# 

# Problem komiwojażera

# Przedmiot: Inteligencja Obliczeniowa

Wydział: Wydział Zarządzania

# Kierunek: Informatyka i Ekonometria, 3 rok, V semestr

Autorzy:

# Marcin Skałka Tomasz Solarski

Mateusz Mulka   
Jakub Laszczyk

[Tabu Search 3](#_Toc124019591)

[Część teoretyczna: 3](#_Toc124019592)

[Zestaw danych *Dane\_TSP\_48.xlsx* 5](#_Toc124019593)

[Zestaw danych *Dane\_TSP\_76.xlsx* 5](#_Toc124019594)

[Zestaw danych *Dane\_TSP\_127.xlsx* 6](#_Toc124019595)

[Podsumowanie: 7](#_Toc124019596)

[Simulated Annealing 7](#_Toc124019597)

[Część teoretyczna: 7](#_Toc124019598)

[Zestaw danych *Dane\_TSP\_48.xlsx* 8](#_Toc124019599)

[Zestaw danych *Dane\_TSP\_76.xlsx* 9](#_Toc124019600)

[Zestaw danych *Dane\_TSP\_127.xlsx* 10](#_Toc124019601)

[Podsumowanie: 11](#_Toc124019602)

[Algorytm Genetyczny 12](#_Toc124019603)

[Wstęp teoretyczny 12](#_Toc124019604)

[Działanie algorytmu 12](#_Toc124019605)

[Analiza doboru rodzaju sąsiedztwa 12](#_Toc124019606)

[Dobór metody krzyżowania 14](#_Toc124019607)

[Metoda doboru rodziców 16](#_Toc124019608)

[Dane TSP 48 20](#_Toc124019609)

[Dane TSP 76 21](#_Toc124019610)

[Dane TSP 127 22](#_Toc124019611)

[Algorytm najbliższego sąsiada (Nearest Neighbours) 25](#_Toc124019612)

[Wprowadzenie 25](#_Toc124019613)

[Działanie w przypadku problemu komiwojażera 25](#_Toc124019614)

[Algorytm wspinaczki (HILL CLIMBING) 25](#_Toc124019615)

[Wprowadzenie 25](#_Toc124019616)

[Optymalizacja Parametrów 26](#_Toc124019617)

[Opis Parametrów: 26](#_Toc124019618)

[Dla 48 miast 27](#_Toc124019619)

[Dla 76 miast 28](#_Toc124019620)

[Dla 127 miast 29](#_Toc124019621)

[Prezentacja najlepszych wyników przy wybranych parametrach 29](#_Toc124019622)

[Metoda najbliższych sąsiadów (Nearest Neighbours) 29](#_Toc124019623)

[Dla 48 miast 30](#_Toc124019624)

[Dla 76 miast 30](#_Toc124019625)

[Dla 127 miast 30](#_Toc124019626)

[Metoda wspinaczkowa (Hill Climbing) 30](#_Toc124019627)

[Dla 48 miast 30](#_Toc124019628)

[Dla 76 miast 31](#_Toc124019629)

[Dla 127 miast 31](#_Toc124019630)

[Excel solver 31](#_Toc124019631)

# Tabu Search

## Część teoretyczna:

Przeszukiwanie z tabu jest metahuerystyką do rozwiązywania problemów optymalizacyjnych, opartą na iteracyjnym przeszukiwaniu przestrzeni rozwiązań, wykorzystując sąsiedztwo pewnych elementów tej przestrzeni oraz zapamiętując przy tym przeszukiwaniu ostatnie ruchy, dopóki nie zostanie spełniony warunek końcowy. W roku 1986 Fred Glover wydał pierwszą pracę na temat Tabu Search.

Ogólna idea:

Wykonane ruchy np. zamiana danego miasta z innym (metoda swap) są zapisywane na **liście tabu**. Obecność danego ruchu na liście tabu jest tymczasowa – jeżeli wykonamy dany ruch to ten ruch zapisywany jest na liście tabu i będzie tam przez określoną liczbę iteracji k jaką ustali użytkownik. Jeżeli dany ruch jest na liście tabu, oznacza to, że tego ruchu nie możemy wykonać przez kolejnych k iteracji – chyba, że ruch spełnia **kryterium aspiracji**. Celem listy tabu jest wykluczenie prawdopodobieńst zapętleń przy przeszukiwaniu i zmuszenie algorytmu do przeszukiwania nowych regionów przestrzeni przeszukiwań. Szybkość algorytmu nie jest rewelacyjna, ponieważ zamieniamy dane rozwiązanie z najlepszym jego sąsiadem, bez względu na to czy sąsiad jest od niego lepszy czy gorszy. Pozytywnym aspektem tego podejścia jest jednak możliwość szybszego wyjścia z regionu przyciągania ekstremum lokalnego, które niekoniecznie musi być globalnie optymalne (w szczególności, może być czasami bardzo odległe od optymalnego).   
Zazwyczaj **warunkiem końcowym przeszukiwania** jest:  
- zadana liczba iteracji  
- brak poprawy rozwiązania przez określoną liczbę iteracji z rzędu  
- osiągnięcie określonej wartość funkcji celu.

Najbardziej czasochłonnym ruchem tego algorytmu jest przegląd całego sąsiedztwa w każdej z iteracji.  
W naszym przypadku wykorzystujemy dwie struktury sąsiedztwa:  
- swap(i,j) – zamiana miejscami i-tego elementu z j-tym  
- invert(i,j) – odwrócenie kolejności w podciągu zaczynającym się na i-tej pozycji i kończącym się na j-tej pozycji

Kryterium aspiracji:  
Spełnienie kryterium aspiracji pozwala na złamanie tabu, czyli wykonanie ruchu, pomimo że jego atrybuty znajdują się na liście ruchów zabronionych. Najprostszym i najczęściej stosowanym kryterium aspiracji jest zezwolenie na wykonanie ruchu, jeżeli prowadzi on bezpośrednio do uzyskania najlepszego znanego globalnie rozwiązania.

Wykorzystanie pamięci krótkotrwałej i długoterminowej:  
- krótkotrwałej – do zapamiętywania ostatnich wykonanych ruchów i nadpisywania zmian w kolejnych iteracjach algorytmu  
- długotrwałej – do zapamiętywania najbardziej atrakcyjnych elementów przestrzeni przeszukiwań

Algorytm w postaci listy kroków:

1. Wygeneruj losowe rozwiązanie, ustal długość listy tabu
2. Przez określoną liczbę iteracji powtarzaj:
   1. Wygeneruj wszystkich sąsiadów dla danego rozwiązania
   2. KRYTERIUM ASPIRACJI jeżeli ten ruch jest na liście tabu, ale jego wartość jest najlepsza globalnie to wybierz tego sąsiada
   3. Wybierz najlepszego sąsiada jeżeli ten ruch nie jest na liście tabu. Jeżeli jest to wybierz kolejnego najlepszego sąsiada, którego nie ma na liście tabu
   4. Jeżeli wartość funkcji celu dla wybranego sąsiada jest najlepsza dotychczas to uaktualnij najlepsze rozwiązanie i wartość jego funkcji celu.
   5. Zaktualizuj listę tabu
3. Zwróć najlepsze rozwiązanie wraz z jego wartością funkcji celu

Na początku sprawdziliśmy dla jakich parametrów algorytm Tabu Search daje średnie najlepsze wyniki. W tym celu algorytm został włączony 5 razy (była to liczba iteracji dla danego kryterium stopu czyli liczby iteracji), dla każdej kombinacji parametrów i zwrócił średnią z tych uruchomień.

## Zestaw danych *Dane\_TSP\_48.xlsx*

W tym przypadku wartości jakie mogły przyjąć parametry, są zamieszczone w tabelce poniżej:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Długość listy tabu | 2 | 10 | 30 | 70 | 150 |  |
| Liczba iteracji | 30 | 100 | 150 | 300 | 900 | 1800 |

Dla tego zestawu danych najlepsze średnie wraz z kombinacjami parametrów, dla odpowiednich rodzajów sąsiedztwa zostały zamieszczone w tabelkach poniżej:

- rodzaj sąsiedztwa: SWAP

|  |  |  |
| --- | --- | --- |
| Średnia | Liczba iteracji | Długość listy tabu |
| 12813 | 1800 | 150 |
| 13026 | 1800 | 70 |
| 13324 | 900 | 150 |
| 13393,8 | 300 | 70 |
| 13624,8 | 300 | 30 |
| 13692 | 900 | 30 |
| 13713,4 | 1800 | 30 |

- rodzaj sąsiedztwa: INVERT

|  |  |  |
| --- | --- | --- |
| Średnia | Liczba iteracji | Długość listy tabu |
| 10883,6 | 1800 | 70 |
| 10903,2 | 100 | 10 |
| 10923,2 | 1800 | 10 |
| 10924,6 | 900 | 30 |
| 10924,8 | 100 | 70 |
| 10957,2 | 900 | 10 |
| 10975 | 300 | 30 |

Po obliczeniach dla pierwszego zbioru danych stwierdziliśmy, że obliczenia na dalszych zbiorach danych wykonamy tylko dla rodzaju sąsiedztwa INVERT, ponieważ wyniki dla sąsiedztwa SWAP były znacząco gorsze.

Po zobaczeniu wyników, doszliśmy do wniosku, że włączymy algorytm jeszcze raz dla najlepszej, naszym zdaniem, kombinacji parametrów w celu znalezienia najbardziej optymalnego rozwiązania.   
Ustaliliśmy, że tym razem włączymy algorytm 80 razy tylko dla liczby iteracji równej 100 oraz długości listy tabu równej 10, ponieważ najlepsza średnie znacząco nie różni się od tej dla kombinacji, którą mamy zamiar powtarzać, lecz bardzo różni się czasem obliczeń. Dzięki temu algorytm będzie znacznie szybciej się wykonywał, a będzie dawał porównywalnie dobre wyniki.

* liczba iteracji: 100, długość listy tabu: 10, wynik: 10653

## Zestaw danych *Dane\_TSP\_76.xlsx*

W tym przypadku wartości jakie mogły przyjąć parametry, są zamieszczone w tabelce poniżej:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Liczba iteracji | 2 | 10 | 30 | 70 | 150 |  |  |
| Długość listy tabu | 30 | 100 | 150 | 300 | 900 | 1800 | 2400 |

Dla tego zestawu danych najlepsze średnie wraz z kombinacjami parametrów, dla odpowiednich rodzajów sąsiedztwa zostały zamieszczone w tabelkach poniżej:

- rodzaj sąsiedztwa: INVERT

|  |  |  |
| --- | --- | --- |
| Średnia | Liczba iteracji | Długość listy tabu |
| 111419,7 | 300 | 150 |
| 111803,2 | 300 | 10 |
| 111935,3 | 150 | 2 |
| 111988,2 | 900 | 2 |
| 112265,9 | 1800 | 10 |
| 112368,8 | 300 | 70 |
| 112392,6 | 1800 | 30 |

Po zobaczeniu wyników, doszliśmy do wniosku, że włączymy algorytm jeszcze raz dla najlepszej, naszym zdaniem, kombinacji parametrów w celu znalezienia najbardziej optymalnego rozwiązania.   
Ustaliliśmy, że tym razem sprawdzimy dla najlepszej kombinacji parametrów i włączymy algorytm 80 razy tylko dla liczby iteracji równej 300 oraz długości listy tabu równej 150.

* liczba iteracji: 300, długość listy tabu: 150, wynik: 108159,44

## Zestaw danych *Dane\_TSP\_127.xlsx*

W tym przypadku wartości jakie mogły przyjąć parametry, są zamieszczone w tabelce poniżej:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Liczba iteracji | 2 | 10 | 30 | 70 | 150 |  |  |
| Długość listy tabu | 30 | 100 | 150 | 300 | 900 | 1800 | 2400 |

Dla tego zestawu danych najlepsze średnie wraz z kombinacjami parametrów, dla odpowiednich rodzajów sąsiedztwa zostały zamieszczone w tabelkach poniżej:

- rodzaj sąsiedztwa: INVERT

|  |  |  |
| --- | --- | --- |
| Średnia | Liczba iteracji | Długość listy tabu |
| 123245,4568 | 300 | 70 |
| 123995,6389 | 1800 | 2 |
| 124472,6166 | 300 | 10 |
| 124678,9531 | 300 | 30 |
| 124757,0977 | 150 | 30 |
| 125216,5378 | 2400 | 2 |
| 125444,549 | 900 | 10 |

Po zobaczeniu wyników, doszliśmy do wniosku, że włączymy algorytm jeszcze raz dla najlepszej, naszym zdaniem, kombinacji parametrów w celu znalezienia najbardziej optymalnego rozwiązania.   
Ustaliliśmy, że tym razem sprawdzimy dla najlepszej kombinacji parametrów i włączymy algorytm 80 razy tylko dla liczby iteracji równej 300 oraz długości listy tabu równej 70.

* liczba iteracji: 300, długość listy tabu: 70, wynik: 119721,82

## Podsumowanie:

Bardzo ciężko jest dobrać idealne parametry dla algorytmu, wiele kombinacji ma bardzo podobne wyniki. W algorytmach optymalizacji bardzo ważny jest czas, więc dobierając parametry powinniśmy zwrócić uwagę na wynik algorytmu oraz jego czas. Jeżeli dla dwóch różnych wykonań algorytmu, jedno wykonuje się 1,5 raza dłużej i zwraca tylko troszkę lepszy wynik, to powinniśmy wybrać ten szybszy algorytm. Dobierając parametry dla algorytmu Tabu Search zauważyliśmy, że wraz ze wzrostem liczby miast, powinniśmy także zwiększyć nieco liczbę wykonywanych iteracji. Długość listy tabu nie powinna być zbyt mała oraz za duża względem liczby iteracji. Zbyt mała, ponieważ wtedy algorytm jest bardziej intensyfikacyjny, tzn. że może krążyć cały czas wokół jednego miejsca oraz za duża, ponieważ wtedy algorytm będzie bardziej w stylu dywersyfikacji tzn. będzie go to odpychało od miejsc gdzie już był, będzie nastawiony na eksplorację nowych regionów. W naszym przypadku o wiele bardziej sprawdziło się sąsiedztwo tworzone metodą invert.

Najlepsze wyniki oraz parametry dla poszczególnych danych(typ sąsiedztwa invert):

* *Dane\_TSP\_48.xlsx:* liczba iteracji: 100, długość listy tabu: 10, wynik: 10653
* *Dane\_TSP\_76.xlsx:* liczba iteracji: 300, długość listy tabu: 150, wynik: 108159,44
* *Dane\_TSP\_127.xlsx:* wynik: liczba iteracji: 300, długość listy tabu: 70, wynik: 119721,82

# Simulated Annealing

## Część teoretyczna:

Symulowane wyżarzanie jest metaheurystyką do rozwiązywania problemów optymalizacyjnych. Algorytm unika wad algorytmów lokalnego przeszukiwania poprzez ograniczone akceptowanie pogorszeń funkcji celu. Cechą charakterystyczną tej metody jest występowanie parametru sterującego zwanego temperaturą, który maleje w trakcie wykonywania algorytmu. Im wyższą wartość ma ten parametr, tym bardziej chaotyczne mogą być zmiany. Podejście to jest inspirowane zjawiskami obserwowanymi w metalurgii – im większa temperatura metalu, tym bardziej jest on plastyczny. (wyżarzanie – powolne zmniejszanie temperatury do chwili, w której cząsteczki ułożą się wzajemnie i osiągną temperaturę zerową). Algorytm pochodzi od algorytmu Monopolis. Algorytm nie przeszukuje całego sąsiedztwa dla danego rozwiązania. Kolejne rozwiązania przyjmuje na podstawie kryterium akceptacji.

**Kryterium akceptacji:**  
Rozwiązania: i – aktualne oraz j – proponowane  
Koszty (wartość funkcji celu): f(i), f(j)  
Temperatura: T  
Kryterium akceptacji określa czy proponowane rozwiązanie jest akceptowane:  
- Jeżeli f(i) – f(j) jest większe lub równe 0 to przyjmujemy proponowane rozwiązanie z prawdopodobieństwem równym 1.  
- Jeżeli f(i) – f(j) jest mniejsze od 0 to przyjmujemy proponowane rozwiązanie z prawdopodobieństwem równym *e(f(i))−f(j))/T*

**Sposób chłodzenia** określa:  
- początkowa wartości temperatury – powinna być odpowiednio wysoka by zapewnić akceptację wszystkich przejść, im większa temperatura tym bardziej chaotyczne są wybory  
- funkcja redukcji temperatury – stosujemy redukcję geometryczną, ustalamy parametr alfa bliski 1, ponieważ wtedy będzie chętnie wchodził w polepszenia i pogorszenia, a wraz ze zmniejszaniem temperatury algorytm będzie wybierał częściej te lepsze rozwiązania, funkcja redukcji temperatury ma więc postać: alfa\*T  
- końcowa wartość temperatury – algorytm może działać, aż do określonej końcowej temperatury lub przez określoną liczbę iteracji lub przez określoną liczbę iteracji bez poprawy

Tak samo jak w algorytmie Tabu Search trzeba rejestrować najlepsze dotychczasowe rozwiązanie w jakim się było, ponieważ algorytm przyjmuje także rozwiązania pogorszające.

Algorytm w postaci listy kroków:

1. Wygeneruj losowe rozwiązanie, ustal początkową wartość temperatury
2. Przez określoną liczbę iteracji powtarzaj:
   1. Wybierz losowe rozwiązanie j znajdujące się w pobliżu dotychczasowego rozwiązania i.
   2. Jeżeli nowe rozwiązanie jest lepsze to je przyjmij, jeżeli jest gorsze wyznacz prawdopodobieństwo akceptacji *e(f(i))−f(j))/T* . Wylosuj liczbę z przedziału [0,1], jeżeli jest ona mniejsza od prawdopodobieństwa akceptacji to przyjmij to rozwiązanie, w przeciwnym wypadku pozostań przy aktualnym rozwiązaniu.
   3. Jeżeli wartość funkcji celu dla wybranego rozwiązania jest najlepsza dotychczas to uaktualnij najlepsze rozwiązanie i wartość jego funkcji celu.
   4. Zmniejsz temperaturę.
3. Zwróć najlepsze rozwiązanie wraz z jego wartością funkcji celu.

Na początku sprawdziliśmy dla jakich parametrów algorytm Simulated Annealing daje średnie najlepsze wyniki. W tym celu algorytm został włączony 5 razy (była to liczba iteracji dla danego kryterium stopu czyli liczby iteracji) dla każdej kombinacji parametrów i zwrócił średnią z tych uruchomień.

## Zestaw danych *Dane\_TSP\_48.xlsx*

W tym przypadku wartości jakie mogły przyjąć parametry, są zamieszczone w tabelce poniżej:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Liczba iteracji | 1000 | 2000 | 5000 | 10000 | 15000 |  |
| Początkowa temperatura | 5000 | 8000 | 14000 | 20000 | 40000 | 70000 |
| Parametr alfa | 0,98 | 0,94 | 0,89 | 0,85 | 0,8 |  |

Dla tego zestawu danych najlepsze średnie wraz z kombinacjami parametrów, dla odpowiednich rodzajów sąsiedztwa zostały zamieszczone w tabelkach poniżej:

- rodzaj sąsiedztwa: SWAP

|  |  |  |  |
| --- | --- | --- | --- |
| Średnia | Liczba iteracji | Temperatura | Parametr alfa |
| 11042,2 | 15000 | 8000 | 0,94 |
| 11050 | 10000 | 8000 | 0,94 |
| 11058,4 | 15000 | 14000 | 0,94 |
| 11066 | 5000 | 20000 | 0,89 |
| 11100,4 | 15000 | 20000 | 0,94 |
| 11101,8 | 15000 | 40000 | 0,94 |
| 11104,6 | 15000 | 8000 | 0,98 |

- rodzaj sąsiedztwa: INVERT

|  |  |  |  |
| --- | --- | --- | --- |
| Średnia | Liczba iteracji | Temperatura | Parametr alfa |
| 10984,8 | 15000 | 40000 | 0,94 |
| 11034,4 | 10000 | 70000 | 0,98 |
| 11035,2 | 15000 | 70000 | 0,94 |
| 11081,2 | 15000 | 5000 | 0,85 |
| 11082,6 | 10000 | 70000 | 0,94 |
| 11083 | 10000 | 20000 | 0,98 |
| 11086,4 | 15000 | 40000 | 0,89 |

Algorytm zwrócił troszkę lepsze wyniki dla sąsiedztwa typu swap. Tym razem pozostawimy obie metody doboru sąsiedztwa i sprawdzimy je dla pozostałych danych. Z tabelki można zauważyć, że występują głównie dwie największe liczby iteracji tzn. 15000 oraz 10000 oraz wysoka temperatura początkowa dla metody invert, z kolei dla swapa dwa najlepsze wyniki to te z temperaturą niską, na poziomie 8000. Najczęściej występującym parametrem alfa jest ten z wartością 0,94. Zdecydowaliśmy, że włączymy algorytm dla metody invert z parametrami: 15000 iteracji, temperaturą początkową równą 40000 oraz parametrem alfa równym 0,94. Spróbujemy także włączyć algorytm dla metody swap z parametrami: 10000 iteracji, temperaturą początkową równą 8000 oraz parametrem alfa równym 0,94. Celowo nie wybraliśmy najlepszej z kombinacji, ponieważ wynik nie różni się znacząco, a liczba iteracji jest 1,5 razy większa.

* sąsiedztwo: invert, liczba iteracji: 15000, temperatura: 40000, parametr alfa: 0,94, wynik: 10663
* sąsiedztwo: swap, liczba iteracji: 10000, temperatura: 8000, parametr alfa: 0,94, wynik: 10661

Jak widać wyniki, różnią się minimalnie. Więc lepiej wybrać algorytm z wynikiem 10661, wykona się szybciej.

## Zestaw danych *Dane\_TSP\_76.xlsx*

W tym przypadku wartości jakie mogły przyjąć parametry, są zamieszczone w tabelce poniżej:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Liczba iteracji | 1000 | 2000 | 5000 | 10000 | 15000 |  |
| Początkowa temperatura | 5000 | 8000 | 14000 | 20000 | 40000 | 70000 |
| Parametr alfa | 0,98 | 0,94 | 0,89 | 0,85 | 0,8 |  |

Dla tego zestawu danych najlepsze średnie wraz z kombinacjami parametrów, dla odpowiednich rodzajów sąsiedztwa zostały zamieszczone w tabelkach poniżej:

- rodzaj sąsiedztwa: SWAP

|  |  |  |  |
| --- | --- | --- | --- |
| Średnia | Liczba iteracji | Temperatura | Parametr alfa |
| 113363,4908 | 15000 | 5000 | 0,8 |
| 114785,9115 | 15000 | 5000 | 0,94 |
| 115205,2719 | 15000 | 70000 | 0,94 |
| 115407,2972 | 15000 | 20000 | 0,98 |
| 115521,0559 | 15000 | 20000 | 0,94 |
| 115540,4906 | 15000 | 8000 | 0,8 |
| 115680,2844 | 10000 | 8000 | 0,98 |

- rodzaj sąsiedztwa: INVERT

|  |  |  |  |
| --- | --- | --- | --- |
| Średnia | Liczba iteracji | Temperatura | Parametr alfa |
| 114289,2202 | 15000 | 14000 | 0,94 |
| 115130,8993 | 15000 | 20000 | 0,98 |
| 115728,3469 | 15000 | 14000 | 0,98 |
| 116070,5947 | 15000 | 40000 | 0,98 |
| 116547,6649 | 10000 | 40000 | 0,94 |
| 117362,0336 | 10000 | 5000 | 0,98 |
| 117597,0619 | 15000 | 5000 | 0,98 |

Z kolei dla tego rozmiaru danych średnie wartości są lepsze dla metody doboru sąsiedztwa typu swap. W tym dla najlepszego wyniku dla swap, parametr alfa jest najniższy z możliwych na naszej liście. Sprawdzimy, czy to nie jest tylko zbieg okoliczności włączając algorytm dla dwóch najlepszych kombinacji parametrów z metody swap.

* sąsiedztwo: swap, liczba iteracji: 15000, temperatura: 5000, parametr alfa: 0,8, wynik: 109639,79
* sąsiedztwo: swap, liczba iteracji: 15000, temperatura: 5000, parametr alfa: 0,94, wynik: 109269,10

Okazało się, że wynik dla parametru alfa = 0,8 jest lepszy.

## Zestaw danych *Dane\_TSP\_127.xlsx*

W tym przypadku wartości jakie mogły przyjąć parametry, są zamieszczone w tabelce poniżej:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Liczba iteracji | 1000 | 2000 | 5000 | 10000 | 15000 |  |  |
| Początkowa temperatura | 5000 | 8000 | 14000 | 20000 | 40000 | 70000 | 100000 |
| Parametr alfa | 0,98 | 0,94 | 0,89 | 0,85 | 0,8 |  |  |

Dla tego zestawu danych najlepsze średnie wraz z kombinacjami parametrów, dla odpowiednich rodzajów sąsiedztwa zostały zamieszczone w tabelkach poniżej:

- rodzaj sąsiedztwa: SWAP

|  |  |  |  |
| --- | --- | --- | --- |
| Średnia | Liczba iteracji | Temperatura | Parametr alfa |
| 139331,6875 | 15000 | 70000 | 0,8 |
| 140550,802 | 15000 | 14000 | 0,94 |
| 141414,9322 | 15000 | 100000 | 0,98 |
| 142141,3861 | 15000 | 8000 | 0,94 |
| 142321,3863 | 15000 | 8000 | 0,89 |
| 142411,1132 | 15000 | 20000 | 0,8 |
| 142476,288 | 15000 | 100000 | 0,85 |

- rodzaj sąsiedztwa: INVERT

|  |  |  |  |
| --- | --- | --- | --- |
| Średnia | Liczba iteracji | Temperatura | Parametr alfa |
| 141514,479 | 15000 | 20000 | 0,98 |
| 142796,6518 | 15000 | 14000 | 0,98 |
| 143061,8637 | 15000 | 40000 | 0,98 |
| 143978,3601 | 15000 | 100000 | 0,98 |
| 144613,3744 | 15000 | 70000 | 0,98 |
| 146878,2031 | 15000 | 8000 | 0,98 |
| 151216,3911 | 15000 | 70000 | 0,94 |

Dla danych o liczbie miast równej 127, lepsze średnie wyniki zwróciło sąsiedztwo swap. Znów dla metody swap na pierwszym miejscu znalazł się parametr alfa = 0,8. Z kolei dla sąsiedztwa invert znów wartość parametru alfa nie jest niższa niż 0,94. Zdecydowaliśmy się włączyć algorytm dla dwóch rodzajów sąsiedztwa:

* sąsiedztwo: swap, liczba iteracji: 15000, temperatura: 70000, parametr alfa: 0,8, wynik: 134130,72
* sąsiedztwo: invert, liczba iteracji: 15000, temperatura: 20000, parametr alfa: 0,98, wynik: 134151,94

Średnie najwyższe wartości wskazywały, że powinnismy użyć sąsiedztwa swap i wykonanie algorytmu także to potwierdziło. Znów osiągamy dobry wynik stosując niski parametr alfa.

## Podsumowanie:

W przypadku algorytmu symulowanego wyżarzania, także staralilismy się dobrać parametry tak aby algorytm był najbardziej optymalny i zwracał najlepsze wyniki funkcji celu. Dla każdego ze zbioru danych algorytm najlepiej poradził sobie stosując metodę generowania sąsiedztwa swap, zarazem najlepszą wartością liczby iteracji była najwyższa z możliwych, więc powinniśmy stosować dużą liczbę iteracji przy tym algorytmie, a jednocześnie patrzeć na czas oraz wynik algorytmu, aby był optymalny. Ciężko powiedzieć jaką najlepszą wartość powinna mieć temperatura, ponieważ dla przykładu dla 48 miast jest ona zbliżona do wartości optymalnej, a dla większych danych jest zwykle taka sama lub mniejsza niż wartość optymalna. Parametr alfa równy 0,8 bardzo dobrze poradził sobie dla sąsiedztwa typu swap. Z drugiej strony alfa wyższe niż 0,94 dobrze sprawdzało się w metodzie tworzenia sąsiadów invert.

* *Dane\_TSP\_48.xlsx:* sąsiedztwo: invert, liczba iteracji: 15000, temperatura: 40000, parametr alfa: 0,94, wynik: 10663
* *Dane\_TSP\_48.xlsx*: sąsiedztwo: swap, liczba iteracji: 10000, temperatura: 8000, parametr alfa: 0,94, wynik: 10661
* *Dane\_TSP\_76.xlsx:* sąsiedztwo: swap, liczba iteracji: 15000, temperatura: 5000, parametr alfa: 0,8, wynik: 109639,79
* *Dane\_TSP\_76.xlsx:* sąsiedztwo: swap, liczba iteracji: 15000, temperatura: 5000, parametr alfa: 0,94, wynik: 109269,10
* *Dane\_TSP\_172.xlsx:* sąsiedztwo: swap, liczba iteracji: 15000, temperatura: 70000, parametr alfa: 0,8, wynik: 134130,72
* *Dane\_TSP\_172.xlsx:* sąsiedztwo: invert, liczba iteracji: 15000, temperatura: 20000, parametr alfa: 0,98, wynik: 134151,94

# Algorytm Genetyczny

## Wstęp teoretyczny

Jest to rodzaj heurystyki szukającej wystarczająco dobrego rozwiązania pośród przestrzeni potencjalnych rozwiązan. Nazwa algorytmów genetycznych nie została ustanowiona bez powodu. Algorytmy te w pewien sposób naśladują mechanizmy ewolucyjne, dobierając rodziców i tworząc z nich dzieci, przy jednoczesnym zachowaniu pewnych fragmentów, krzyżowaniu cech dwójki rodziców, a także dokonując mutacje.

## Działanie algorytmu

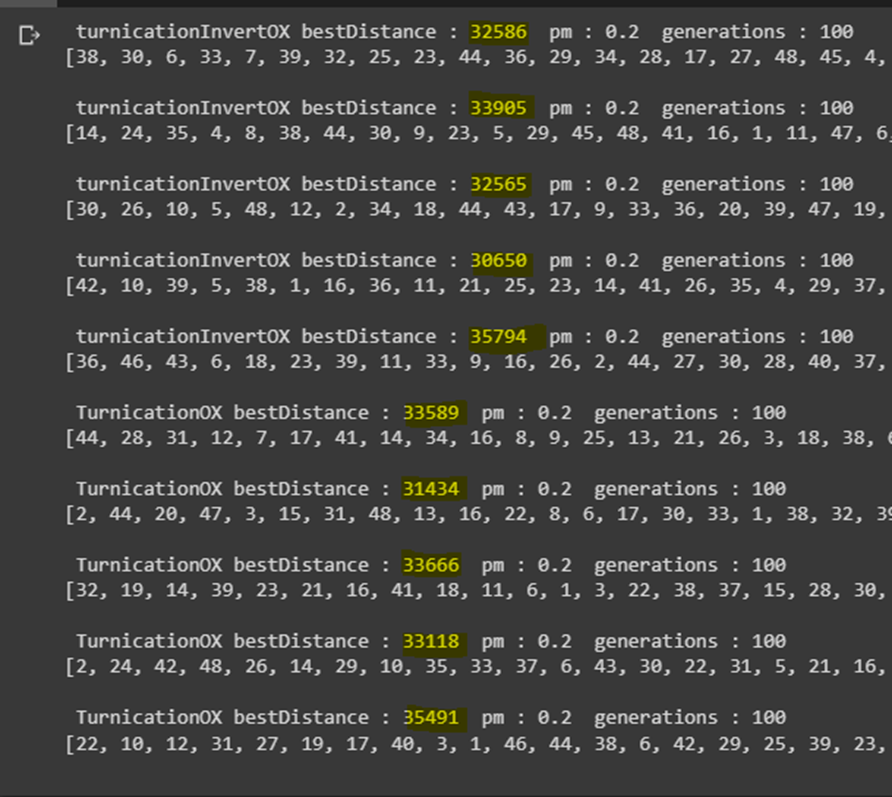
W pierwszej iteracji startowa populacja zostaje wylosowana, jest to ciąg kolejności miast bez powtórzeń, której wielkość uzależniona jest od rozmiaru danych. Z populacji startowej wybierani są rodzice, w przypadku tego projektu zdecydowaliśmy się na użycie metod tournament oraz turnication. Metoda tournament polega na losowaniu dwójki osobników z bazowej populacji i wyborze lepszego osobnika jako rodzica (dla którego trasa jest krótsza). Schemat powtarzany dwukrotnie dla każdego dziecka, a następnie powtarzany jest aż do osiągnięcia oczekiwanej ilości rodziców. Na tak przygotowanej populacji rodziców przeprowadzana jest krzyżówka (crossover), w naszym przypadku korzystamy z metod PMX oraz OX, w obu metodach losowane są fragmenty rodziców, który przepisywane są do dzieci, a następnie reszta dziecka uzupełniania jest po kolei (zaczynając od prawej strony tego fragmentu), z drugiego rodzica z uwzględnieniem zasady, że wartości z poszczególnych komórek nie mogą się powtarzać. Po dokonaniu krzyżówki otrzymujemy populacje dzieci, populacja ta powinna dawać nie gorsze wyniki niż populacja rodziców, a w pożądanym przypadku będzie dawać wyniki lepsze. Na tym etapie musimy jeszcze przeprowadzić mutację. Jednym z parametrów funkcji jest szansa na mutację. Mutacje użyte w tym algorytmie są takie same jak w poprzednich algorytmach, używamy metod invert oraz swap. Mutacja rozwiązuje bardzo istotny problem, bez niej istnieje duże prawdopodobieństwo zamknięcia się naszych wyników w znacznie ograniczonym podzbiorze rozwiązań, w przypadku natrafienia na ekstremum lokalne albo na przykład, gdy osobniki populacji startowej są stosunkowo podobne do siebie. Po przejściu pełnej iteracji algorytmu populacja dzieci przekazywana jest do kolejnej iteracji, jako populacja rodziców. Algorytm powtarzany jest do uzyskania oczekiwanej ilości populacji lub do momentu, w którym przez 10 populacji pod rząd najlepszy wynik dla danej populacji się nie poprawi.

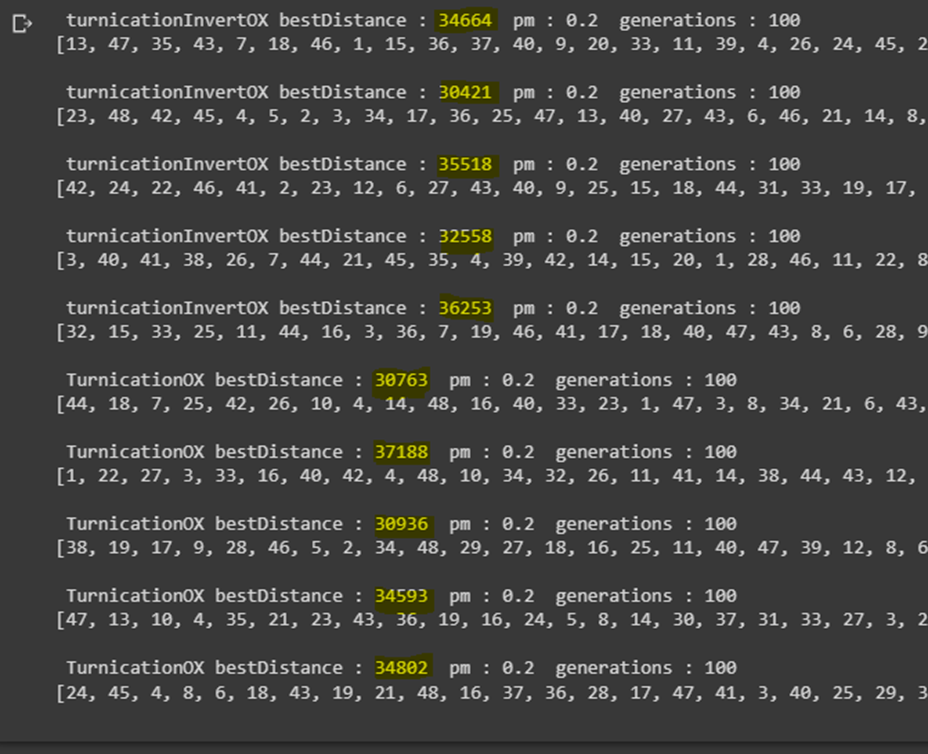
## Analiza doboru rodzaju sąsiedztwa

Pierwszym argumentem który będziemy weryfikować jest rodzaj sąsiedztwa, z uwagi na specyfikę tych metod spodziewamy się braku szczególnych zmian w wynikach wynikających z doboru metody.

Rodzaje sąsiedztwa które testujemy to invert oraz swap.

Zdecydowaliśmy się na testowe puszczenie 10 iteracji na danych TSP 48 dla stałej szansy na mutację = 20% oraz dla 100 pokoleń, przy stałej metodzie doboru rodziców – turnication oraz przy ostałej funkcji crossover – OX.

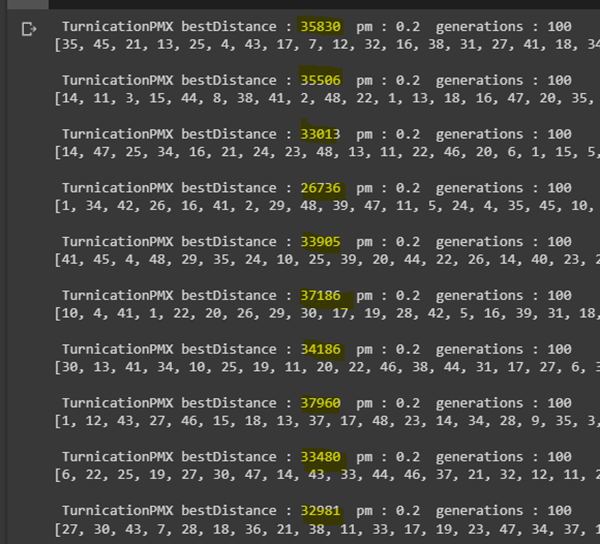


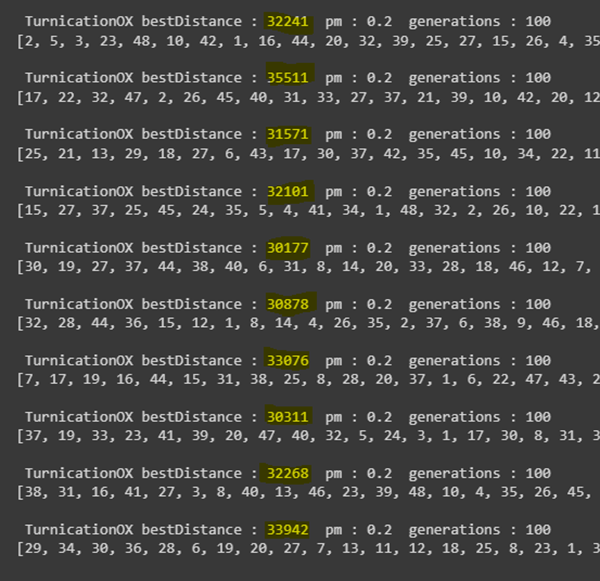


Jak możemy zauważyć wyniki nie różnią się istotnie między sobą, a różnice tu występujące wynikają jedynie z losowości, jednocześnie wartości minimalne dla obu metod są do siebie zbliżone. Z tego powodu w dalszych rozważaniach będziemy przyjmować metodę swap ze względu na mniejszą złożoność obliczeniową. Argumentami pozostałymi do analizy są zatem : metoda doboru rodziców, metoda krzyżowania, prawdopodobieństwo mutacji oraz wielkość populacji.

## Dobór metody krzyżowania

Z uwagi na podobieństwo (kopiowanie środkowego członu osobnika) między metodami crossover OX oraz crossover PMX, a także z uwagi na element losowy, można przypuszczać, że również dobór metody krzyżowania nie będzie miał aż tak dużego przełożenia na wynik, jak dobór rodziców, ilość generacji czy też szansa na mutację. Aby to sprawdzić ponownie na danych TSP 48 i przy stałych argumentach pm oraz generations weryfikujemy poprawność tej tezy.





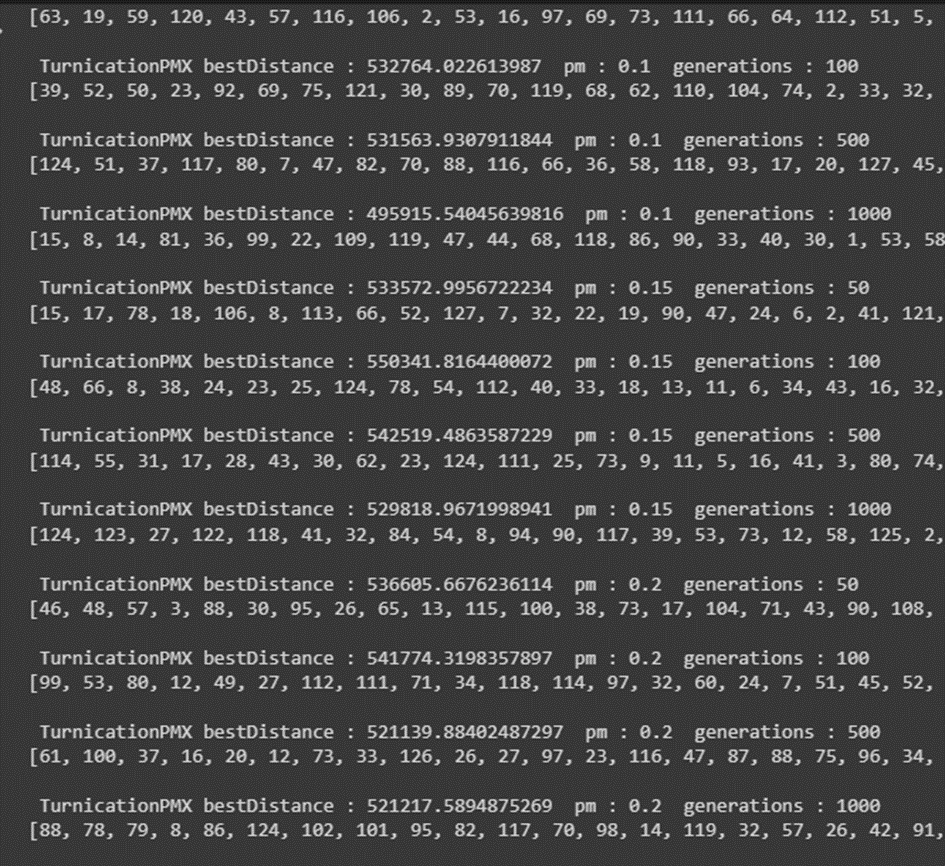
Dla metody PMX średni wynik wynosi 34078,3 , a wartość minimalna to 26736.

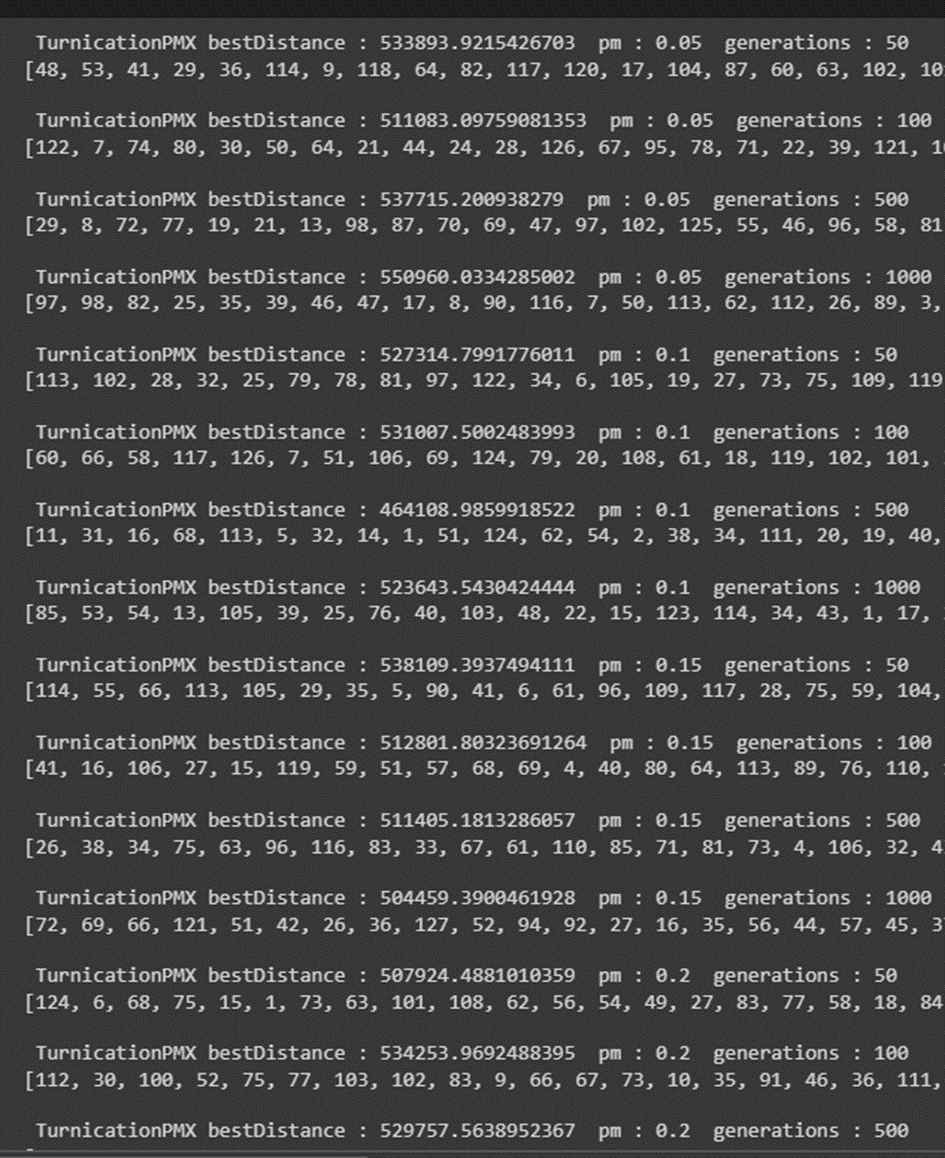
Dla metody OX średni wynik wynosi 32207,6 , a wartość minimalna to 30177.

Jak widzimy wyniki nie różnią się istotnie między sobą, a ewentualne różnice wynikają z elementu losowości. Przy naszej implementacji PMX powinien mieć też przewagę złożoności obliczeniowej nad OX, a minimalna wartość dla PMX wyszła nam mniejsza, niż dla OX. Z tych powodów w dalszych rozważaniach przyjmujemy metodę PMX jako metodę krzyżowania rodziców.

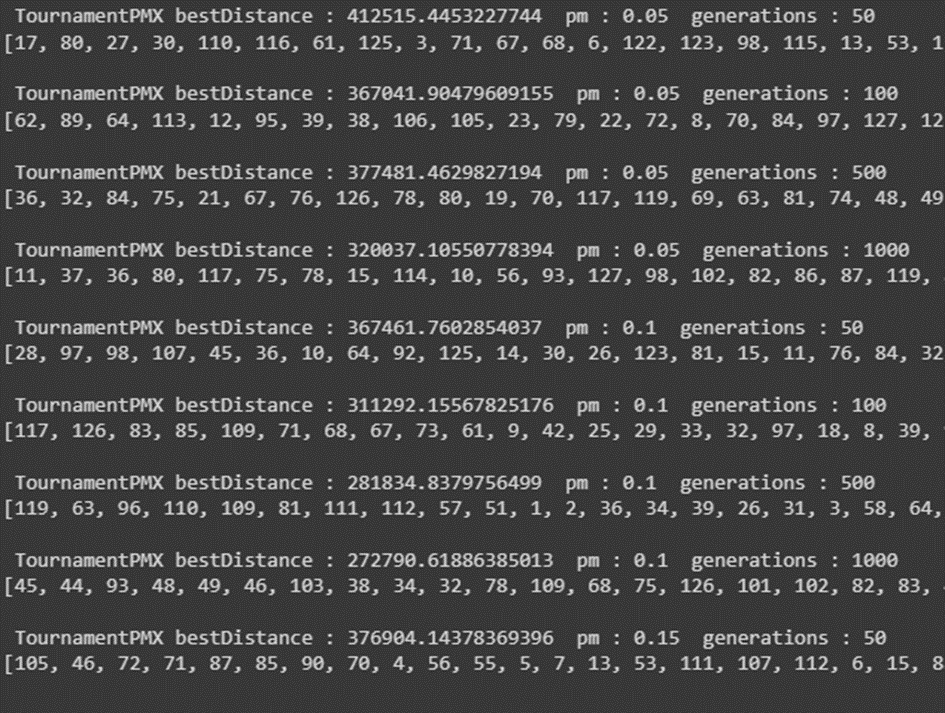
## Metoda doboru rodziców

Metoda doboru rodziców również może bardzo dużo zmieniać w perspektywie otrzymywanych wyników, dzieje się tak dlatego, że turniej wybiera lepszego osobnika jako rodzica, a turnication jedynie losuje rodziców z najlepszych 66,(7)% osobników bez weryfikowania który z nich jest lepszy.





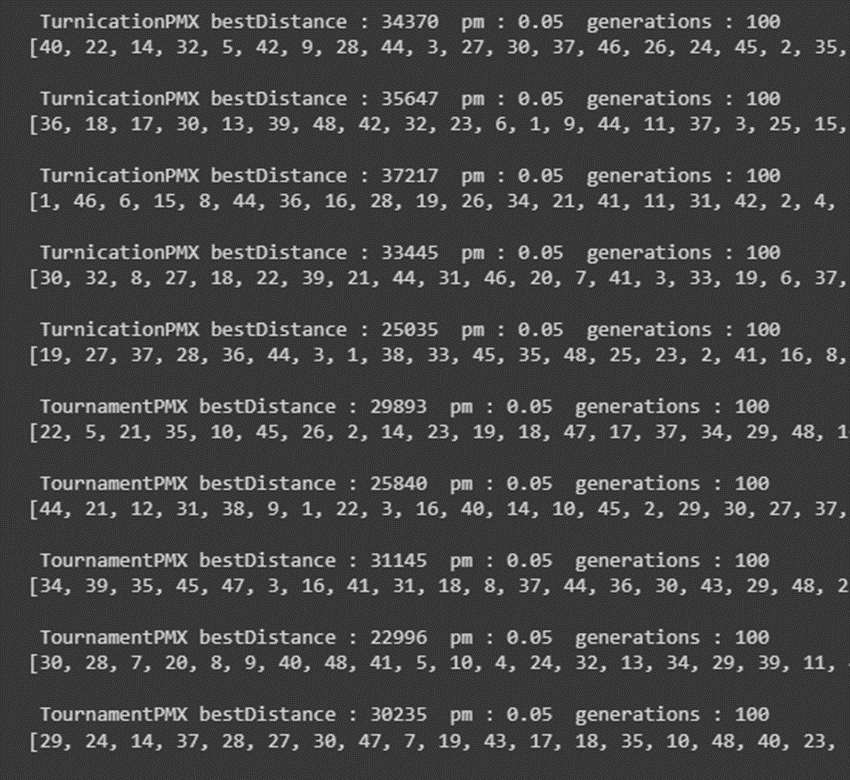
Weryfikując to na zbiorze 127 widzimy, że przy różnych rozmiarach pm oraz generations dla TurnicationPMX wyniki są cały czas oscylują w okolicach 500 000



Jednocześnie wyniki dla tournament PMX w przypadku tych iteracji nie wyszły ponad około 412 500, uzyskując zaledwie w siódmej iteracji wynik poniżej 300 000

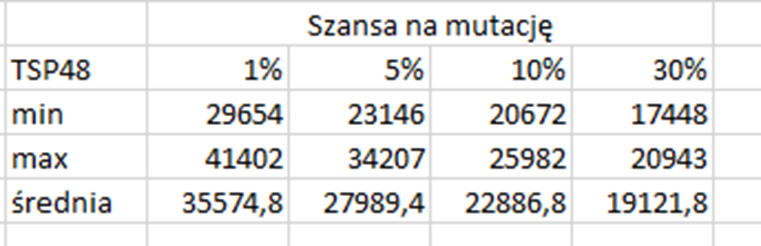
Podobną sytuację możemy zaobserwować na poniższym zrzucie ekranu, średnie wyniki dla metody doboru tournament są istotnie niższe niż dla turnication. Z tego powodu decydujemy się na używanie metody tournament.

Argumentami pozostałymi do analizy są zatem : prawdopodobieństwo mutacji oraz wielkość populacji. W przypadku tych argumentów przewidujemy możliwie większe różnice w wynikach, przeanalizujemy je zatem z osobna dla wszystkich zbiorów danych.



## Dane TSP 48

Na danych TSP 48 chcemy zweryfikować dla jakiej szansy na mutację algorytm ma największe szanse na uzyskanie niskiego wyniku. W tym celu dla wartości 1%, 5%, 10%, 30% puścimy algorytm wielokrotnie i przeanalizujemy zmianę wyników.



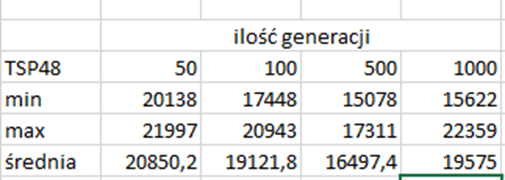
Jak widzimy wraz ze zwiększaniem wartości szansy na mutacje poprawiają się nam wyniki, dla rozpatrywanych wartości parametrów najlepsze wyniki osiąga parametr 30%

Na tym etapie naszym najlepszym wynikiem jest 17448 , uzyskanym dla miast o kolejności :

[37, 19, 27, 6, 48, 42, 26, 4, 35, 45, 10, 24, 13, 11, 15, 40, 36, 28, 7, 30, 17, 43, 25, 5, 39, 32, 21, 47, 46, 38, 31, 18, 33, 23, 14, 34, 29, 2, 41, 3, 22, 16, 1, 8, 9, 12, 20, 44]

Następnie rozpatrywać będziemy wpływ zmiany ilości populacji , pamiętać tu musimy o możliwości trafienia w ekstremum lokalne, przez co również w tym przypadku obliczenia musimy puścić wielokrotnie

Ilości generacji które będziemy rozpatrywać to 50, 100, 500, 1000



Najlepsze wyniki, zarówno w przypadku wartości średniej jak i minimalnej, otrzymaliśmy dla liczby pokoleń równej 500. Należy tu jednak zwrócić uwagę, że z uwagi na losowość niekiedy wyższa ilość generacji wcale nie musi oznaczać lepszego wyniku, w naszym kodzie punktem krytycznym jest przejście z 500 na 1000 generacji bo będzie jeszcze widoczne w późniejszych rozważaniach.

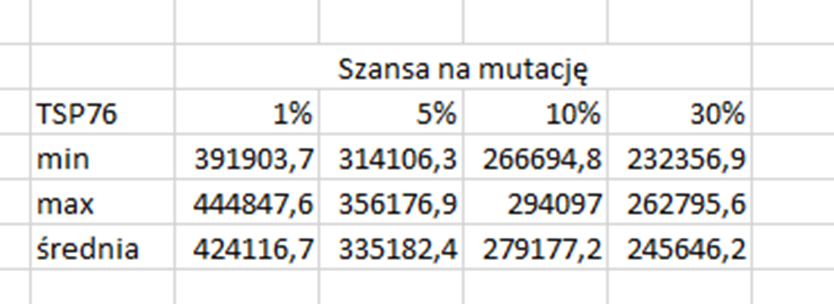
Naszym nowym najlepszym wynikiem jest 15078, wynik ten był uzyskany dla kolejności miast :

[29, 26, 4, 2, 45, 35, 24, 32, 48, 10, 42, 5, 25, 39, 13, 20, 30, 19, 37, 17, 43, 27, 6, 28, 36, 18, 7, 44, 33, 12, 15, 11, 47, 21, 14, 34, 3, 23, 40, 9, 38, 31, 46, 8, 1, 16, 22, 41]

## Dane TSP 76

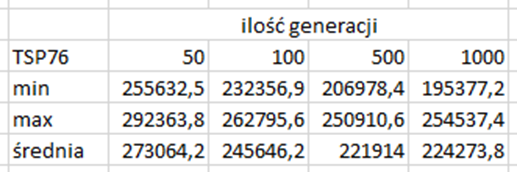
Na danych TSP 48 chcemy zweryfikować dla jakiej szansy na mutację algorytm ma największe szanse na uzyskanie niskiego wyniku. W tym celu dla wartości 1%, 5%, 10%, 30% puścimy algorytm wielokrotnie i przeanalizujemy zmianę wyników.

Wyniki oraz wnioski z nich płynące są analogiczne jak w przypadku TSP 48 , ponownie szansa 30% wypadła tu najlepiej



Na tym etapie minimalny wynik jaki uzyskaliśmy wynosi w przybliżeniu 232356,9 i uzyskany został dla kolejności miast : [48, 53, 40, 60, 42, 54, 76, 75, 2, 3, 8, 7, 5, 20, 19, 29, 10, 13, 74, 14, 4, 21, 25, 22, 6, 62, 71, 72, 73, 63, 64, 51, 56, 58, 35, 18, 37, 38, 39, 41, 55, 46, 24, 30, 31, 28, 27, 26, 44, 45, 33, 32, 9, 16, 17, 36, 59, 61, 57, 65, 66, 67, 68, 70, 52, 34, 11, 15, 12, 1, 23, 43, 49, 50, 69, 47]

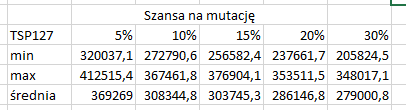
Przejdźmy zatem do analizy ilości generacji. Ponownie wielokrotnie uruchomiliśmy nasz kod dla takich samych parametrów : 50, 100, 500 i 1000 populacji. Wyniki widoczne na poniższym zrzucie ekranu nie są już tak jednoznaczne, jak w przypadku danych TSP 48.



Tak jak wcześniej zauważyliśmy, zmiana ilości iteracji z 500 na 1000 nie wpływa w naszym algorytmie na znaczną poprawę wyników. W tym przypadku uzyskaliśmy najniższy średni wynik ponownie dla 500 populacji, jednak tym razem najniższa trasa została uzyskana w wyniku iterowania algorytmu 1000 razy. Nasz nowy najlepszy wynik dla 76 miast to w przybliżeniu 195377,2 , a został on uzyskany dla trasy : [47, 46, 25, 4, 7, 8, 14, 13, 15, 16, 11, 26, 48, 49, 50, 71, 72, 73, 56, 42, 43, 53, 52, 51, 55, 57, 63, 64, 62, 61, 59, 58, 66, 65, 70, 67, 54, 34, 35, 39, 38, 40, 44, 45, 24, 22, 23, 1, 21, 30, 19, 6, 74, 12, 9, 10, 37, 18, 17, 36, 41, 60, 5, 3, 75, 76, 2, 20, 31, 28, 32, 33, 29, 27, 69, 68]

## Dane TSP 127

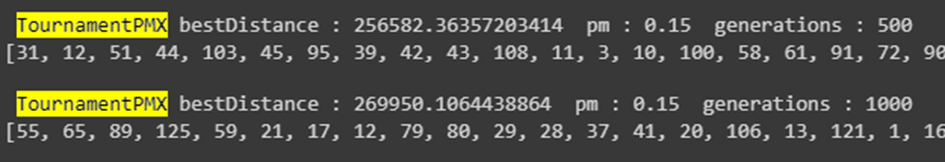
Z uwagi na złożoność problemu dla 127 miast postanowiliśmy jeszcze dokładniej sprawdzić nasze rozwiązania, tym razem nie sprawdzaliśmy rozwiązań dla 1%, 5%, 10%, 30% szans na mutację, z uwagi na wcześniej uzyskiwany najlepszy wynik dla wartości 30% zrezygnowaliśmy ze sprawdzenia trasy dla 1% szans na mutację, zamiast tego do naszej puli dodaliśmy 15% oraz 20% szans na mutację, by sprawdzić czy najlepsze rozwiązanie nie leży gdzieś po środku, między wcześniej sprawdzanymi wartościami 10% oraz 30% .

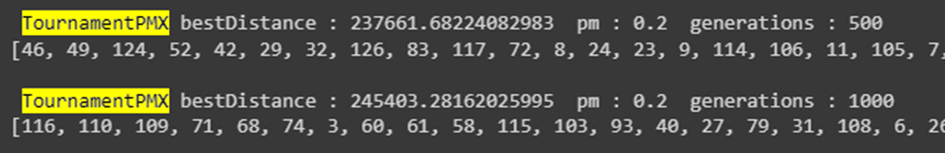


Jak możemy zaobserwować wraz ze wzrostem szansy na mutacje maleje minimalna oraz średnia trasa. Ponownie najlepszy wynik uzyskaliśmy przy szansie na mutację równą 30%. Na tym etapie nasz najmniejszy wynik to 205824.5 uzyskany dla ciągu miast :

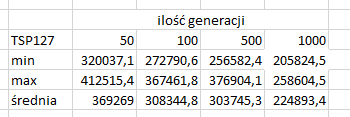
[80, 117, 84, 75, 70, 68, 71, 60, 62, 59, 110, 126, 82, 102, 101, 83, 90, 113, 89, 92, 99, 65, 55, 124, 52, 35, 43, 42, 34, 30, 41, 14, 27, 31, 26, 33, 28, 122, 39, 36, 114, 3, 9, 11, 24, 17, 21, 72, 18, 78, 76, 108, 6, 106, 7, 16, 2, 13, 100, 58, 61, 74, 73, 69, 109, 96, 63, 119, 88, 87, 86, 85, 104, 125, 91, 64, 66, 47, 103, 45, 44, 51, 115, 48, 53, 49, 94, 54, 57, 56, 10, 120, 116, 67, 77, 1, 15, 20, 4, 23, 8, 19, 22, 123, 97, 98, 127, 107, 111, 118, 46, 112, 93, 95, 29, 81, 32, 25, 38, 40, 121, 5, 50, 105, 37, 12, 79]

Dokonując wyboru ilości generacji po raz kolejny doszliśmy do podobnych wniosków. Wraz ze wzrostem ilości generacji średnio łączna trasa ulega zmniejszeniu. Można przyjąć, że dla 1000 pokoleń wyniki są najlepsze, jednak ponownie napotkaliśmy kilka wyjątków od tej reguły :





Wyniki dla poszczególnych liczności populacji prezentują się następująco:



Nasz najlepszy wynik pozostaje bez zmian i wynosi 205824.5

Średnie wyniki dla 1000 populacji prezentują się znacznie lepiej niż dla 500 populacji mimo pojedynczych odstępstw od tej reguły.

Podsumowanie

W przypadku tego algorytmu analizowaliśmy wpływ następujących parametrów :

* Rodzaj sąsiedztwa – invert(odwrócenie kolejności w podciągu zaczynającym się na i-tej pozycji i koczącym się na j-tej pozycji) lub swap (zamiana miejscami i-tego i j-tego elementu)
* Rodzaj doboru rodziców – tournament lub turnication
* Rodzaj mutacji – OX lub PMX
* Szansa na mutację – 1%/5%/10%/30% , w przypadku danych TSP 127 również 15% oraz 20%, z pominięciem 1%
* Ilość generacji – 50/100/500/1000

W przypadku doboru rodzaju sąsiedztwa oraz rodzaju mutacji nie zauważyliśmy znaczących zmian w wynikach, tournament selection uzyskał znaczną przewagę nad turnication, najlepsze wyniki uzyskaliśmy dla 30% szans na mutację, w przypadku ilości generacji z uwagi na możliwość znalezienia się w ekstremum lokalnym oraz zależność od pierwszej populacji – wybranej losowo , często 500 populacji uzyskiwało podobne wyniki jak 1000 populacji. Mimo tego faktu w makro skali wyniki dla 1000 populacji są lepsze niż dla 500. Widać jednak znaczną różnicę porównując 500/1000 populacji do 100 i 50 populacji, niższe wartości uzyskiwały znacznie gorsze wyniki.

# Algorytm najbliższego sąsiada (Nearest Neighbours)

## Wprowadzenie

Jest to algorytm wykorzystujący strategię zachłanną co oznacza że dokonuje on wybory najbardziej opłacanego na dany moment, dokonuje wyborów lokalnie najlepszych, licząc na to, że będzie to również optymalne rozwiązanie globalne. Jednak zazwyczaj w przypadku algorytmów zachłannych tak nie jest. Jednocześnie są to algorytmy deterministyczne – nie ma w nich losowości. Złożoność pamięciowa algorytmu jest bardzo niewielka (warto pamiętać jedynie, które wierzchołki zostały już odwiedzone). Na ogół rozwiązania znalezione przez ten algorytm są średnio o ok. 25% gorsze od optymalnych w problemie komiwojażera. Jest to jeden z najmniej skutecznych algorytmów, lecz posiada bardzo niską złożoność czasową. Jego wyniki zależą od wierzchołka(miasta) startowego i wyniki dla danego miasta początkowego są zawsze takie same

## Działanie w przypadku problemu komiwojażera

1. Wybieramy miasto (wierzchołek) startowy – ustawiamy go jak odwiedzony i aktualny
2. Wybieramy miasto najbliższe do miasta aktualnego które jednocześnie nie już jest odwiedzone
3. Dołączamy do rozwiązania odległość (krawędź) pomiędzy miastami
4. Znalezione miasto oznaczamy jako odwiedzone oraz aktualne
5. W przypadku gdy zostały nieodwiedzone miasta wracamy do pkt 2
6. Gdy nie ma już miast nie odwiedzonych, dodajemy do rozwiązania odległość pomiędzy ostatnim i początkowym miastem i otrzymujemy wynik – dla danego miasta początkowego.

Dla poznania wszystkich wyników za pomocą tego algorytmu wystarczy podjąć powyższe kroki dla każdego miasta jako początkowe

# Algorytm wspinaczki (HILL CLIMBING)

## Wprowadzenie

Algorytm wspinaczkowy w przeciwieństwie do algorytmu najbliższych sąsiadów nie buduje drogi po kawałku lecz, iteracja zwraca całkowity wynik z trasy która jest w sąsiedztwie do trasy poprzedniej. Wynik gorszy jest odrzucany a wynik lepszy poddawany jest algorytmowi od początku aż do momenty gdy wyniki będzie lokalnie najlepszy. Wadą algorytmu jest wysoka zależność pomiędzy miastem startowym a wynikiem, należy więc wiele razy modyfikować trasę początkową w celu znalezienia najlepszego rozwiązania. Do wada należy również nieodporność algorytmu na maksima lokalne których nie da się już poprawić a mogą nie być globalnie optymalne. Zaletami algorytmu jest szybkie działanie.

Zasada działania:

1. Losujemy trasę początkową i ustalmy jej wynik (odległość całej trasy).
2. Tworzymy listę sąsiadów dla aktualnej trasy, (sposób tworzenia sąsiedztwa może być parametrem algorytmu)
3. Sprawdzamy czy lista sąsiadów nie jest pusta jeśli tak to kończymy algorytm rozwiązanie zwracając aktualny wynik jako maksimum lokalne. W przypadku gdy lista nie jest pusta przechodzimy do kroku następnego.
4. Losujemy jednego z sąsiadów. Sprawdzamy dla niego wynik, jeśli wynik okazał się lepszy, ustalamy sąsiada jako aktualne rozwiązanie i wracamy do punktu 2.
5. Jeśli wynik jest gorszy usuwamy rozwiązanie z listy sąsiadów i wracamy do punktu 3.

Dany algorytm należy przeprowadzić wiele razy losując za każdym razem nowe rozwiązanie początkowe.

# Optymalizacja Parametrów

## Opis Parametrów:

**Liczba iteracji wewnątrz algorytmu –** parametr ten ustawiamy aby ograniczyć ilość przejść między rozwiązaniami w metodzie (1 iteracja to przejście z 1 rozwiązania do „lepszego” sąsiada jako nowego aktualnego rozwiązania) – w skrypcie nazwa zmiennej to ihc\_itteration

**Wartość startowa –** Wartość ta określa maksymalną długość trasy dla trasy początkowej, która wybierana jest losowo, algorytm nie przejdzie do kolejnego kroku dopóki wylosowana trasa początkowa nie osiągnie wystarczająca małej trasy.

**Rodzaj sąsiedztwa:**

**Insert –** w ten rodzaj sąsiedztwo wchodzą wszystkie rozwiązania które od aktualnego rozwiązania różnią się 1 indeksem który przełożony jest pomiędzy inne 2 rożne indeksy, pozostała część rozwiązania musi być identyczna [1,2,3,4] -> [2,**1**,3,4],[2,3,**1**,4],[2,3,4,**1**][1,3,**2**,4][1,3,4,**2**][**3,**1,2,4]…..

**Losowanie miasta startowego –** Losowanie miasta startowego wykonuje się zawsze na samym początku algorytmu więc im więcej razy chcemy wykonać algorytm w celu poszukiwania jak najlepszego wyniku, tym większy musimy ustawić ten parametr - większe znaczeni ma sprawdzenie powyższych 3 parametrów gdyż tą wartość ustalimy na jak najwyższą

## Dla 48 miast

Algorytm uruchomiony na 10 iteracji dla 30 miast startowych, dla każdej z 10 iteracji wybrany został 1 najlepszy wynik z 30 startów. Przedstawiono średnią z 10 wyników oraz najlepszy z nich

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sąsiedztwo i iterację** | | | | | |
| **sąsiedztwo** | **Iteracje w algorytmie** | **Najlepszy Wynik** | | **Średnia** | |
| SWAP | 100 | 13102 | | 14073,5 | |
| 150 | 11456 | | 12569 | |
| **200** | 11656 | | 12473,1 | |
| 300 | 11897 | | 12407,2 | |
| **INSERT** | 100 | 11973 | | 12830,9 | |
| 150 | 10882 | | 11123,9 | |
| **200** | 10816 | | 11026,9 | |
| 300 | 10866 | | 11031,8 | |
| **zmiana parametru wejścia (maksymalny dystans trasy)** | | | | | | |
| **sąsiedztwo** | **wartość** | | **Wynik** | | **Średnia** | |
| **INSERT** | 60000 | | 10886 | | 11038,3 | |
| 50000 | | 10711 | | 11011,8 | |
| **45000** | | 10801 | | 10998,06 | |
| 40000 | | 10797 | | 11006,82 | |

Porównując wyniki można zauważyć że lepsze rezultaty zwraca sąsiedztwo typu insert oraz nie ma potrzeby zwiększać liczby iteracji powyżej 200.

Najlepsze wartość wejścia została osiągnięta dla wartości 45000

## Dla 76 miast

Algorytm uruchomiony na 15 iteracji dla 10 miast startowych i 2 rodzajów sąsiedztwa, dla każdej z 15 iteracji wybrany został 1 najlepszy wynik z 10 startów. Przedstawiono średnią z 15 wyników oraz najlepszy z nich

|  |  |  |  |
| --- | --- | --- | --- |
| **Sąsiedztwo** | **średnia** | **najlepszy wynik** |  |
|  |
| N1 - SWAP | 155220 | 143191,08 |  |
|  |
| **N2 - INSERT** | 120267 | 116031,26 |  |
|  |

Algorytm uruchomiony na 10 iteracji dla 10 miast startowych i 3 wartości parametru wartość startowa, dla każdej z 10 iteracji wybrany został 1 najlepszy wynik z 10 startów. Przedstawiono średnią z 10 wyników oraz najlepszy z nich

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | **Wartość startowa** | | **średnia** | **najlepszy wynik** |  | **Iteracje w algorytmie** | **średnia** | **najlepszy wynik** | |  | | 900000 | | 122310 | 118653 |  | 250 | 122182 | 117118 | | 700000 | | 120395 | 118221 |  | **500** | **119233** | **114370** | | **500000** | | **114076** | **118259** |  | 1000 | 119726 | 114065 | |  |  |  |  |  |  |  |  |   Jak widzimy dla 76 miast najlepiej wybrać:   * Sąsiedztwo: insert * Wartość startowa: 500000 (Dla niższych wartości program się zawieszał) * Iteracje w algorytmie: 500 (brak popraw dla większej liczby iteracji)  Dla 127 miast Algorytm uruchomiony na 7 iteracji dla 3 miast startowych i (2 parametrów sąsiedztwa, 3 wartości parametru „wartość startowa” oraz 3 wartości parametru „iteracje w algorytmie”), dla każdej z 7 iteracji wybrany został 1 najlepszy wynik z 3 startów. Przedstawiono średnią z 7 wyników oraz najlepszy z nich  Gdy parametry są stałe (czyli w momencie gdy badamy inną zmienną): sąsiedztwo: insert, wartość startowa 700000, iteracje w algorytmie:1000      Jak widzimy dla 127 miast najlepiej wybrać:   * Sąsiedztwo: insert * Wartość startowa: 580 000 * Iteracje w algorytmie: 1000 – brak poprawy średniej dla 2000  Prezentacja najlepszych wyników przy wybranych parametrachMetoda najbliższych sąsiadów (Nearest Neighbours)Dla 48 miast **Miasto startowe** 35  **Najlepsza trasa** [35, 45, 10, 24, 42, 5, 48, 39, 32, 21, 47, 11, 23, 14, 25, 13, 12, 15, 33, 46, 44, 18, 7, 28, 36, 30, 6, 37, 19, 27, 43, 17, 20, 40, 9, 1, 8, 38, 31, 22, 16, 3, 34, 41, 29, 2, 26, 4, 35]  **Długość trasy** 12012 Dla 76 miast **Miasto startowe** : 16  **Najlepsza trasa**  [16, 15, 13, 14, 12, 11, 17, 18, 37, 36, 35, 34, 40, 41, 60, 59, 58, 57, 63, 64, 62, 61, 55, 56, 51, 66, 65, 50, 49, 52, 53, 54, 42, 43, 28, 29, 30, 31, 19, 20, 5, 6, 7, 8, 9, 10, 4, 3, 2, 1, 23, 22, 21, 25, 24, 46, 45, 44, 48, 47, 69, 68, 67, 70, 71, 72, 73, 39, 38, 32, 33, 27, 26, 75, 76, 74, 16]  **Długość trasy** 130921.00 Dla 127 miast **Miasto startowe** 117  **Najlepsza trasa** [117, 84, 81, 126, 82, 83, 75, 76, 78, 80, 79, 77, 18, 21, 17, 22, 4, 23, 24, 6, 106, 15, 108, 20, 19, 72, 8, 9, 11, 114, 105, 7, 1, 16, 2, 51, 57, 54, 45, 103, 44, 35, 36, 37, 41, 14, 12, 31, 27, 30, 43, 34, 39, 38, 26, 25, 33, 122, 28, 29, 32, 42, 40, 121, 5, 56, 124, 52, 50, 13, 115, 10, 120, 3, 90, 116, 60, 62, 61, 91, 58, 64, 100, 113, 66, 55, 47, 49, 53, 118, 48, 46, 94, 112, 111, 107, 127, 93, 95, 123, 97, 98, 101, 102, 63, 119, 96, 109, 87, 86, 85, 88, 110, 71, 70, 69, 68, 73, 74, 67, 59, 125, 89, 92, 99, 65, 104, 117]  **Długość trasy**  133970.65   Metoda wspinaczkowa (Hill Climbing)Dla 48 miast Parametry  **Najlepszy wynik** 10628  **Najlepsza trasa**: [34, 29, 2, 26, 4, 35, 45, 10, 24, 42, 5, 48, 39, 32, 21, 47, 20, 33, 46, 36, 30, 43, 17, 27, 19, 37, 6, 28, 7, 18, 44, 31, 38, 8, 1, 9, 40, 15, 12, 11, 13, 25, 14, 23, 3, 22, 16, 41]  **Liczba iteracji w najlepszej próbie**:142 Dla 76 miast Parametry:  **Najlepsza trasa**: [57, 56, 55, 52, 53, 54, 42, 43, 28, 27, 26, 29, 30, 31, 19, 20, 5, 4, 3, 2, 75, 76, 1, 23, 22, 21, 25, 24, 46, 45, 44, 48, 47, 69, 68, 70, 67, 50, 49, 51, 66, 65, 71, 72, 73, 64, 63, 62, 61, 41, 40, 34, 35, 36, 37, 18, 17, 11, 12, 16, 15, 13, 14, 74, 8, 7, 6, 9, 10, 32, 33, 38, 39, 60, 59, 58]  **Długość trasy** : 112250  Liczba iteracji w najlepszej próbie : 255 Dla 127 miast Parametry  **Najlepsza trasa** [44, 54, 57, 35, 30, 31, 27, 26, 33, 122, 28, 97, 98, 123, 95, 40, 41, 14, 12, 20, 21, 72, 8, 9, 11, 3, 61, 62, 60, 59, 104, 125, 89, 92, 99, 65, 113, 66, 55, 124, 52, 5, 121, 51, 2, 105, 15, 108, 4, 22, 17, 79, 80, 32, 29, 25, 38, 39, 42, 34, 43, 36, 37, 16, 1, 7, 120, 10, 100, 64, 58, 91, 90, 116, 67, 73, 74, 68, 75, 69, 70, 71, 110, 85, 86, 87, 88, 109, 96, 119, 63, 102, 101, 83, 82, 126, 81, 84, 117, 78, 76, 77, 18, 19, 23, 24, 6, 106, 114, 13, 115, 50, 56, 47, 53, 49, 118, 48, 46, 94, 112, 111, 107, 127, 93, 103, 45]  **Długość trasy** 132166.43  liczba iteracji wewnątrz algorytmu:478 |

## Excel solver

Excel Solver jest narzędziem służącym do dostosowywania wartości w komórkach zmiennych decyzyjnych tak, aby spełnić limity obejmujące komórki ograniczeń i uzyskać pożądany wynik w komórce celu. Komórką celu była w tym wypadku długość trasy którą staraliśmy się zminimalizować zmieniając kolejność miast. Jako ograniczenia ustawiliśmy, że miasta nie mogą się powtarzać. Do rozwiązania użyliśmy metody ewolucyjnej ponieważ służy ona do rozwiązywania problemów o charakterze niegładkim. Dla każdego zestawu danych solver został wykonany 10 razy, za każdym razem początkowa trasa była ustawiana losowo, a wyniki znajdują się w poniższej tabeli.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Zestawienie wyników | | | |
| Dane | tsp29 | tsp48 | tsp76 | Tsp127 |
| Średnia | 2058,3 | 11324,7 | 120587,1 | 147787,4 |
| Najlepszy wynik | 2028 | 11075 | 117213 | 135644,4 |