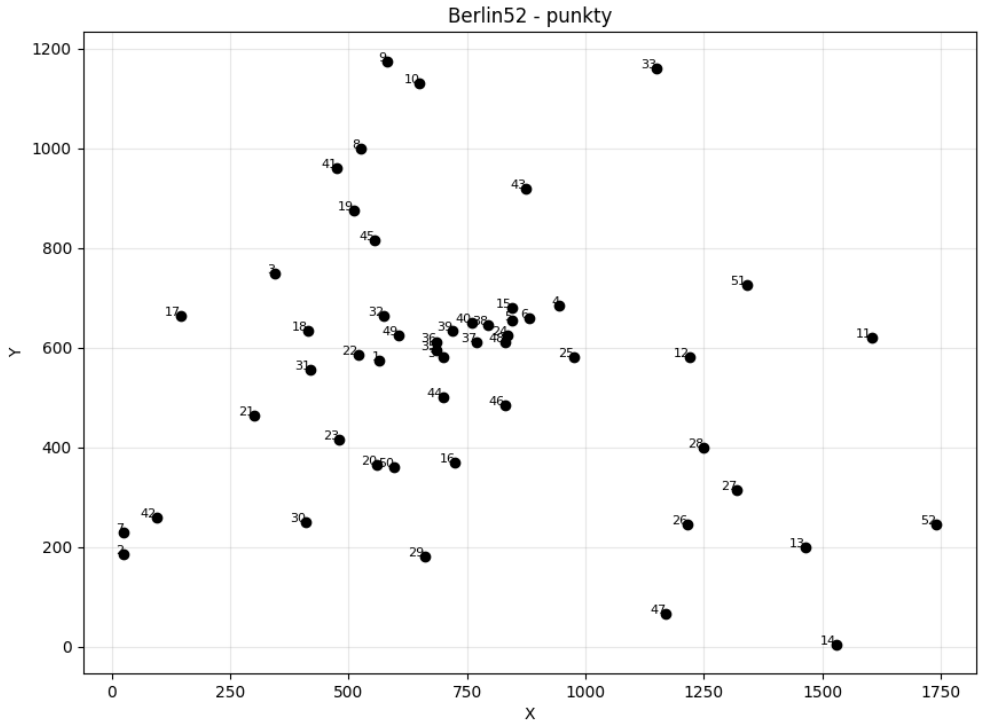
TSP

Aleks Czarnecki 160 190

Algorytm Genetyczny

1. ### Inicjalizacja

   Przykładowa instancja – berlin52



2. ### Opis Algorytmu

Celem algorytmu jest znalezienie najkrótszej drogi przechodzącej przez wszystkie miasta dokładnie jeden raz i powracającej do miasta początkowego.

Algorytm składa się z następujących głównych części:

Inicjalizacja: Generowana jest początkowa populacja tras, 40% tras metodą najbliższego sąsiada, zaczynając od losowego miasta, 60% tras to losowe permutacje miast optymalizowane algorytmem 2-opt.

Selekcja: Rodzice wybierani są poprzez turniejowy mechanizm selekcji z turniejem o rozmiarze 5 osobników. Najlepsze rozwiązanie z populacji (elita) automatycznie przechodzi do następnego pokolenia.

Krzyżowanie: wykorzystywany jest operator EAX (Edge Assembly Crossover) tworzy trasę potomną rozpoczynając od losowego miasta z pierwszego rodzica. Dla aktualnego miasta sprawdzani są jego

bezpośredni sąsiedzi w obu rodzicach, spośród których wybierany jest ten o najmniejszej odległości. Jeśli nie ma dostępnych sąsiadów z rodziców, wybierane jest najbliższe nieodwiedzone miasto. Proces jest powtarzany aż do utworzenia pełnej trasy, która zostaje zamknięta przez dodanie miasta startowego.

Mutacja: wykonywana jest mutacja metodą 2-opt, która zamienia krzyżujące się krawędzie na niekrzyżujące się, jeśli prowadzi to do skrócenia trasy.

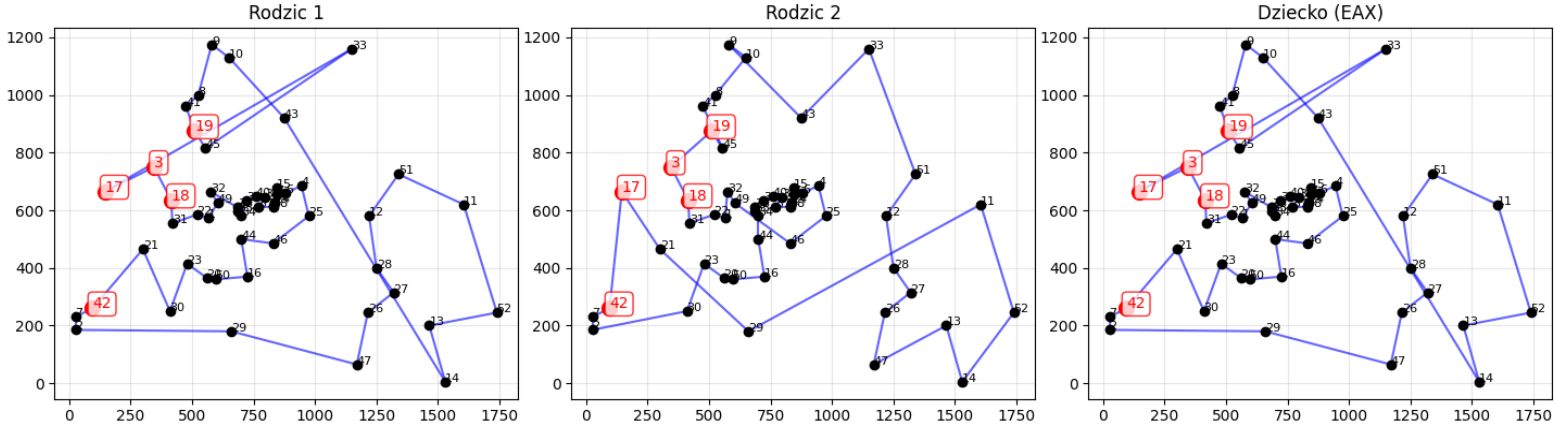
Zapobieganie stagnacji: Po stagnacji trwającej 50 pokoleń połowa populacji jest zastępowana nowymi osobnikami wygenerowanymi tymi samymi metodami co przy inicjalizacji.

3. Pseudokod

```
Funkcja algorytmGenetyczny(liczbaPokolen, rozmiarPopulacji, wspolczynnikMutacji, rozmiarTurnieju):
   miastaMacierz <- Utwórz macierz odległości między wszystkimi miastami
   populacja <- []

   Dla 40% populacji:
       Wybierz losowe miasto startowe z miastaMacierz
       trasa <- Wygeneruj trasę metodą najbliższego sąsiada
       Dodaj trasa do populacji

   Dla pozostałych 60%:
       trasa <- Wygeneruj losową permutację miast
       poprawa2opt(trasa)
       Dodajtrasa  do populacji

   stagnacja <- 0
   najlepszaOdleglosc ← ∞
   najlepszaTrasa ← null

   Dopóki i < liczbaPokolen:
       nowaPopulacja <- []

       elita < -Znajdź najlepszego osobnika z populacja
       Dodaj elita do nowaPopulacja

       Dopóki rozmiar(nowaPopulacja) < rozmiar(populacja):
           rodzic1 <- selekcjaTurniejowa(populacja, rozmiarTurnieju)
           rodzic2 <- selekcjaTurniejowa(populacja, rozmiarTurnieju)
           dziecko <- krzyzowanieEAX(rodzic1, rodzic2, miastaMacierz)

           Z prawdopodobieństwem wspolczynnikMutacji:
               poprawa2opt(dziecko)

           Dodaj dziecko do nowaPopulacja

       populacja <- nowaPopulacja

       trasa <- Znajdź najlepszego osobnika i oblicz jego koszt

       Jeśli trasa < najlepszaTrasa:
           najlepszaTrasa <- trasa
           stagnacja <- 0
       Wpp:
           stagnacja++
       Jeśli stagnacja > 50:
           Wymień połowę populacji na nowe osobniki
           stagnacja <- 0

   Zwróć najlepszaTrasa i jej koszt


Funkcja krzyzowanieEAX(rodzic1, rodzic2, miastaMacierz):
   dziecko <- []
   nieodwiedzone <- lista wszystkich miast

   miasto <- Wybierz losowe miasto z rodzic1
   Dodaj miasto do dziecko
   Usuń miasto z nieodwiedzone
```

```
   Dopóki nieodwiedzone nie jest puste:

       kandydaci <- []

       Dla rodzic w (rodzic1, rodzic2):
           nastepne <- Znajdź następne miasto po obecnym w rodzic
           Jeśli nastepne jest w nieodwiedzone:
               dodaj nastepne do kandydaci

       Jeśli kandydaci nie jest pusty:
           miasto <- Znajdź najbliższego kandydata według miastaMacierz
       Wpp:
           miasto <- Znajdź najbliższe nieodwiedzone miasto

       Dodaj miasto do dziecko
       Usuń miasto z nieodwiedzone

   Dodaj pierwsze miasto na koniec dziecko
   Zwróć dziecko


Funkcja poprawa2opt(trasa, miastaMacierz):
   zmiana <- prawda
   najlepszyKoszt <- koszt(trasa)

   Dopóki zmiana:
       zmiana <- fałsz

       Dla każdej pary miast (i,j) w trasie:
           kosztObecny <- Oblicz koszt krawędzi (i-1,i) + (j,j+1)
           kosztNowy <- Oblicz koszt krawędzi (i-1,j) + (i,j+1)

           Jeśli kosztNowy < kosztObecny:]
               Odwróć fragment trasy od i do j

               kosztCałkowity <- Oblicz całkowity koszt nowej trasy

               Jeśli kosztCałkowity < najlepszyKoszt:
                   najlepszyKoszt <- kosztCałkowity
                   zmiana <- prawda
                   Przerwij pętlę wewnętrzną
               Wpp:
                   Odwróć fragment z powrotem

   Zwróć trasa


Funkcja selekcjaTurniejowa(populacja, rozmiarTurnieju, miastaMacierz):
   turniej <- Wybierz losowo rozmiarTurnieju osobników z populacja
   najlepszy <- pierwszy osobnik z turniej

   Dla osobnik w turniej:
       Jeśli koszt(osobnik) < koszt(najlepszy):
           najlepszy <- osobnik

   Zwróć najlepszy
```

4. **Przykład obrazujący działanie**



Proces decyzyjny wyboru miast do dziecka:

Start z miasta **17** (wybrane losowo)

Krok 1:

W rodzicu 1 następne miasto to **3** (odległość: 217.31)

W rodzicu 2 następne miasto to **42** (odległość: 408.07)

Wybrano miasto **3** z rodzica 1

Krok 2:

W rodzicu 1 następne miasto to **18** (odległość: 134.63)

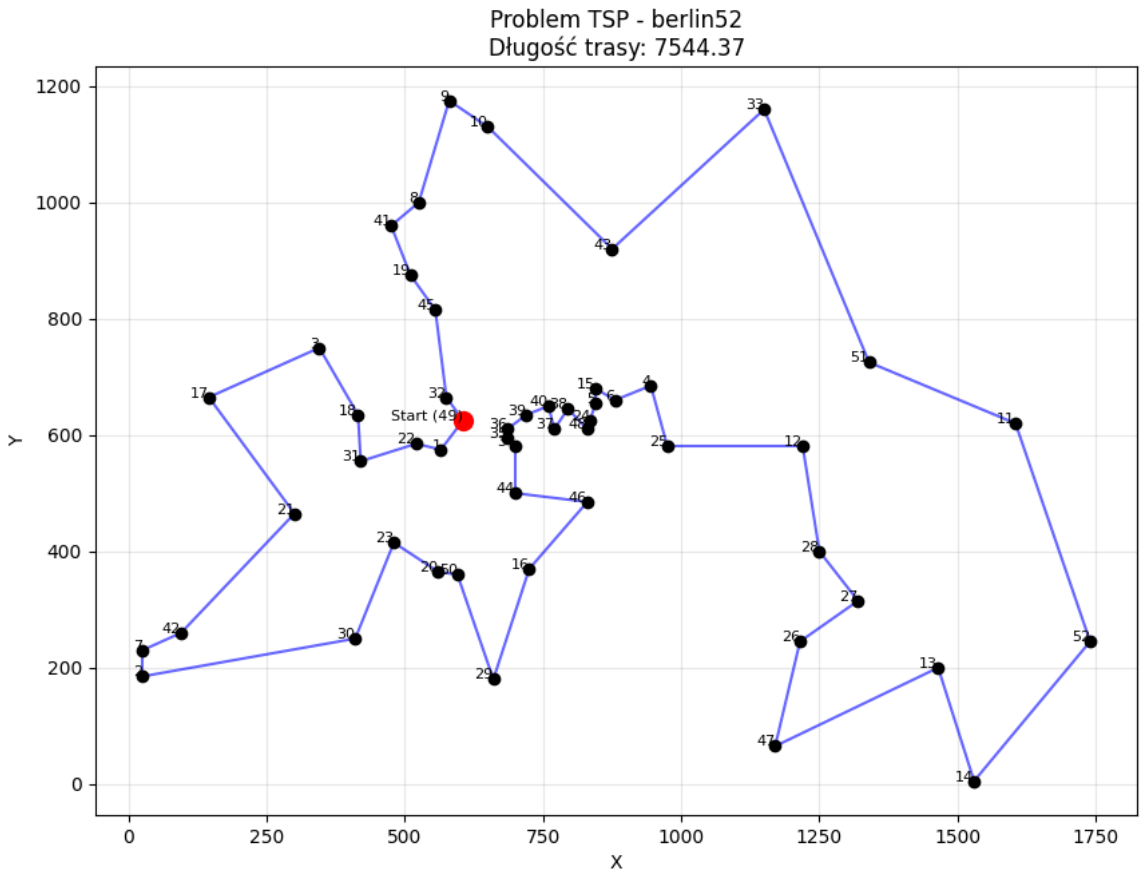
W rodzicu 2 następne miasto to **19** (odległość: 207.00)

Wybrano miasto **18** z rodzica 1

Gdy nie ma dostępnego następnego miasta w obu rodzicach (np. miasto jest ostatnim w trasie lub następne miasta zostały już odwiedzone), algorytm wybiera najbliższe nieodwiedzone miasto spośród wszystkich pozostałych.

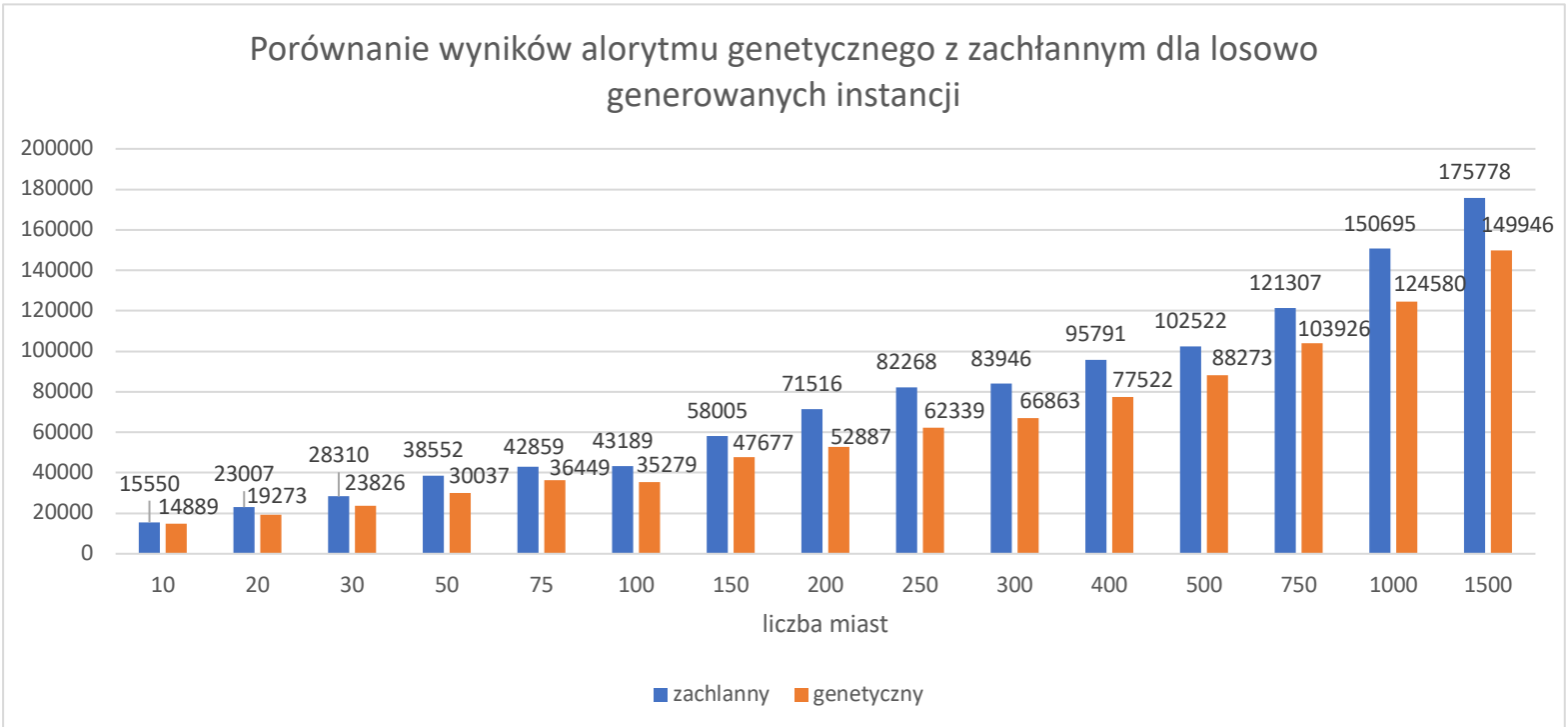
Proces tworzenia dziecka kończy się, gdy wszystkie miasta zostały dokładnie raz odwiedzone. Na końcu trasy dodawane jest miasto startowe, aby zamknąć cykl.

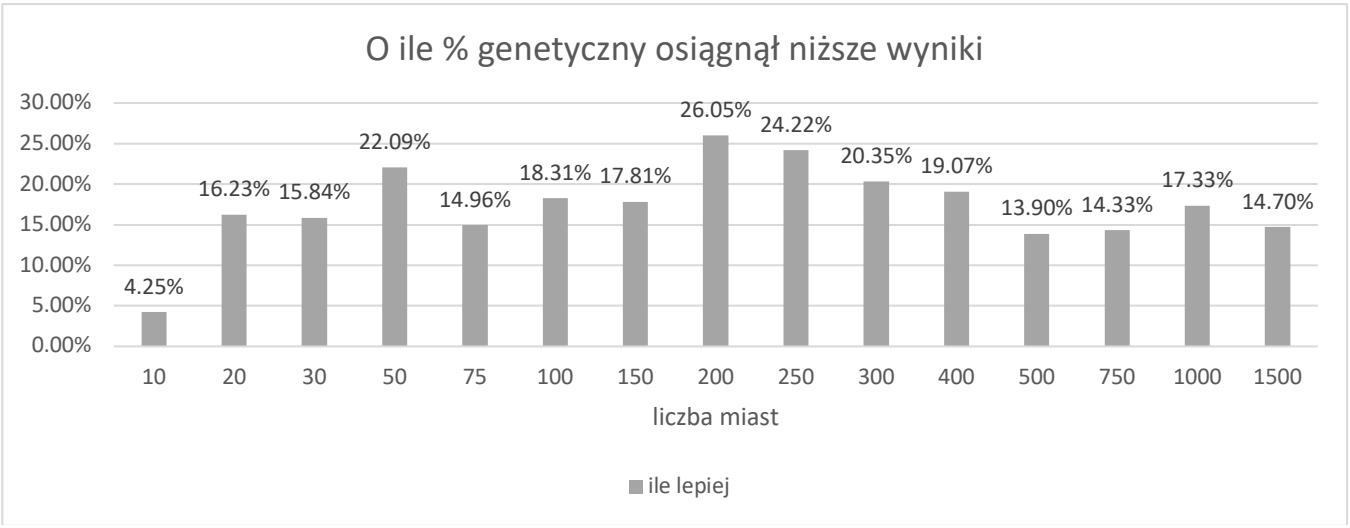
5. **Finalizacja**



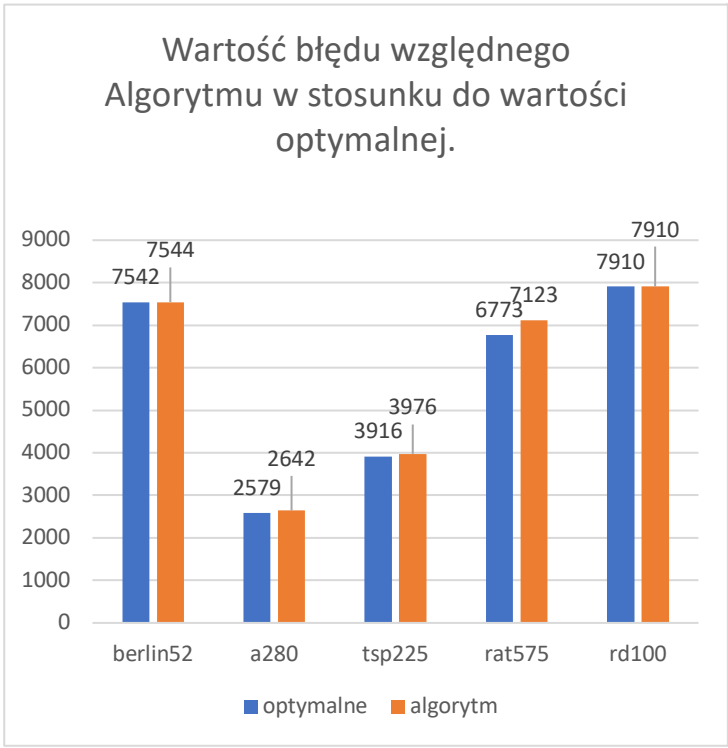
Wykresy

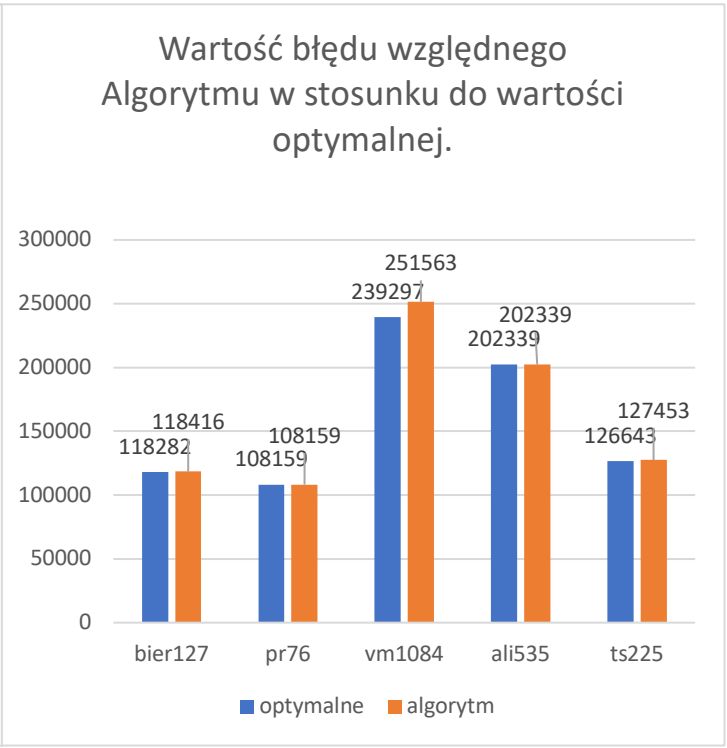
1. Porównaj optymalizowaną wartość (wynik) Algorytmu z A. zachłannym.

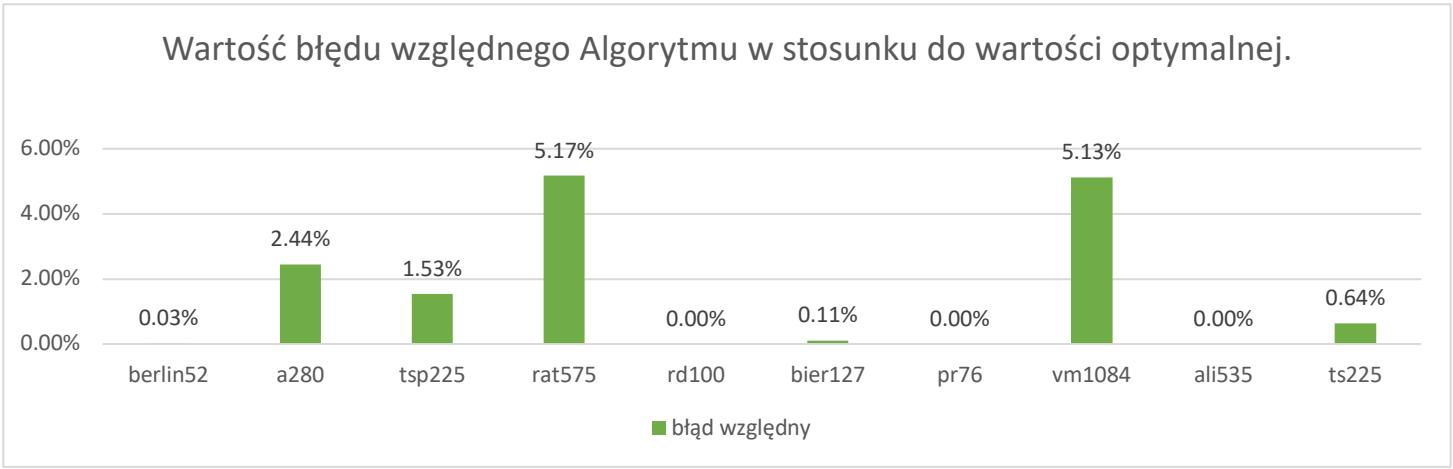




2. Korzystając z bibliotek instancji (benchmarków) pokaż na wykresie wartość błędu względnego Algorytmu w stosunku do wartości optymalnej.







3. Tabela wyników algorytmu dla instancji rankingowych

berlin52	**7544**
bier127	**118416**
tsp250	**12667**
tsp500	**86418**
tsp1000	**24165**