41 160 365 68 180 165 202 122 119 168 140 372 356 328 257 217 9 384 383 361 146 144 105 391 154 111 369 103 374 363 289 129 255 110 109 275 210 265 326 258 106 359 214 192 189 162 48 358 281 124 284 290 125 74 216 200 71 95 55 305 215 309 252 191 187 234 228 198 264 213 278 224 316 285 282 293 262 261 85 132 243 175 291 204 161 148 325 134 306 319 135 280 12 42 136 137 54 336 292 329 294 227 381 345 173 344 177 158 269 91 239 70 149 347 401 354 342 337 93 203 295 190 138 266 86 101 332 296 128 150 176 156 390 153 75 10 331 378 297 233 380 315 157 1 171 63 251 330 343 159 323 188 299 259 130 141 116 184 4 43 34 385 199 307 143 352 195 155 7 341 102 166 300 335 170 334 126 174 185 178 100 98 179 368 181 346 183 17 186 53 400 50 236 212 254 223 73 193 207 45 338 231 288 249 220 197 142 222 250 211 320 302 89 235 121 24 238 229 240 219 237 244 80 388 279 123 399 398 268 16 32 274 283 277 286 273 317 208 313 20 311 271 327 366 321 362 127 396 357 242 117 97 276 298 201 118 139 87 194 324 375 371 350 230 382 196 0*

Źródło: opracowanie własne

# Wnioski

Wnioskuje się, że algorytmy *tabu search* oraz symulowanego wyżarzania są w stanie wygenerować rozwiązania dla znacznie większych problemów niż algorytmy dokładne (w tym omówione w trakcie poprzednich projektów algorytmy *brute force* i *branch-and-bound*) w tym samym czasie, jednak, zgodnie z przedstawionymi we wstępie teoretycznym informacjami, nie muszą one gwarantować znalezienia globalnego optimum dla danego problemu – uzyskane wyniki różniły się od deklarowanych optimów (str. 10). Są to algorytmy, których implementacja jest złożona ze względu na różne możliwości realizacji ich poszczególnych fragmentów. Istotny jest dobór parametrów – w tym definicji sąsiedztwa, współczynnika schładzania dla algorytmu *SA* oraz sposób dywersyfikacji oraz realizacji listy tabu dla algorytmu *TS*. Przypuszcza się, że ich lepszy dobór umożliwiłby uzyskanie wyników jeszcze bardziej zbliżonych do najlepszych znanych rozwiązań dla poszczególnych plików. Zauważono jednocześnie, że, dla danego algorytmu, błąd oraz czas uzyskiwanego rozwiązania nie był proporcjonalny do rozmiaru problemu (Wykres 4.1), na co wpływ mogły mieć wykorzystane parametry i „stopień trudności” każdego z problemów - dla problemu *ftv170.atsp* najlepsze rozwiązanie znajdowane było tuż po uruchomieniu algorytmu  (Tabela 4.2 i Tabela 4.5).

# Bibliografia

1. Robert Wieczorkowski: „Algorytmy stochastyczne w optymalizacji dyskretnej przy zaburzonych wartościach funkcji”
2. Algorytmy heurystyczne, UWr: <https://ii.uni.wroc.pl/~prz/2009lato/ah/ah.html>

1. <https://www.cs.cmu.edu/~15281/coursenotes/localsearch/index.html> [↑](#footnote-ref-1)
2. <http://www.cs.put.poznan.pl/mkasprzak/zp/ZP-wyklad3.pdf> [↑](#footnote-ref-2)
3. <https://cs.pwr.edu.pl/zielinski/lectures/om/localsearch.pdf> [↑](#footnote-ref-3)
4. <http://www.pi.zarz.agh.edu.pl/intObl/notes/IntObl_w2.pdf> [↑](#footnote-ref-4)
5. [https://www.researchgate.net/publication/266033160\_13091453v1#pf11](https://www.researchgate.net/publication/266033160_13091453v1%23pf11) [↑](#footnote-ref-5)
6. <https://cs.pwr.edu.pl/zielinski/lectures/om/localsearch.pdf> [↑](#footnote-ref-6)
7. <http://home.agh.edu.pl/~bartus/index.php?action=dydaktyka&subaction=statystyka&item=miary_zmiennosci> [↑](#footnote-ref-7)
8. <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ATSP.html> [↑](#footnote-ref-8)