Andrzej Pultyn

325213

WSI – zadanie 2

• Wstęp

Celem zadania było rozwiązanie problemu komiwojażera przy pomocy algorytmu ewolucyjnego. Należy wyznaczyć możliwie najkrótszą pętlę między miastami o podanych koordynatach.

Algorytm został zaimplementowany w klasie TSP, którą inicjujemy, podając listę koordynatów miast. Następnie, używając funkcji solve, możemy uruchomić algorytm ewolucyjny z różnymi parametrami.

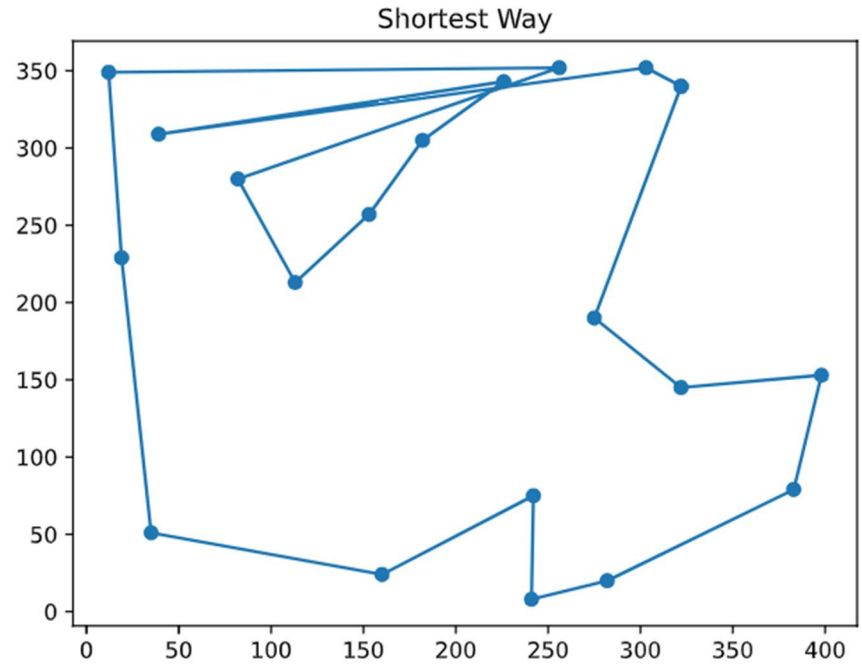
Na początku mamy populację zupełnie losowych ścieżek, wraz z ich długościami. Populacja jest posortowana według długości ścieżek rosnąco. Następnie wybieramy tylu osobników o najkrótszej ścieżce, ilu chcemy rodziców. Następnie ich krzyżujemy, czyli bierzemy część z jednego rodzica i łączymy z częścią z drugiego rodzica. To, jaki wkład ma dany z rodziców można ustawić parametrem alpha. Po utworzeniu takiego potomka, jest szansa (parametr mutate rate), że zmutuje, czyli ileś razy (od 0 do parametru mutate amount) zostaną zamienione ze sobą 2 losowe miasta w ścieżce. Następnie tacy potomkowie dodawani są do populacji, i usunięte zostanie tyle najgorszych (czyli o najdłuższej ścieżce) osobników, ile urodziło się dzieci, żeby liczebność populacji pozostała bez zmian.

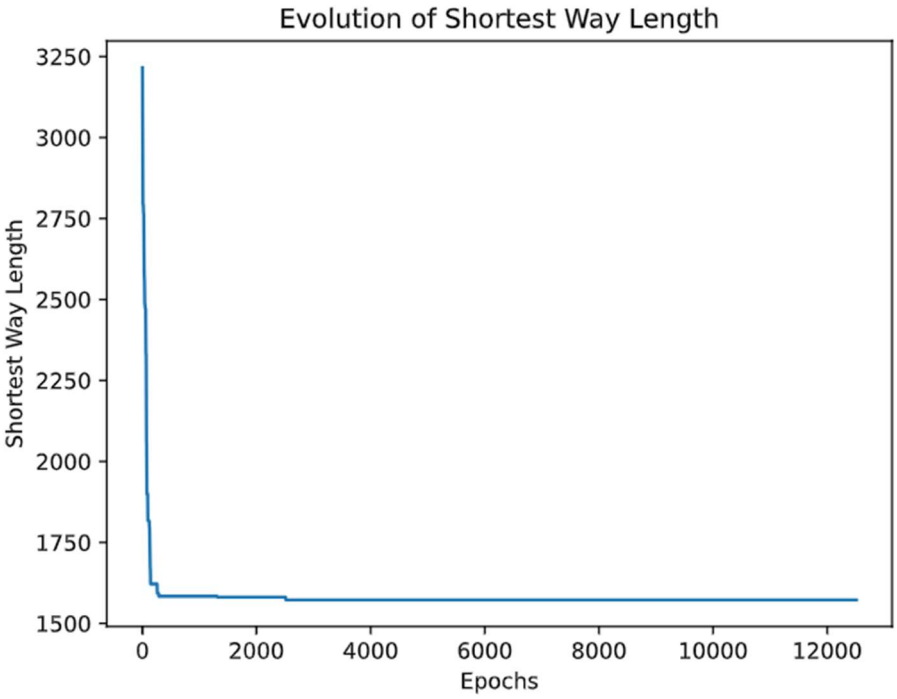
Tak iterujemy, aż nie przekroczymy maksymalnej ilości epok, albo przez określoną ilość epok algorytm nie znajdzie lepszej ścieżki niż obecna najlepsza.

• Przykład działania

Przykładowe rozwiązanie dla miast podanych w treści zadania i parametrach:

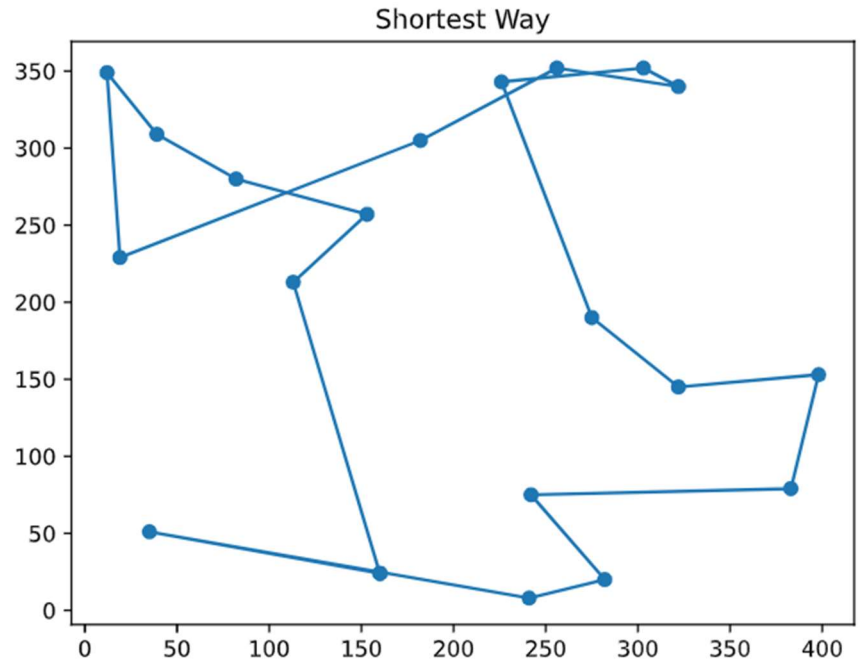
- Maksymalna ilość epok 20000
- Limit epok bez zmian: 10000
- Populacja: 1000
- Ilość rodziców: 400
- Szansa na mutację: 80%
- Siła mutacji (ilość zamienień): 5
- Procent udziału pierwszego rodzica: 50%

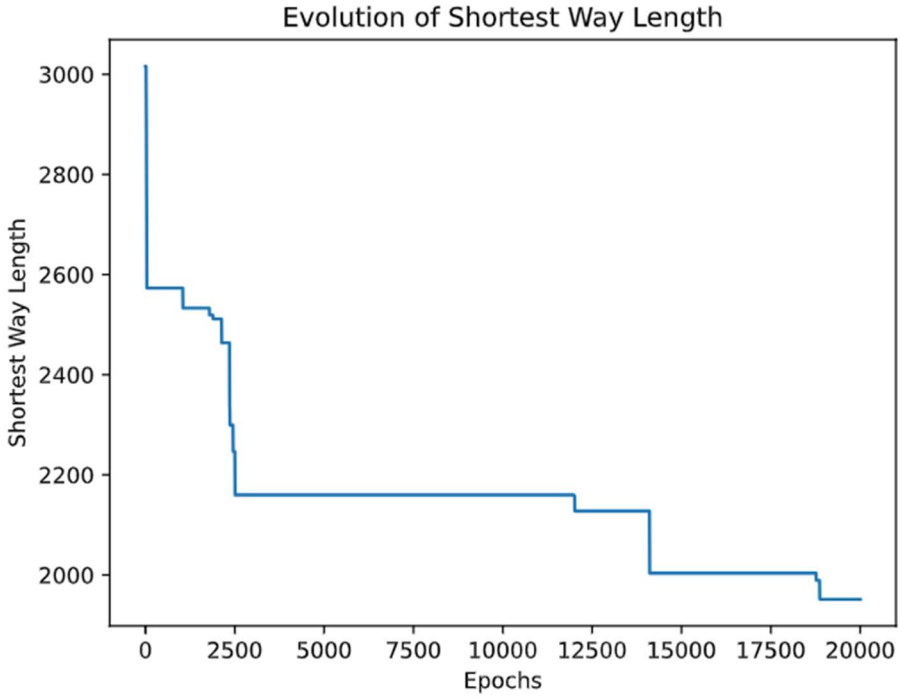




Jak widać, algorytm dosyć szybko skrócił długość trasy z ok. 3200 do ok. 1600. Ostatnia poprawa nastąpiła około w iteracji nr 2500. Później przez 10000 iteracji nie znalazł poprawy, więc algorytm się zatrzymał. Ostateczna trasa może i nie jest idealna, ale na pewno nie jest najgorsza.

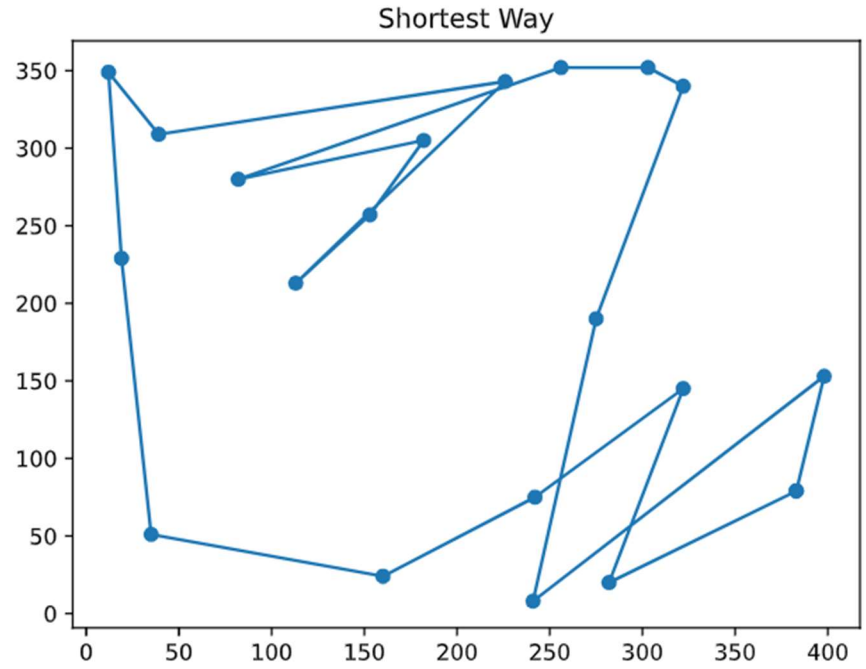
Drugim przykładem jest takie rzadszych mutacji, ale za to silnych (szansa na mutację – 0.1, siła mutacji – 20):

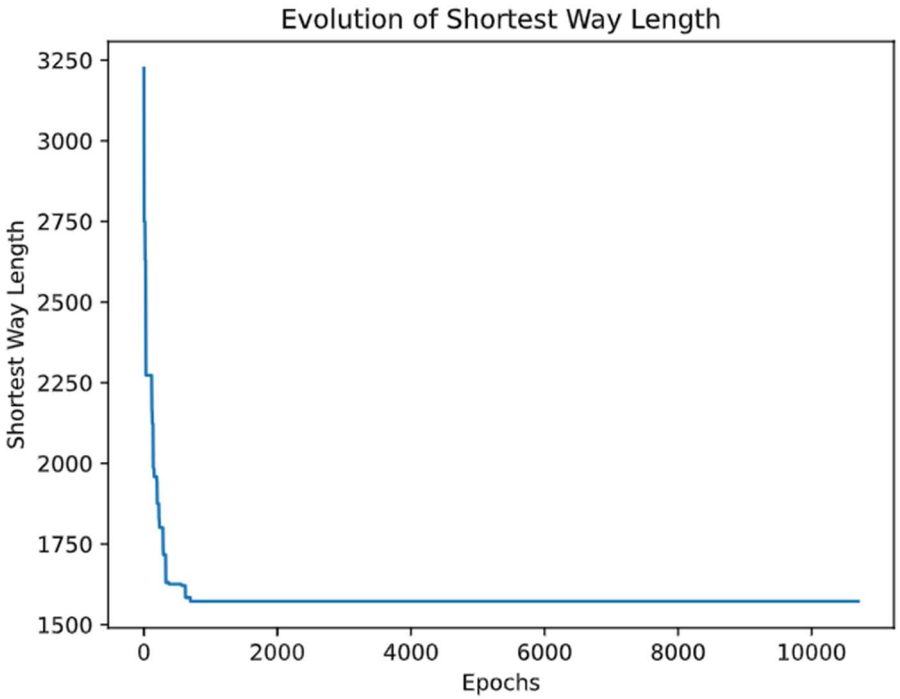




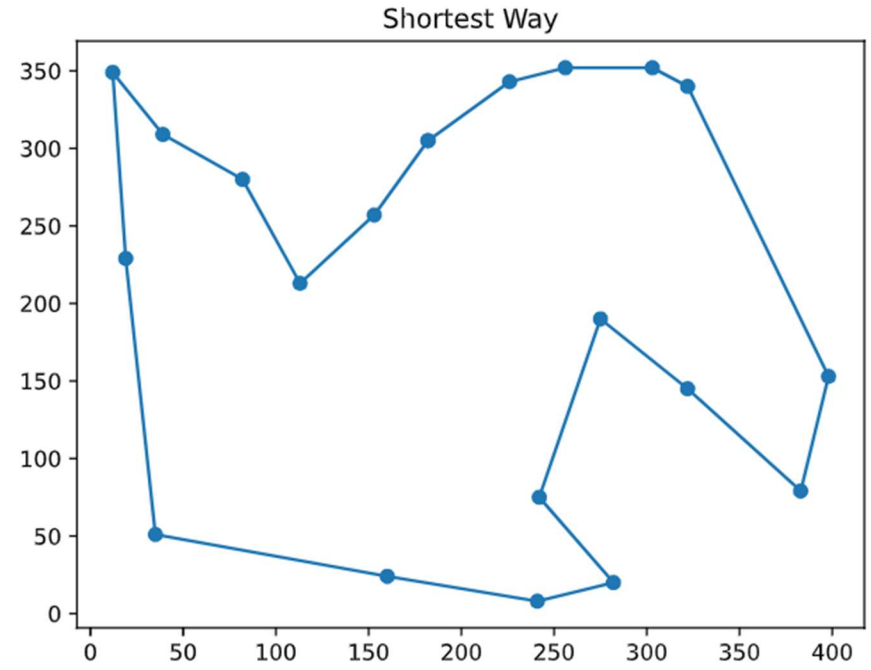
Tutaj algorytm działał o wiele wolniej i uzyskał wynik gorszy, ale za to ciągle się poprawiał. W sytuacji, w której mutacja może nawet wywrócić całą kolejność do góry nogami, jest mniejsza szansa na to, że algorytm się zadowoli jakimś wynikiem.

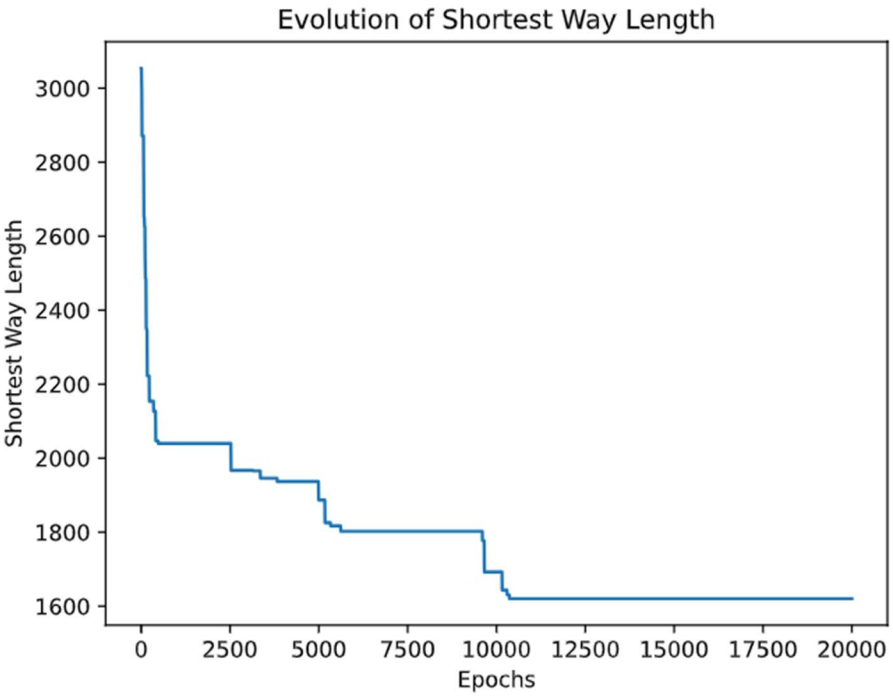
Inne porównanie jakie zrobiłem było porównanie działania dla małej ilości rodziców i dużej:





Duża ilość rodziców (80% populacji)

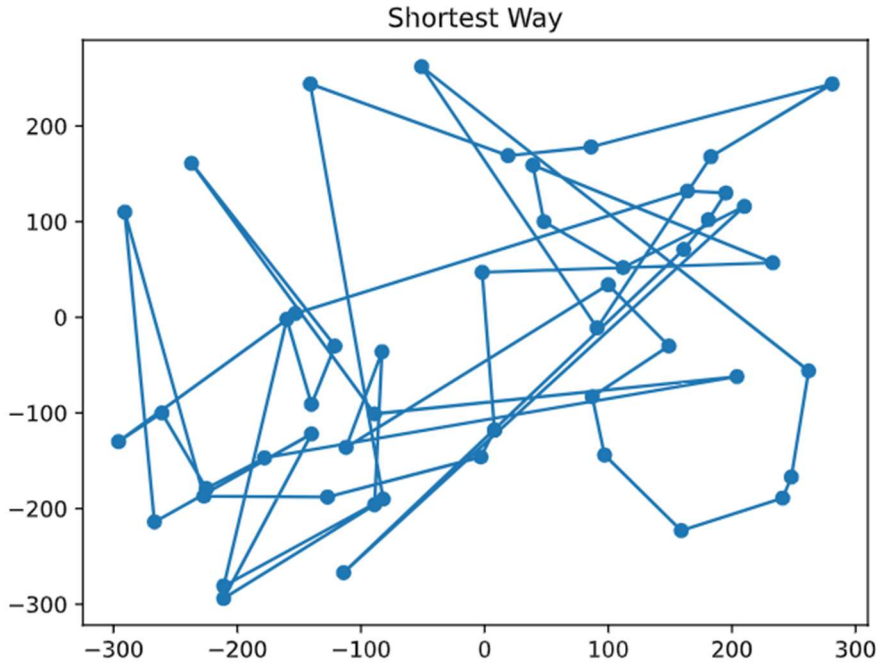


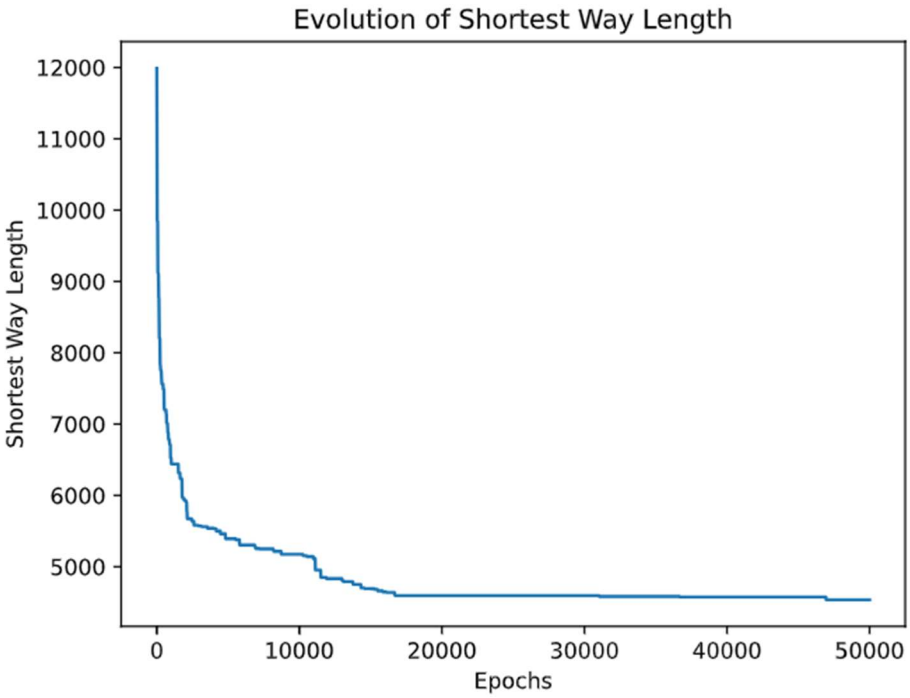


Mała ilość rodziców (20% populacji)

Mała ilość rodziców sprawiła, że znaleziona została najbardziej optymalna trasa, ale zajęło jej to sporo epok. Duża ilość rodziców sprawiła, że szybko znalazł jakieś rozwiązanie, ale się zablokował. Na dodatek każda epoka przy dużej ilości rodziców jest liczona dłużej, ponieważ trzeba wykonać więcej krzyżowań.

Ostatnim testem było zobaczenie, jaką trasę wymyśli dla 50 miast. Wiadomo, że ilość możliwych tras wynosi 50!, co jest niewyobrażalnie wielką liczbą. Po 50000 iteracjach otrzymałem wynik:





Jak widać, algorytm ciągle znajdywał jakieś ulepszenia dotyczące trasy, więc gdyby ustawić mu większą ilość iteracji, to ciągle by się poprawiał. Niestety, taka ilość iteracji przy takiej ilości miast zajmuje ponad 20 minut.

- # Wnioski

Algorytm ewolucyjny jest bardzo losowym algorytmem. Może się okazać, że już w początkowej populacji znajdywać się będzie najbardziej optymalne rozwiązanie. Następnie ilość i rodzaj mutacji może dać albo duże usprawnienie, albo zupełnie złą trasę. Algorytm może się zablokować na danej trasie, a może nastąpić mutacja, która go z tej blokady wydostanie. Ten algorytm jest dobry żeby

wyznaczać w miarę dobre trasy, ale na pewno nie najbardziej optymalne. W większości moich przypadków, gdzie rozwiązywanie problemu zajmowało maksymalnie pojedyncze minuty (poza ostatnią próbą), udało się skrócić pierwotną kompletnie losową trasę o ok. 50%.