# Sprawozdanie
# Projektowanie Efektywnych Algorytmów

Temat:
Implementacja i analiza efektywności algorytmu Tabu Search i Symulowanego Wyżarzania dla problemu komiwojażera

Politechnika Wrocławska
**Dawid Szeląg 264008**
Prowadzący: dr. inż. Jarosław Mierzwa
07.12.2023r.

Politechnika
Wrocławska

# Spis treści

# 1 Metaheurystyki

## 1.1 Tabu Search

Algorytm Tabu Search, należy do rodziny algorytmów **local search**. Metoda polega na przeszukiwaniu z zakazami (jak sama nazwa zresztą wskazuje). Przeszukujemy sąsiedztwo aktualnie posiadanego rozwiązania, a następnie wybieramy najbardziej obiecującego sąsiada. Ruch, który doprawdza nas do wspomnianego sąsiada, wpisujemy na liste Tabu (blokujemy na określoną kadencje),a czynność powtarzamy, aż do warunku zatrzymania.
Jako, że **tabu search** jest metodą heurystyczną, oznacza to, że algorytm nie gwarantuje znalezienia optymalnego rozwiązania. Procent błędu w stosunku do optimum, jest większy/mniejszy w zależności od:

- rodzaju problemu,
- wielkości problemu (np. ilości miast),
- implementacji dodatkowych funkcji w algorytmie,
- znajomości na temat rozważanego problemu,
- czasu wykonywania,
- długości kadencji (ilość iteracji, która opisuje na ile danych ruch został zablokowany),
- rodzaju sąsiedztwa, etc.

W rozważanym problemie ATSP, wykorzystane zostaną 3 typy sąsiedztwa:

- swap (bierzemy dwa elementy xi i xj, zamieniamy je ze sobą miejscami),
- insert (bierzemy dwa elementy xi i xj, xj wstawiamy przed xi),
- invert (bierzemy dwa elementy xi i xj, między xi i xj włącznie zamieniamy wszystkie elementy miejscami, pierwszy element idzie na miejsce ostatniego, ostatni na miejsce pierwszego,. drugi na miejsce przedostatniego, przedostatni na miejsce drugiego itd.)

Lista tabu została zrealizowana jako tablica dwuwymiarowa, gdzie w komórkach przechowujemy aktualne kadencje ruchów, np. table[xi][xj] = 4, oznacza, że ruch xi <->xj, jest zablokowany na 4 następne iteracje.
Dywersyfikacja (przeszukiwanie większej części rozwiązań), zostało zrealizowane za pomocą **Critical Event** oraz poprzez rozmiar kadencji.

Liczenie kosztu ścieżki podczas przeszukiwania sąsiedztwa, zaimplementowano w sposób taki, że nie wykonujemy operacji swap/insert/invert na aktualnej ścieżce, lecz obliczamy ile wynosiłby koszt ścieżki po takiej zmianie, a dopiero po wybraniu najlepszej możliwej opcji, wykonujemy operacje swap/insert/invert na ścieżce.
Kadencja jest liczona na podstawie wzoru: $kadencja = k \cdot \sqrt{N}$ , gdzie k jest parametrem, a N oznacza liczbe miast

Implementacja **tabu search**, została zrealizowana w następujący sposób:

1. Wybieramy drogę początkową za pomocą metody zachłannej oraz wyliczamy jej koszt.

2. Tworzymy pustą listę tabu (w tym przypadku tablice dwuwymiarową z wartościami = 0).

3. Sprawdzamy czy ilość iteracji bez zmian aktualnego optimum, przekroczyła dopuszczalną wartość (**Critical Event**). Jeśli tak, to wybieramy losowo, którą połówkę aktualnej ścieżki należy przetasować (losowo pozamieniać miasta w danej połówce) oraz wyczyścić listę tabu. W przeciwnym razie kontynuujemy algorytm.

4. Przeszukujemy wszystkich sąsiadów typu swap lub insert lub invert aktualnego rozwiązania oraz wybieramy najlepszego, dla którego ruch nie znajduje się na liście tabu **LUB** dla którego ruch znajduje się na liście tabu, lecz znaleziony sąsiad, jest lepszy o zadane **kryterium aspiracji** od najlepszego dotychczasowego rozwiązania.

5. Po znalezieniu sąsiada, redukujemy listę tabu o 1 (wszystkie wartości w tablicy zmniejszamy o 1, oprócz tych które wynoszą 0).

6. Ruch, który doprowadził do wybranego sąsiada, dopisujemy do listy tabu (blokujemy na zadaną kadencję).

7. Aktualizujemy aktualną ścieżkę według wybranego sąsiada oraz jeśli aktualna ścieżka jest lepsza od aktualnego optimum, należy nadpisać aktualne optimum.

8. Powtarzamy kroki **3-7**, dopóki nie minie zadany czas (**warunek zakończenia**).

## 1.2 Symulowane wyżarzanie

Nazwa metody jest odniesieniem do zjawiska stygnięcia metali. Jeżeli spadek temperatury jest wystarczająco wolny to cząsteczki tworzą równomierną strukturę. W termodynamice do opisania tego zjawiska wykorzystywany jest następujący wzór: $P(E) = e^{-\frac{E}{kT}}$.
Podany wcześniej wzór (w lekko zmienionej formie), zostanie wykorzystane do obliczenia prawodopodobieństwa, które będzie potrzebne do algorytmu

Algorytm symulowanego wyżarzania jest oparty na metodzie iteracyjnej. Metoda ta zakłada, że ciągle ulepszamy istniejące rozwiązanie, aż w końcu nie będzie można go dalej poprawić. Przejście z jednego rozwiązania do drugiego polega na znalezieniu lepszego rozwiązania sąsiedniego, czyli takiego, który znajduje się w sąsiedztwie wcześniejszego. W algorytmie symulowanego wyżarzania istnieje jednak możliwość wyboru gorszego rozwiązania z pewnym prawdopodobieństwem. Szansa na wybór gorszego rozwiązania jest zależna od temperatury, która maleje wraz z czasem wykonywania programu. Taki mechanizm sprawia, że podczas wykonywania algorytmu będzie możliwość „wyjścia" z minimum lokalnego, czyli zastąpienie minima lokalnego rozwiązaniem gorszym.

Jako, że symulowane wyżarzanie jest metodą heurystyczną (tak samo jak tabu search), oznacza to, że algorytm nie gwarantuje znalezienia optymalnego rozwiązania. Procent błędu w stosunku do optimum, jest większy/mniejszy w zależności od:

- temperatury początkowej,
- wielkości epoki,
- schematu chłodzenia,
- czasu wykonywania, etc.

Do implementacji SA, użyto przeglądania sąsiedztwa typu **insert** oraz dodano optymalizację, która polega na przejściu przez wszystkie możliwe rozwiązania w sąsiedztwie 2-opt podczaj 1 epoki, a następniu wybraniu najlepszej permutacji 2 liczb, która dopiero później trafia do wzoru z prawdopodbieństwem.
Parametery potrzebne do algorytmu, ustalane są w następujący sposób:

- temperatura początkowa = koszt początkowej ścieżki $\cdot$ 10
- wielkość epoki = 500

- schemat schładzania: T+1 = a · T, gdzie a przyjmuje 3 możliwe wartości **0.99, 0.999, 0.9999**

Symulowane wyżarzanie przebiega następująco:

1. Ustalamy temperature początkową oraz wielkość epoki za pomocą określonego wcześniej wzoru.

2. Przechodzę do epoki.

3. Generuje wszystkie możliwe permutacje 2 liczb oraz dla każdej z nich przeglądam sąsiedztwo typu insert. Zapamiętuje najlepsze 2 liczby, dla których sąsiedztwo typu insert jest najlepsze oraz różnice pomiędzy aktualną ścieżką.

4. Jeśli wybrane najlepsze sąsiedztwo jest lepsze niż aktualne rozwiązanie, zamieniam aktualne rozwiązanie na owe sąsiedztwo. Jeśli nie, korzystam z wzoru: $(exp(\frac{-(cost(x')-cost(x))}{temperatura})$, gdzie x' - nowe możliwe rozwiązanie, x - aktualne rozwiązanie), który wylicza nam prawdopodobieństwo zmiany na gorsze rozwiązanie. Obliczone prawdopodobieństwo porównuje z wygenerowaną losowo liczba z przedziału [0.0] - [1.0]. Jeśli liczba jest mniejsza od prawdopodobieństwa, zamieniam rozwiązanie pomimo tego że jest gorsze. W przeciwnym wypadku nic nie robię.

5. Sprawdzam dodatkowo, czy aktualne rozwiązanie nie jest lepsze on najlepszego aktualnego optimum. Jeśli jest, to zamieniam optimum.

6. Wykonuje punkty 2 - 5, dopóki nie wykonam tyle iteracji, ile wynosi wielkość epoki.

7. Po przejściu epoki, obniżam temperature, za pomocą wspomnianego wcześniej wzoru.

8. Sprawdzam czy podczas epoki, zaszła jakaś zmiana najlepszego aktualnego optimum. Jeśli nie, to wybieramy losowo, którą połówkę aktualnej ścieżki należy przetasować (losowo pozamieniać miasta w danej połówce).

9. Powtarzam punkty 2-8, dopóki nie minie zadany czas (**warunek zatrzymania)**

# 2 Klasy w projekcie

## 2.1 TabuSearch

Klasa, która ma w sobie wszytkie rzeczy związane z metodą Tabu search.

Zmienne/kolekcje wykorzystane w klasie:

```
AdjacencyMatrix* matrix;
std::vector<int\textgreater shortestPath;
//przchowuje zmiany sciezki na lepsze (cost,time)
std::vector<std::vector<double\textgreater\textgreater
    shortestPathChanging;
int shortestPathValue;
int greedyPathValue;
double timeStop;
int term;
double timeWhenBest;
int iterationsWithNoChangeMAX;
int iterationsWithNoChange;
///1 - SWAP ; 2 - INSERT ; 3 - INVERT
int neighborhoodType;
//roznica pomiedzy znalezionym rozwiazaniem, a najlepszym,
    óktra pozwala na zignorowanie tabu
int aspirationPlus;
//znane optymalne rozwiazanie, do porownania wynikow
int bestKnownResult;
```

Najważniejsze metody:

```
//liczenie ąpocztkowej sciezki metoda łzachanna
void greedyRoute();
//przelosowanie po CriticalEvent
void randomRoute(std::vector<int\textgreater &source);
void start();
//liczenie sciezki po mozliwej zmianie w sasiedztwie
int countPath(int tempValue, int I, int J,
    std::vector<int\textgreater &source);
```

, gdzie **int countPath(...)**:

```
int TabuSearch::countPath(int tempValue, int elementI, int
    elementJ, std::vector<int\textgreater &source) {
int minus = 0, plus = 0;
```

```
switch (neighborhoodType) {
    case 1: /// SWAP
        if (abs(elementI - elementJ) \textgreater 1) {
            //zmniejsz dotychczasowa wartosc sciezki o krawedzie, z
                ktorymi uczestnicza zamieniane wierzcholki (PO
                MOZLIWEJ ZAMIANIE)
            minus = (matrix-\textgreatergetWsk()[source[elementI -
                1]][source[elementI]] +
                     matrix-\textgreatergetWsk()[source[elementI]][source[elementI
                         + 1]]
                     + matrix-\textgreatergetWsk()[source[elementJ -
                         1]][source[elementJ]] +
                     matrix-\textgreatergetWsk()[source[elementJ]][source[elementJ
                         + 1]]);

            //powieksz dotychczasowa wartosc sciezki o krawedzie, z
                ktorymi uczestnicza zamieniane wierzcholki (PO
                MOZLIWEJ ZAMIANIE)
            plus = (matrix-\textgreatergetWsk()[source[elementI -
                1]][source[elementJ]] +
                    matrix-\textgreatergetWsk()[source[elementJ]][source[elementI
                        + 1]]
                    + matrix-\textgreatergetWsk()[source[elementJ -
                        1]][source[elementI]] +
                    matrix-\textgreatergetWsk()[source[elementI]][source[elementJ
                        + 1]]);
        } else
        {
            //zmniejsz dotychczasowa wartosc sciezki o krawedzie, z
                ktorymi uczestnicza zamieniane wierzcholki (PO
                MOZLIWEJ ZAMIANIE)
            minus = (matrix-\textgreatergetWsk()[source[elementI -
                1]][source[elementI]] +
                     matrix-\textgreatergetWsk()[source[elementI]][source[elementI
                         + 1]] +
                     matrix-\textgreatergetWsk()[source[elementJ]][source[elementJ
                         + 1]]);

            //powieksz dotychczasowa wartosc sciezki o krawedzie, z
                ktorymi uczestnicza zamieniane wierzcholki (PO
                MOZLIWEJ ZAMIANIE)
            plus = (matrix-\textgreatergetWsk()[source[elementI -
```

```
            1]][source[elementJ]] +
                matrix-\textgreatergetWsk()[source[elementJ]][source[elementI]]
                    +
                matrix-\textgreatergetWsk()[source[elementI]][source[elementJ
                    + 1]]);
    }
    break;
case 2: /// INSERT
        minus = (matrix-\textgreatergetWsk()[source[elementI -
            1]][source[elementI]] +
                 matrix-\textgreatergetWsk()[source[elementJ-1]][source[elementJ]]
                 +
                     matrix-\textgreatergetWsk()[source[elementJ]][source[elementJ
                     + 1]]);

        plus = (matrix-\textgreatergetWsk()[source[elementI -
            1]][source[elementJ]] +
                matrix-\textgreatergetWsk()[source[elementJ]][source[elementI]]
                + matrix-\textgreatergetWsk()[source[elementJ -
                    1]][source[elementJ + 1]]);
    break;
case 3: ///INVERT
    int max = abs(elementJ - elementI);

    plus += matrix-\textgreatergetWsk()[source[elementI -
        1]][source[elementJ]];
    minus += matrix-\textgreatergetWsk()[source[elementI -
        1]][source[elementI]];

    for (int i = 0; i < max; ++i) {
        minus += matrix-\textgreatergetWsk()[source[elementI +
            i]][source[elementI + 1 + i]];
        plus += matrix-\textgreatergetWsk()[source[elementJ -
            i]][source[elementJ - 1 - i]];
    }

    minus +=
        matrix-\textgreatergetWsk()[source[elementJ]][source[elementJ
        + 1]];
    plus +=
        matrix-\textgreatergetWsk()[source[elementI]][source[elementJ
        + 1]];
```

```
        break;
}

return tempValue - minus + plus;
}
```

## 2.2 SimulatedAnnealing

Klasa, która ma w sobie wszytkie rzeczy związane z metodą symulowanego wyżarzania.

Zmienne/kolekcje wykorzystane w klasie:

```
AdjacencyMatrix* matrix;
std::vector<int> shortestPath;
std::vector<std::vector<double>> shortestPathChanging;
int shortestPathValue;
int greedyPathValue;
double timeStop;
double timeWhenBest;
int neighborhoodType;  ///1 - SWAP ; 2 - INSERT ; 3 - INVERT
int bestKnownResult;  //znane optymalne rozwiazanie, do
    porownania wynikow
int iterationsWithChange;
double temperature;
double temperatureStart;
int L;                //dlugosc epoki
double a;             //wspolczynnik schladzanie (Temp + 1 =
    a*temp), gdzie a < 1
```

Metody pokrywają się w większej części z metodami z klasy TabuSearch.

# 3 Plan eksperymentu

Program został napisany w języku C++ w środowisku CLion i testowany byl w wersji Release.
Grafy wejściowe, na których należało przeprowadzić badania to pliki:

- ftv55.atsp (optymalny znany koszt: 1608)
- ftv170.atsp (optymalny znany koszt: 2755)
- rbg358.atsp (optymalny znany koszt: 1163)

Pomiar czasu został przeprowadzony przy pomocy wykorzystania zegara z biblioteki **<chrono>**: std::chrono::high_resolution_clock, a generowanie liczb, zostało zrealizowane za pomocą generatora liczb mt19937 z biblioteki **<random>**.

Dla każdego pliku wykonano 10 uruchomień z ustalonymi wcześniej parametrami startowymi oraz dodatkowo wykonano pomiary dla różnego typu sąsiedztw.

## 3.1 Wyniki dla Tabu Search

Dla plików przyjęto następujące parametry:

- ftv55.atsp:
  - sąsiedztwo typu insert
  - maksymalna ilość iteracji bez zmian = 150
  - k = 0.65
  - aspiracja = 0
- ftv170.atsp:
  - sąsiedztwo typu insert
  - maksymalna ilość iteracji bez zmian = 250
  - k = 3.5
  - aspiracja = 50
- rbg358.atsp:
  - sąsiedztwo typu insert

– maksymalna ilość iteracji bez zmian = 350
– k = 3
– aspiracja = 50

| **Plik** | **Znane optimum** | **Znalezione rozwiązanie** | **Błąd względny** | **Czas [s]** | **Czas znalezienia rozwiązania [s]** |
|---|---|---|---|---|---|
| *ftv55.atsp* | 1608 | 1644 | 2.23% | 120 | 15.541 |
| *ftv170.atsp* | 2755 | 3350 | 21.59% | 240 | 156.948 |
| *rbg358.atsp* | 1163 | 1338 | 15.04% | 360 | 269.408 |

Tabela 1: Najlepsze rozwiązania - tabu search

Ścieżki dla przedstawionych rozwiązań są następujące:

**ftv55.atsp:**
0-33-55-34-1-2-13-35-4-6-5-47-31-48-3-7-32-8-36-9-37-11-19-18-39-38-10-51-14-12-15-16-17-52-26-25-24-20-40-42-21-22-41-50-23-54-27-49-45-44-28-53-43-30-46-29-0

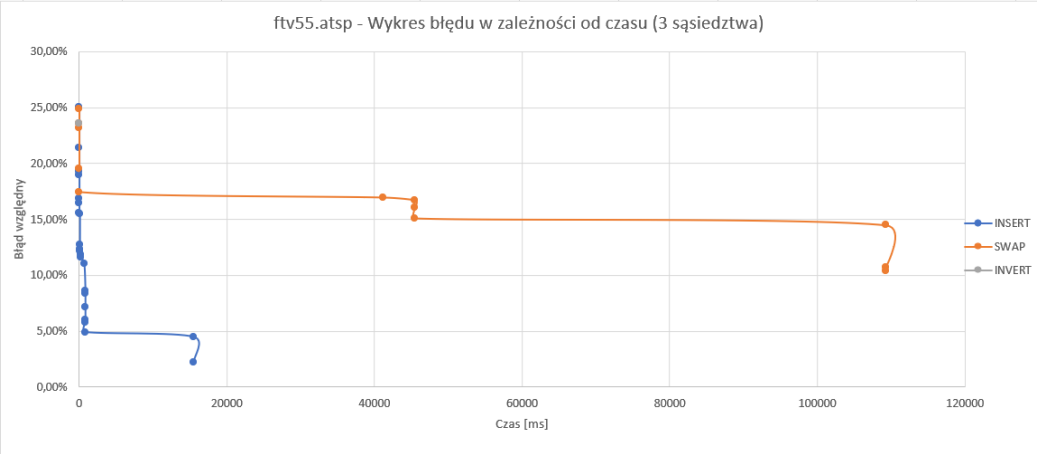
**ftv170.atsp:**
0-1-2-81-80-79-82-78-77-111-112-132-133-169-3-4-9-10-76-74-75-11-12-13-17-18-19-37-20-21-29-22-23-26-27-28-30-31-158-32-36-157-33-35-34-156-40-38-39-155-41-42-45-44-46-47-48 -49-170-168-72-71-60-50-51-52-53-43-55-54-58-59-73-5-134-141-6-7-8-14-15-159-16-24-25-150-148-149-161-160-151-152-142-131-113-164-130-135-136-137-138-139-140-110-109-108-166-93- 107-106-105-97-99-98-95-94-96-165-104-114-127-126-125-124-129-146-145-144-143-147-128-115-116-117-118-119-120-121-122-162-123-101-100-102-103-163-92-91-154-90-89-88-153-87-70-69-67-66-63-64-65-56-57-62-61-68-167-85-86-83-84-0

**rbg358.atsp:**
0-69-39-249-287-236-357-96-117-333-273-232-135-102-272-76-1-125-321-356-170-166-157-30-20-298-248-222-114-63-50-206-192-247-339-230-71-53-93-26-112-103-89-276-191-60-189-204-28-251 -163-7-284-145-15-246-6-81-56-23-75-54-185-307-323-302-200-45-25-12-68-124-148-51-78-97-208-347-291-90-305-91-86-87-83-134-82-214-70-342-120-266-278-325-350-300-80-322-296-101-22 1-244-211-207-17-285-239-21-10-27-168-219-196-35-42-198-240-182-279-250-215-308-212-174-8-100-286-22-233-19-88-58-257-156-295-193-77-18-315-64-158-344-288-153-85-317-130-349-331-22 8-327-338-61-154-353-313-194-131-311-116-252-335-147-138-113-84-32-31-263-282-274-92-275-33-164-218-108-290-136-111-195-217-201-169-348-292-140-330-261-151-143-161-152-270-329-79-142 -352-259-260-183-184-319-243-159-3-312-216-177-150-141-139-271-310-59-121-13-345-289-162-133-337-258-132-171-

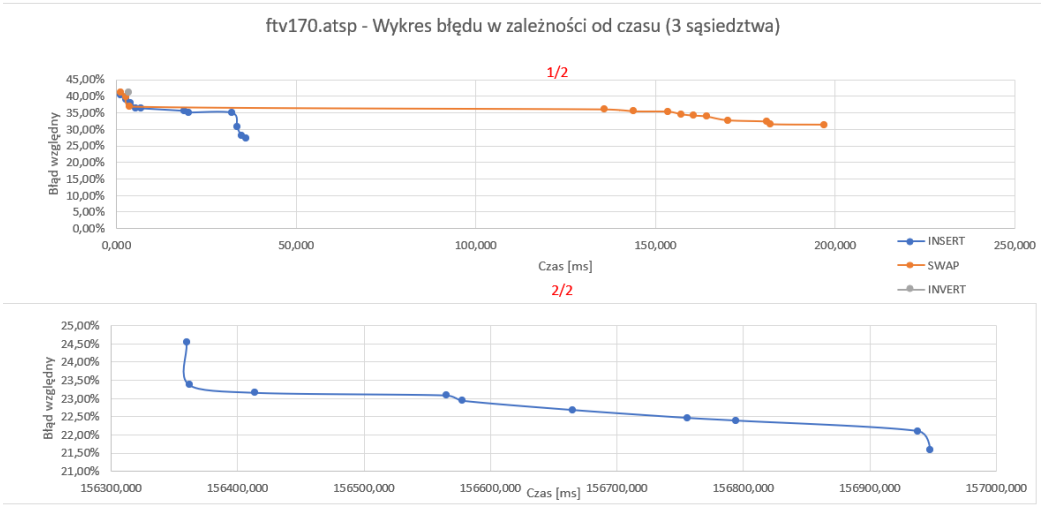
334-48-73-62-66-14-4-44-137-254-318-306-351-197-52-104-106-262-332-40-3 6-29-299-187-205-226-283-118-109-277-336-105-227-65-16-37-242-181-316-224-129-57-256-160-269-220-46-149-210-265-180-165-346-245-122-199-209-49-5-175-172-253-178-176-110-99-186-107-9 4-72-115-223-264-341-280-127-98-179-173-126-267-326-213-234-303-301-343-155-67-55-229-309-314-203-74-146-255-320-95-237-355-268-119-188-41-294-167-2-43-38-235-11-225-293-241-238-354 -24-231-328-144-123-47-340-281-324-128-202-190-297-304-34-9-0

### 3.1.1 Wykresy błędu w funkcji czasu (3 sąsiedztwa)

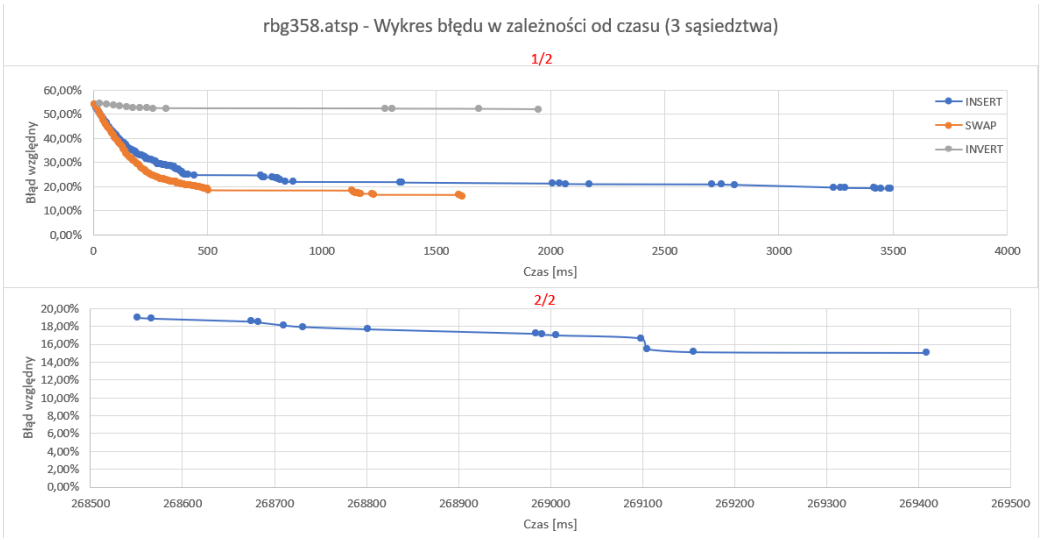
Dla pomiaru pozostałych sąsiedztw, przyjęto takie same parametry jak dla przeglądania sąsiedztwa typu insert.




Rysunek 1: Wykres błędu TS - ftv55




Rysunek 2: Wykres błędu TS - ftv170




Rysunek 3: Wykres błędu TS - rbg358

## 3.2 Wyniki dla Symulowanego wyżarzania

Wyniki przedstawione są dla a = 0.99.

| Plik | Znane optimum | Znalezione rozwiązanie | Błąd względny | Czas[s] | Czas znalezienia rozwiązania [s] | Temp. początkowa | Temp. końcowa |
|---|---|---|---|---|---|---|---|
| ftv55.atsp | 1608 | 1651 | 2,67% | 120 | 100 | 20120 | 3,83E-64 |
| ftv170.atsp | 2755 | 3680 | 33,58% | 240 | 0,001 | 39230 | 2,80E-10 |
| rbg358.atsp | 1163 | 1443 | 24,08% | 360 | 0,53 | 18120 | 2,43543 |

Tabela 2: Najlepsze rozwiązania - symulowane wyżarzanie

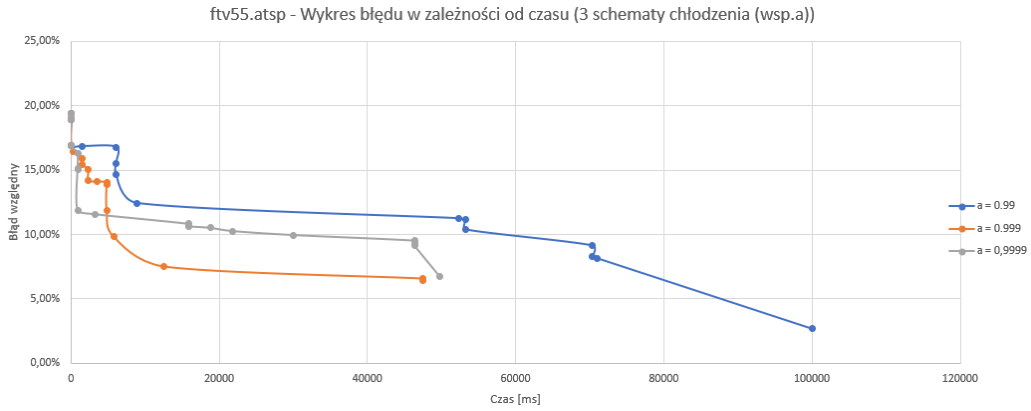
Ścieżki dla przedstawionych rozwiązań są następujące:

**ftv55.atsp:**
0-33-2-13-35-4-5-6-7-32-8-36-9-37-11-38-10-51-12-39-14-15-16-17-52-26-25-24-18-19-20-40-42-21-22-41-50-23-54-27-49-45-30-46-55-34-1-3- 48-31-47-53-43-44-28-29-0

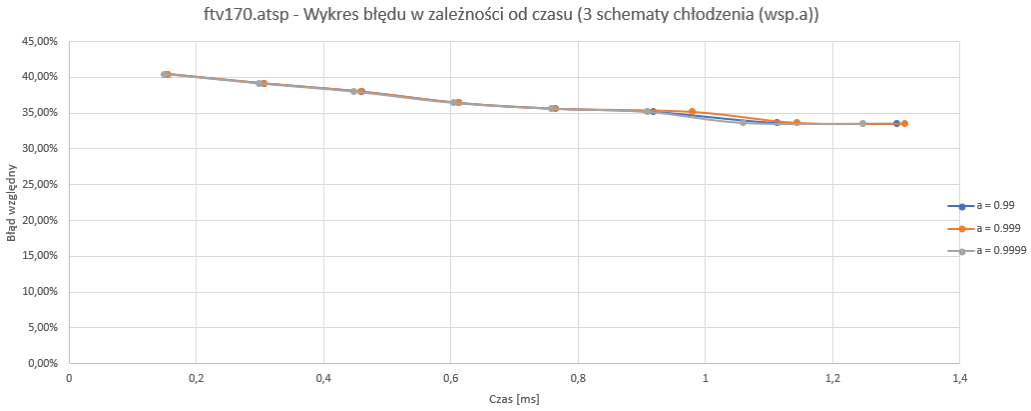
**ftv170.atsp:**
0-1-2-77-73-170-49-50-51-52-53-43-55-54-58-59-60-61-68-69-67-167-70-87-85-86-83-84-71-66-65-64-56-57-62-63-88-153-154-89-90-91-94-96-97- 99-98-95-92-93-166-108-107-106-105-165-163-100-102-103-117-118-119-120-121-122-123-162-101-104-114-110-109-113-164-127-126-125-124-129-128-130-1 31-132-133-134-141-6-7-8-9-10-76-74-75-11-12-18-19-20-21-29-22-23-26-27-28-30-31-33-34-156-40-39-38-37-35-36-157-41-155-42-45-44-46-47-48 -168-72-78-82-79-80-81-3-4-5-169-111-158-32-16-17-24-25-150-161-160-151-152-142-143-144-140-149-148-147-137-136-138-135-139-14-15-159-13-112- 115-116-146-145-0

**rbg358.atsp:**
0-21-69-39-287-239-305-40-4-44-91-10-27-16-17-13-3-164-218-321-260-55-34-9-166-35-18-343-313-227-315-270-329-64-36-29-259-337-137-37-28-24- 41-205-180-47-352-79-45-7-145-25-12-48-11-246-15-284-6-81-56-23-185-209-49-78-5-304-222-104-97-208-63-19-20-120-286-22-148-233-98-114-124-13 5-1-58-243-51-67-95-52-171-53-59-125-311-116-66-14-232-249-283-118-146-68-297-99-70-26-112-103-72-172-255-173-176-183-184-214-76-196-77-42-19 8-80-106-181-328-61-86-165-223-88-123-285-89-276-192-248-90-347-83-92-275-191-206-204-230-182-207-93-32-31-117-333-87-288-119-267-75-212-197-1 32-316-224-122-303-200-126-202-102-210-265-226-127-131-245-128-293-129-54-57-130-111-101-240-295-94-134-82-253-136-85-84-33-108-290-138-38-139- 271-310-140-110-142-109-96-144-107-193-153-65-168-154-43-215-308-100-291-187-50-319-257-155-115-292-156-30-345-289-157-105-158-149-241-189-342-2 63-282-274-159-121-161-152-252-335-147-162-133-143-163-2-247-266-151-349-141-167-8-62-169-348-170-174-46-312-195-175-307-258-177-150-216-178-320 -301-296-299-179-356-331-186-346-238-268-235-188-211-262-332-272-190-73-280-194-74-340-281-324-279-338-213-203-278-251-217-201-236-357-273-219-3 22-354-277-302-269-220-250-113-317-221-199-326-334-228-71-314-229-309-325-350-300-225-294-298-60-344-339-330-261-242-306-336-327-234-244-323-351- 254-160-256-318-231-353-264-341-237-355-0

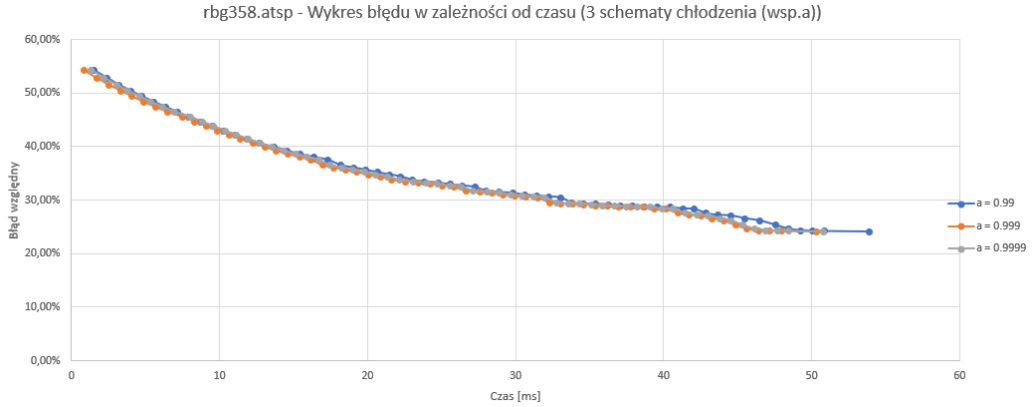
### 3.2.1 Wykresy błędu w funkcji czasu (3 schematy chłodzenia (wsp.a)




Rysunek 4: Wykres błędu SA - ftv55



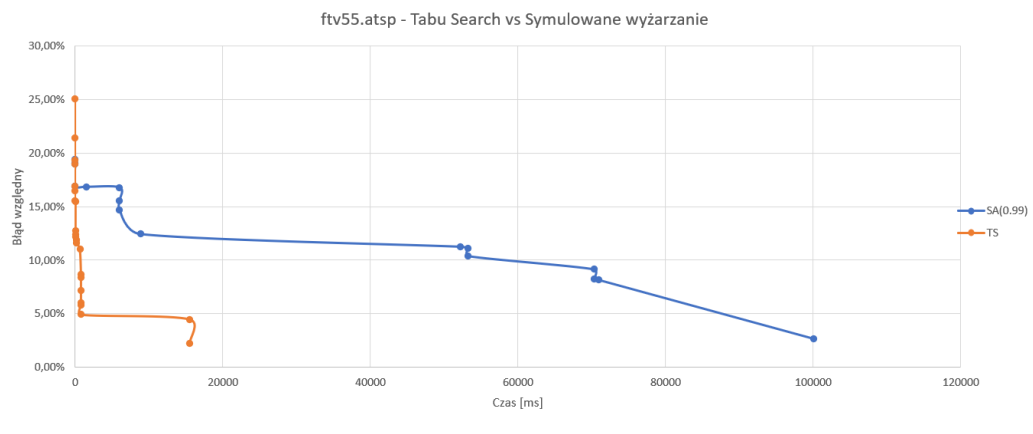

Rysunek 5: Wykres błędu SA - ftv170




Rysunek 6: Wykres błędu SA - rbg358

## 3.3 Tabu Search vs Symulowane wyżarzanie

Porównanie zostało wykonane dla pliku: **ftv55.atsp**, dla najlepszych wyników obu algorytmów.




Rysunek 7: Tabu Search vs Symulowane wyżarzanie - wykres

# 4 Wnioski

Analizując wszystkie wykresy oraz wyniki, dochodzę do wniosków:

- Tabu Search wypada lepiej niż symulowane wyżarzanie, gdyż dla 3 plików, miał lepsze rozwiązania. Różnica może leżej tutaj w charakterystyce plików lub w kalibracji algorytmów.

- Dla 3 plików, najlepsze sąsiedztwo to typu insert.
- Dla SA, zmiana parametru **a**, niewiele pomagała. Tylko i jedynie w przypadku ftv55.atsp, było widać większe zmiany. Możliwe, że trzeba byłoby zmienić schemat schładzania np. na Cauchy'ego.
- Dla TS błąd względny mieścił się w granicach ok. 2% - 21%, co jest dobrym wynikiem, biorąc pod uwagę czas rozwiązania.
- Dla SA błąd względny mieścił się w granicach ok. 2% - 33%, co jest również całkiem dobrym wynikiem, biorąc pod uwagę czas rozwiązania.
- Plik ftv170.atsp okazał się najtrudniejszy, co widać po wynikach. Może to wynikać z optimum, które znajduje się na bardzo małej przestrzeni, przez co ciężko jest je znaleźć.

Podsumowując, używanie metod heurystycznych do ATSP, jest całkiem dobrym pomysłem, biorąc pod uwagę ile wynosiłby czas przeliczenia wszystkich kombinacji dla np. 358 miast. Tutuj w ciągu kilku minut (a nawet często krócej), uzyskujemy wynik który jest na poziomie <20% błędu.