Optymalizacja trasy pojazdu

Raport zaliczeniowy z nieklasycznych metod optymalizacji

Antoni Ballaun	Laura Hoang	Jan Wojnowski	Mateusz Leszczyński
ab140447	lh140381	jw140701	ml82786

1 czerwca 2025

1 Wprowadzenie

1.1 Problem biznesowy

Firma dystrybucyjna obsługuje kilkadziesiąt punktów odbioru towarów zlokalizowanych na terenie miasta i okolic. Każdego dnia samochód dostawczy musi załadować towar w magazynie, a następnie odwiedzić wszystkie punkty odbioru, dostarczając zamówione produkty. Pokonana odległość ma bezpośredni wpływ na koszty paliwa oraz czas pracy kierowców, zatem jej optymalizacja istotnie wpływa na wynik finansowy firmy. Kierownik logistyki stoi więc przed konkretnym problemem: w jakiej kolejności odwiedzać kolejne lokalizacje, aby łączna długość trasy była możliwie najmniejsza?

1.2 Ujęcie teoretyczne

Opisany problem stanowi przykład szerszej klasy zagadnień znanych jako *Vehicle Routing Problem* (VRP). Są to problemy decyzyjne związane z tzw. routingiem - znajdowaniem optymalnych tras pojazdów odwiedzających wiele punktów. Optymalizacji może podlegać sumaryczna długość trasy, koszt lub czas potrzebny do jej pokonania. W analizach można uwzględniać różne ograniczenia takie jak np. pojemność ładunkowa lub maksymalny czas pracy kierowców.

Jednym z klasycznych przypadków VRP jest tzw. problem komiwojażera (*Travelling Salesman Problem*, TSP), w którym zakłada się, że jeden pojazd musi odwiedzić raz każdy z zadanych punktów (miast), a następnie wrócić do punktu początkowego, minimalizując całkowitą długość trasy. W ujęciu formalnym celem jest zatem znalezienie minimalnego cyklu Hamiltona w pełnym grafie ważonym odległościami między poszczególnymi wierzchołkami (miastami).

Liczba możliwych tras w TSP rośnie bardzo szybko wraz z liczbą punktów. Dla n miast istnieje dokładnie:

$$\frac{(n-1)!}{2}$$

unikalnych tras (przy założeniu, że kierunek trasy nie ma znaczenia, a trasa zaczyna się i kończy w tym samym miejscu). Oznacza to, że już przy $n = 10$ liczba możliwych tras wynosi 181 440, a przy $n = 15$ przekracza już 40 miliardów. Tak gwałtowny przyrost powoduje, że przy wysokiej liczbie punktów próba przeszukania wszystkich możliwych tras staje się praktycznie niemożliwa nawet dla współczesnych komputerów. Z tego powodu w praktyce stosuje się algorytmy optymalizacyjne i heurystyczne, które pozwalają znaleźć bardzo dobre (choć niekoniecznie optymalne) rozwiązania w rozsądnym czasie obliczeniowym.

2 Metodyka

2.1 Generowanie danych wejściowych

W celu symulacji rzeczywistego problemu TSP wykorzystano kod generujący losowe współrzędne punktów (miast) na płaszczyźnie dwuwymiarowej. Funkcja `generate_coordinates` przyjmuje liczbę miast oraz zakresy współrzędnych i zwraca listę par (x, y) reprezentujących ich położenie.

Następnie, na podstawie wygenerowanych współrzędnych, funkcja `compute_distance_matrix` oblicza macierz odległości między wszystkimi parami punktów, wykorzystując metrykę Euklidesową.

Macierz ta jest elementem koniecznym do przeprowadzenia dalszych obliczeń.

Stałe ziarno generatora losowego (`random.seed(101)`) zapewnia powtarzalność eksperymentu oraz porównywalność wyników uzyskanych kolejnymi metodami.

2.2 Zastosowane algorytmy

W niniejszej pracy zaimplementowano i porównano trzy metody rozwiązywania problemu komiwojażera[1]: algorytm najbliższego sąsiada (Nearest Neighbor), metodę Monte Carlo oraz wyżarzanie symulowane (Simulated Annealing). Poniżej przestawiono poszczególne algorytmy.

2.2.1 Algorytm najbliższego sąsiada (Nearest Neighbor)

Algorytm najbliższego sąsiada jest prostą i zachłanną heurystyką, która w każdym kroku wybiera najbliższe, jeszcze nieodwiedzone miasto, minimalizując tym samym lokalny dystans.

Algorytm przebiega według następujących kroków:

1. Wybór miasta startowego.
2. Znalezienie najbliższego nieodwiedzonego miasta i przejście do niego.
3. Oznaczenie odwiedzonego miasta jako odwiedzone.
4. Powtarzanie kroków 2 i 3, aż wszystkie miasta zostaną odwiedzone.
5. Powrót do miasta startowego.

Największą zaletą tej metody jest szybka implementacja oraz efektywność obliczeniowa – przy dobrej implementacji wymaga ona wykonania około $\frac{n(n-1)}{2}$ obliczeń dystansu, co odpowiada złożoności obliczeniowej w notacji Big-O: $O(n^2)$.

Wadami algorytmu są:

- Nie gwarantuje znalezienia rozwiązania optymalnego – wybiera zawsze najkrótszy możliwy następny krok, nie uwzględniając globalnych warunków problemu.
- Wynik działania algorytmmu jest silnie uzależniony od wyboru miasta startowego. Aby uzyskać lepsze rozwiązanie, należy wielokrotnie uruchamiać algorytm z różnymi punktami startowymi i wybrać najkrótszą spośród uzyskanych tras.
- Brak możliwości oceny, jak blisko optymalnego rozwiązania znajduje się wynik (w przeciwieństwie do innych metod, np. 2-opt).
- Algorytm podejmuje decyzje bez możliwości ich zmiany – raz wybrana ścieżka nie jest korygowana.
- Wynik działania jest wrażliwy na rozkład położenia punktów – dla niektórych zestawów miast metoda sprawdza się dobrze, a dla innych radzi sobie znacznie gorzej.

2.2.2 Algorytm Monte Carlo

Algorytm Monte Carlo to podejście probabilistyczne, które polega na wielokrotnym losowym generowaniu rozwiązań i wyborze najlepszego z nich. W przypadku problemu komiwojażera metoda ta opiera się na losowym tworzeniu permutacji miast i ocenie długości uzyskanych tras.

Działanie algorytmu można opisać w następujących krokach:

1. W każdej iteracji generowana jest losowa permutacja reprezentująca trasę między miastami.
2. Dla wygenerowanej trasy obliczana jest jej całkowita długość (w tym powrót do miasta początkowego).
3. Jeżeli trasa ta jest krótsza od dotychczas najlepszej, zostaje zapamiętana jako aktualne najlepsze rozwiązanie.
4. Po zakończeniu ustalonej liczby iteracji algorytm zwraca najkrótszą znalezioną trasę jako przybliżenie rozwiązania optymalnego.

Poniżej omówiono zalety i wady algorytmu w kontekście TSP.

[1] Wszelkie kody można znaleźć na repozytorium: https://github.com/AntekBall/NMOprojektTSP.

Zalety:

- Algorytm cechuje się bardzo prostą i szybką implementacją.
- Jest niezależny od konkretnej postaci problemu, dzięki czemu można go łatwo zastosować do różnych instancji problemu komiwojażera.
- Dużą zaletą jest również możliwość skalowania – zwiększenie liczby iteracji pozwala poprawić dokładność uzyskiwanych wyników.

Wady:

- Metoda nie zawiera żadnego mechanizmu lokalnej optymalizacji, przez co uzyskane rozwiązania mogą być znacznie gorsze od optymalnych.
- Skuteczność algorytmu silnie zależy od liczby iteracji, przez co w przypadku większej liczby miast konieczne jest przeprowadzenie bardzo wielu prób.
- Algorytm nie korzysta z informacji z poprzednich iteracji, ponieważ każda próba jest całkowicie niezależna, co ogranicza jego efektywność.
- W rezultacie metoda Monte Carlo okazuje się mało wydajna w rozwiązywaniu większych instancji problemu komiwojażera, szczególnie w porównaniu do metod wykorzystujących heurystyki lub optymalizację lokalną.

2.2.3 Algorytm wyżarzania symulowanego (Simulated Annealing)

Wyżarzanie symulowane to stochastyczny algorytm inspirowany procesem fizycznego wyżarzania metali, polegającym na powolnym schładzaniu materiału w celu osiągnięcia stanu minimalnej energii. W kontekście problemu komiwojażera algorytm ten wykorzystuje mechanizm probabilistycznego akceptowania gorszych rozwiązań, co pozwala unikać lokalnych minimów i zbliżyć się do rozwiązania globalnego.

W niniejszej implementacji rozwiązanie reprezentowane jest jako permutacja indeksów miast, określająca kolejność ich odwiedzania. Koszt rozwiązania (długość trasy) obliczany jest na podstawie macierzy odległości.

Działanie algorytmu można opisać w następujących krokach:

1. **Inicjalizacja:**
   Generowana jest losowa trasa, permutacja numerów miast, jako rozwiązanie początkowe. Obliczany jest jej koszt i przyjmowany jako bieżący. Ustalana jest temperatura początkowa T_0, oraz współczynnik chłodzenia $a \in (0, 1)$, który określa szybkość spadku temperatury.
2. **Pętla główna (do `max_iter` iteracji):**
   W każdej iteracji temperatura jest aktualizowana zgodnie z regułą:

   $$T_{k+1} = a \cdot T_k$$

   Generowanych jest `num_cand` kandydatów na rozwiązania sąsiednie.
3. **Generowanie sąsiedztwa:**
   Generowanych jest kilka (np. 50) kandydatów na nowe trasy (tzn. sąsiedztwa). Każdy kandydat powstaje poprzez zamianę miejscami losowo wybranej pary sąsiednich miast (ang. *adjacent swap*) w bieżącej trasie. Obliczany jest koszt (wartość minimalizowanej funkcji celu) nowej trasy.
4. **Kryterium akceptacji (funkcja Metropolisa):**
   Jeśli koszt nowej trasy $f(x')$ jest mniejszy niż bieżącej $f(x)$, rozwiązanie jest akceptowane. W przeciwnym przypadku, zostaje zaakceptowane z prawdopodobieństwem:

   $$A = \min\left(1, \exp\left(-\frac{f(x') - f(x)}{T}\right)\right)$$

5. **Aktualizacja najlepszego rozwiązania:**
   Jeśli nowe rozwiązanie jest najlepsze dotąd, zostaje zapamiętane jako rozwiązanie optymalne. Dodatkowo, zapisywana jest historia tras, kosztów, temperatur i wartości funkcji aktywacji.

6. **Zakończenie:** Po wykonaniu wszystkich iteracji algorytm zwraca najlepszą znalezioną trasę (oraz historię zmian kosztów, temperatur i tras).

Poniżej omówiono zalety i wady algorytmu w kontekście TSP.

Zalety:

- Pozwala na ucieczkę z lokalnych minimów dzięki mechanizmowi probabilistycznej akceptacji gorszych rozwiązań.
- Może znaleźć rozwiązania znacznie lepsze niż algorytmy zachłanne, szczególnie przy odpowiednim doborze parametrów.
- Algorytm jest deterministycznie kontrolowany przez parametry (temperatura początkowa, współczynnik chłodzenia), co umożliwia jego strojenie do konkretnego przypadku.

Wady:

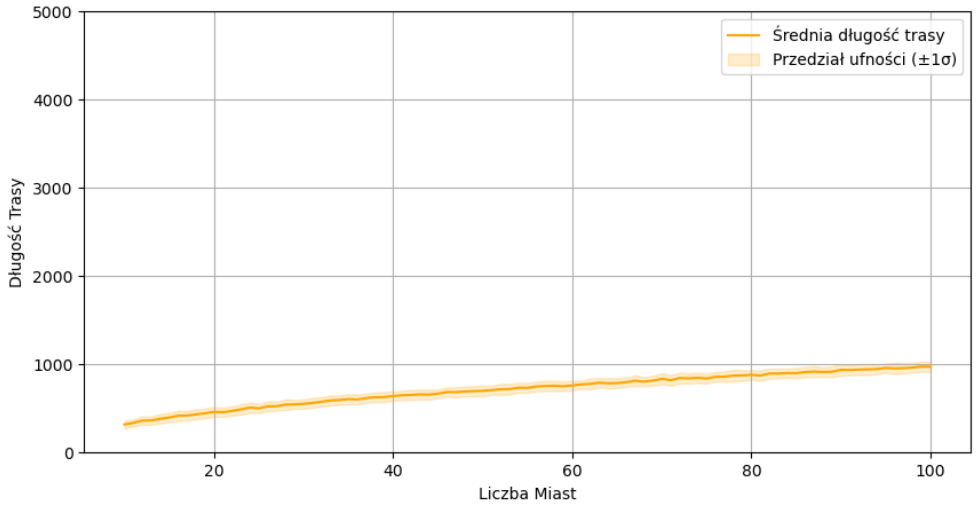
- Wymaga doboru wielu parametrów, co może być trudne i wpływa na jakość rozwiązania.
- Przy złym doborze parametrów może nie dojść do dobrego rozwiązania lub zakończyć działanie zbyt wcześnie.
- Działa wolniej niż proste algorytmy losowe lub zachłanne, zwłaszcza przy dużej liczbie iteracji i kandydatów.
- Nie gwarantuje znalezienia rozwiązania optymalnego.

3 Wyniki

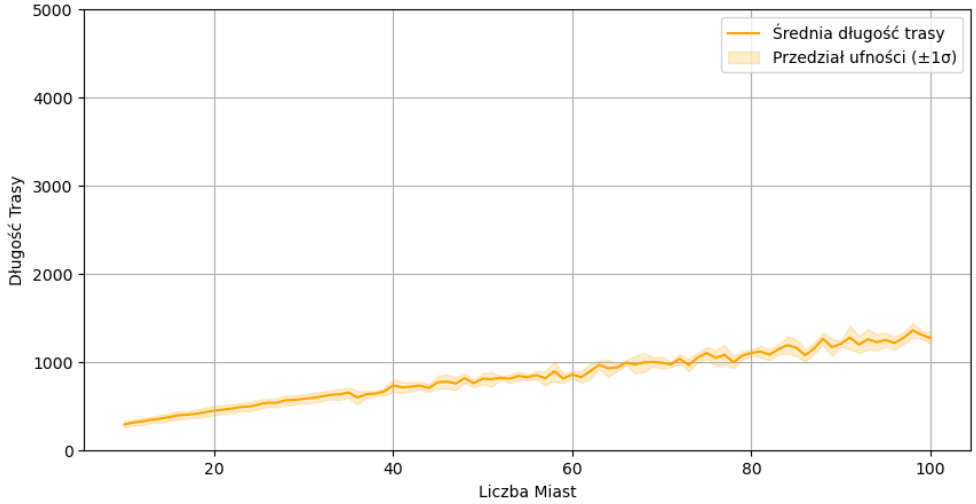
W ramach porównania, rozwiązania zostały przeanalizowane pod względem długości uzyskanej trasy oraz czasu wykonania każdej metody, w zależności od liczby miast.

3.1 Długość trasy

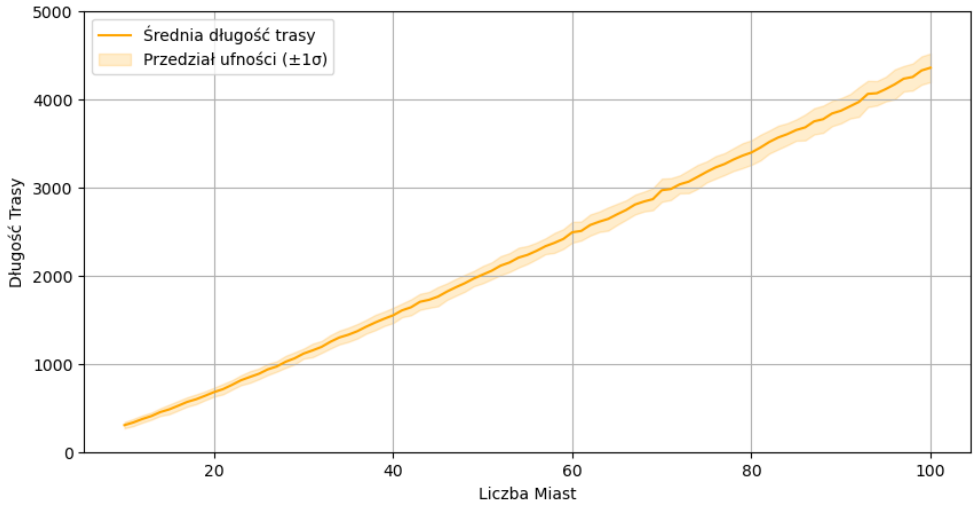
Pod względem jakości optymalizacji trasy, ogólnie najlepsze wyniki uzyskał algorytm Nearest Neighbor, natomiast najgorzej wypadł algorytm Monte Carlo. Simulated Annealing osiągał wyniki zbliżone do Nearest Neighbor, jednak były one nieco słabsze i mniej stabilne.



Rysunek 1: Całkowita Długość Trasy a Liczba Miast dla **Nearest Neighbor**



Rysunek 2: Całkowita Długość Trasy a Liczba Miast dla **Simulated Annealing**



Rysunek 3: Całkowita Długość Trasy a Liczba Miast dla **Monte Carlo**

Dla instancji do 40 miast algorytmy Nearest Neighbor i Simulated Annealing osiągają bardzo zbliżone wyniki. Jednak przy większej liczbie miast, algorytm Nearest Neighbor zaczyna wyraźnie przewyższać Simulated Annealing — dla ponad 80 miast różnica staje się zauważalna: długości tras wyznaczonych przez Nearest Neighbor nie przekraczają 1000, podczas gdy dla Simulated Annealing są już wyraźnie wyższe. Algorytm Monte Carlo odstawał jakościowo już od początku. Nawet przy mniej niż 30 miastach generował trasy o długości około 1000, a przy 100 miastach długości te przekraczały 4000.

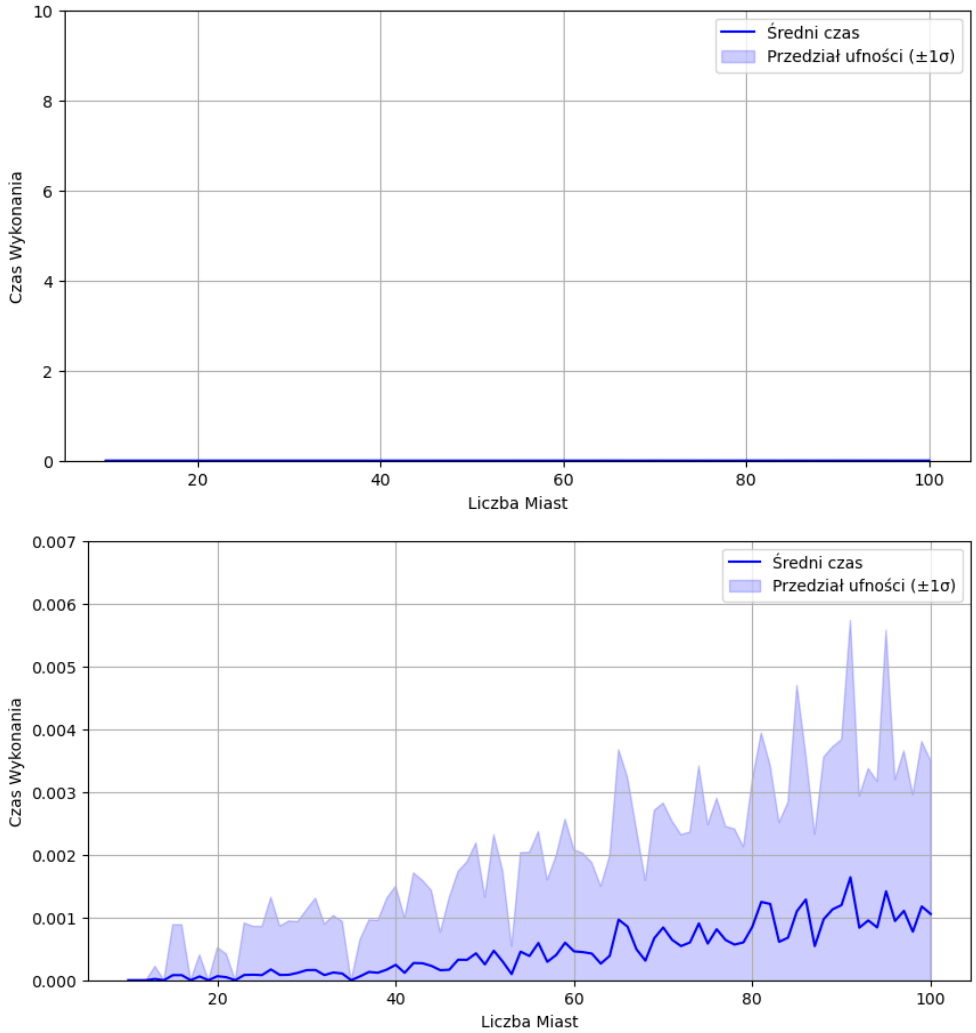
3.2 Czasy wykonania

Każdy z analizowanych algorytmów charakteryzuje się inną złożonością obliczeniową:

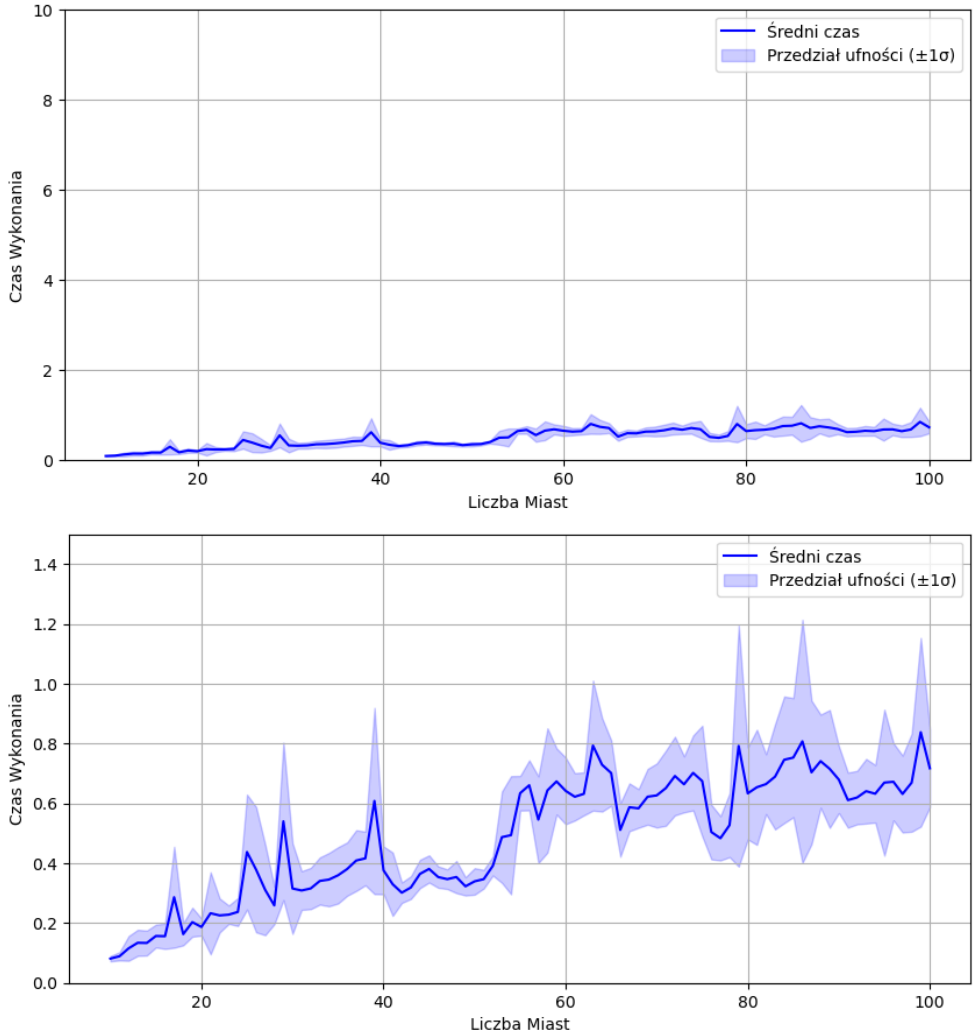
- **Nearest Neighbor (NN)** – deterministyczny, o złożoności w przybliżeniu $\mathcal{O}(n^2)$, gdzie n to liczba miast. Dzięki prostocie działania jest bardzo szybki nawet dla większych instancji.
- **Monte Carlo (MC)** – złożoność zależy głównie od liczby generowanych losowych permutacji. Przy dużej liczbie prób jego czas działania może znacznie wzrosnąć.
- **Simulated Annealing (SA)** – złożoność rośnie proporcjonalnie do liczby iteracji oraz liczby rozważanych sąsiadów. W praktyce złożoność może być zbliżona do $\mathcal{O}(n^2)$ lub wyższa, szczególnie przy dużych instancjach i małym tempie chłodzenia.

Pod względem czasu wykonania, ponownie najlepiej wypadł algorytm Nearest Neighbor — jego średni czas działania nie przekraczał nawet 0.002 sekundy. Drugie miejsce zajął algorytm Monte Carlo, osiągając maksymalny średni czas na poziomie około 0.8 sekundy.

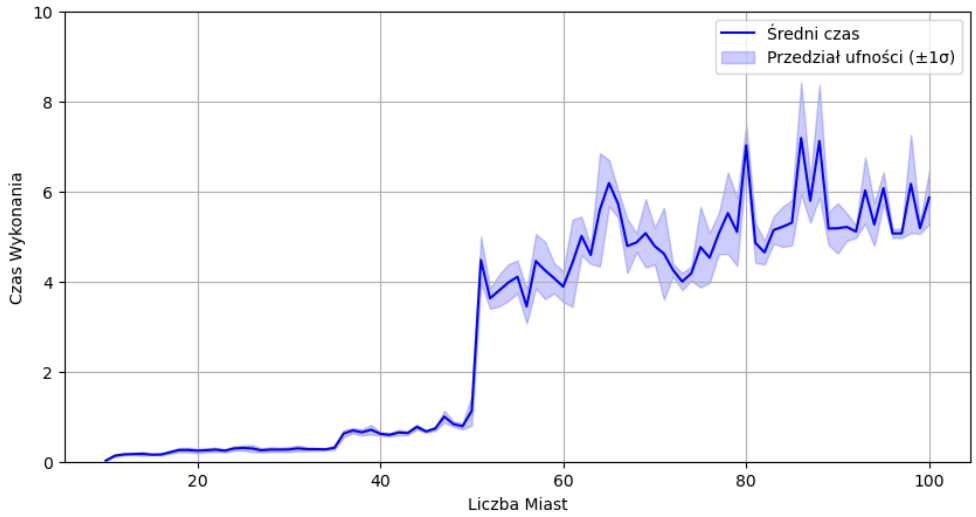
Najbardziej czasochłonnym algorytmem okazał się Simulated Annealing. Dla większych instancji, powyżej 50 miast, czas jego działania dochodził do niemal 7 sekund. Wynika to z dostosowania parametrów algorytmu w celu poprawy jakości rozwiązań, co wyraźnie wpłynęło na wzrost kosztu obliczeniowego.



Rysunek 4: Czas Wykonania a Liczba Miast dla **Nearest Neighbor** (pod: przeskalowany wykres)



Rysunek 5: Czas Wykonania Trasy a Liczba Miast dla **Monte Carlo** (pod: przeskalowany wykres)



Rysunek 6: Czas Wykonania Trasy a Liczba Miast dla **Simulated Annealing**

.

3.3 Symulacje testowe dla wybranych instancji

W celu porównania skuteczności algorytmów przeanalizowano dwa przypadki testowe: trasy obejmujące 25 oraz 50 miast.

Przypadek 1: 25 miast

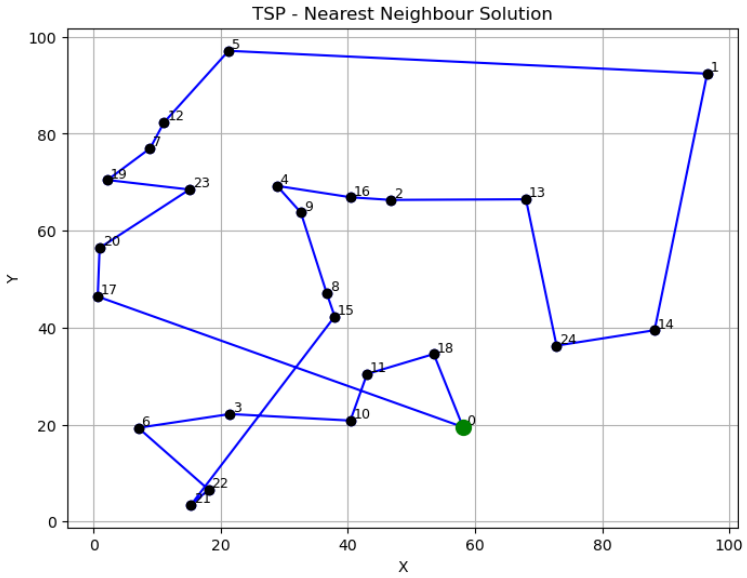
Dla instancji zawierającej 25 miast najlepszy wynik pod względem długości trasy uzyskał algorytm Simulated Annealing (510.93), nieznacznie wyprzedzając Nearest Neighbor (518.31). Algorytm Monte Carlo osiągnął wyraźnie gorszy rezultat (821.43), co świadczy o jego niskiej skuteczności przy ograniczonej liczbie losowań.

Pod względem czasu wykonania, Nearest Neighbor był zdecydowanie najszybszy (praktycznie natychmiastowy czas działania), podczas gdy Simulated Annealing potrzebował około 0.30 sekundy, a Monte Carlo aż 2.96 sekundy.

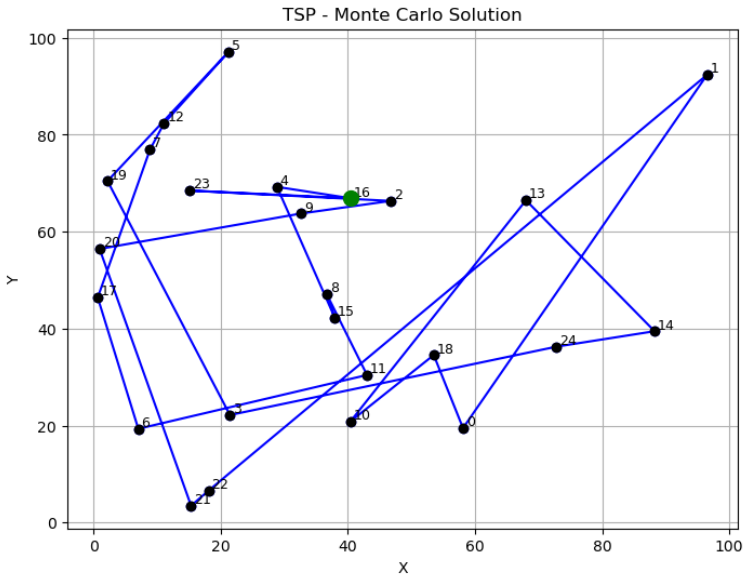
Ostatecznie, Simulated Annealing zapewnił najlepszą jakość rozwiązania, jednak Nearest Neighbor może być korzystniejszym wyborem w zastosowaniach wymagających minimalnego czasu obliczeń. Algorytm Monte Carlo okazał się najmniej efektywny zarówno pod względem jakości trasy, jak i czasu działania.

Algorytm	**Długość trasy**	**Czas wykonania [s]**
Nearest Neighbor	518,31	0,000
Monte Carlo	821,43	2,962
Simulated Annealing	510,93	0,297

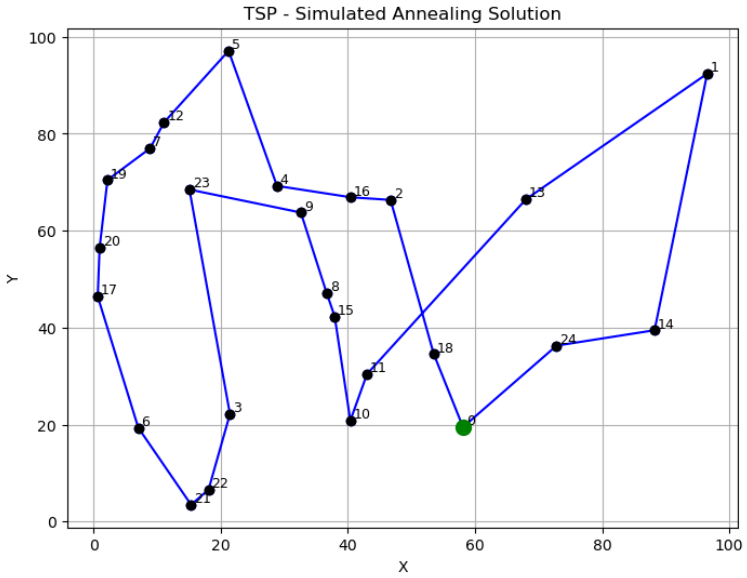
Tabela 1: Porównanie jakości i czasu działania algorytmów dla instancji z 25 miastami



Rysunek 7: Trasy z 25 miastami dla **Nearest Neighbor**



Rysunek 8: Trasy z 25 miastami dla **Monte Carlo**



Rysunek 9: Trasy z 25 miastami dla **Simulated Annealing**

Przypadek 2: 50 miast

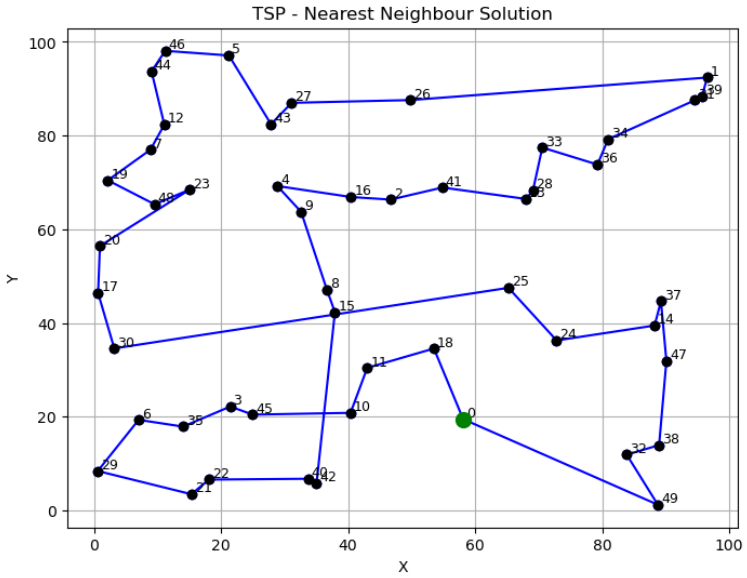
Dla instancji z 50 miastami algorytm Simulated Annealing ponownie uzyskał najkrótszą trasę (510.93), wyraźnie przewyższając jakością zarówno algorytm Nearest Neighbor (641.78), jak i Monte Carlo (821.43).

Pod względem czasu wykonania, Nearest Neighbor nadal pozostaje bezkonkurencyjny, kończąc działanie praktycznie natychmiast. Simulated Annealing, choć wolniejszy (0.30 s), zapewnia istotnie lepsze wyniki optymalizacyjne. Monte Carlo ponownie wypada najsłabiej, zarówno pod względem długości trasy, jak i czasu (2.96 s).

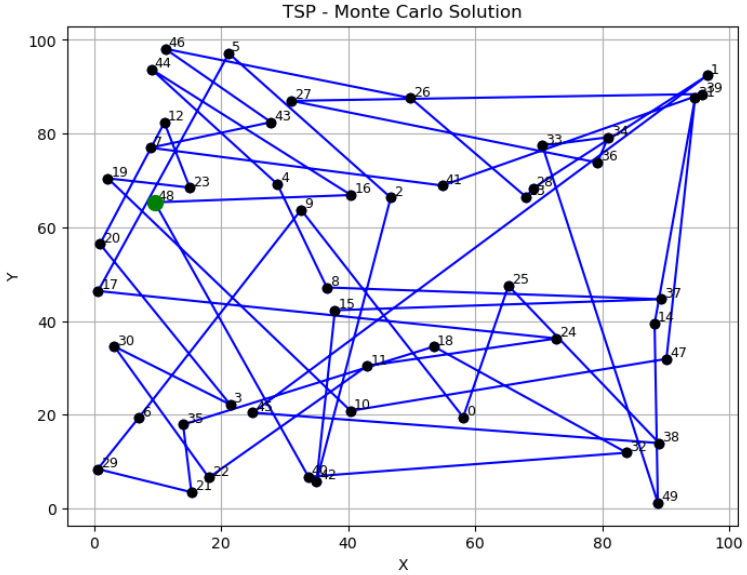
Wnioskując, Simulated Annealing jest najskuteczniejszym algorytmem jakościowo, jednak Nearest Neighbor może być wybierany tam, gdzie czas wykonania ma kluczowe znaczenie, a akceptowalne są rozwiązania gorszej jakości. Monte Carlo, mimo prostoty, okazuje się mało praktyczny przy większych instancjach.

Algorytm	**Długość trasy**	**Czas wykonania [s]**
Nearest Neighbor	641,78	0,000
Monte Carlo	821,43	2,962
Simulated Annealing	510,93	0,297

Tabela 2: Porównanie jakości i czasu działania algorytmów dla instancji z 50 miastami

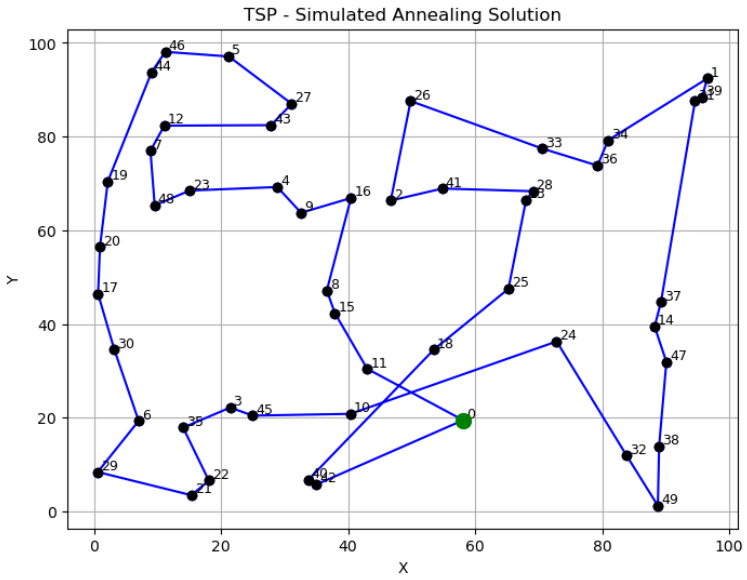


Rysunek 10: Trasy z 50 miastami dla **Nearest Neighbor**



Rysunek 11: Trasy z 50 miastami dla **Monte Carlo**

.



Rysunek 12: Trasy z 50 miastami dla **Simulated Annealing**

4 Wnioski

5 Podsumowanie

Na podstawie przeprowadzonych eksperymentów można stwierdzić, że spośród trzech analizowanych metod - Nearest Neighbor, Monte Carlo oraz Simulated Annealing - to właśnie algorytm najbliższego sąsiada wykazał się najlepszym ogólnym bilansem między jakością a czasem działania. Dla większości instancji NN generował trasy o długościach zbliżonych do tych uzyskiwanych przez SA, przy zdecydowanie niższym czasie wykonania.

Choć w niektórych przypadkach Simulated Annealing uzyskiwał nieco lepsze wyniki jakościowe, to jego skuteczność była silnie uzależniona od doboru parametrów (temperatury początkowej, schematu chłodzenia, liczby iteracji i sąsiedztwa). W obecnej konfiguracji, z ustalonymi parametrami, nie przewyższył jednak jakościowo NN na tyle istotnie, by zrekompensować wielokrotnie dłuższy czas działania.

Monte Carlo okazał się metodą najmniej efektywną, zarówno pod względem jakości rozwiązań, jak i kosztu obliczeniowego. Brak lokalnej optymalizacji oraz losowy charakter generowanych tras skutkowały bardzo niestabilnymi i wyraźnie gorszymi wynikami.

W przyszłych pracach warto rozważyć następujące usprawnienia:

- **Strojenie parametrów SA:** zastosowanie adaptacyjnych strategii chłodzenia lub automatycznej kalibracji parametrów mogłoby znacząco poprawić efektywność tej metody.
- **Zaawansowane operatory sąsiedztwa:** wprowadzenie przekształceń takich jak *2-opt* lub *3-opt* w miejsce prostych zamian dwuelementowych mogłoby umożliwić skuteczniejsze eksplorowanie przestrzeni rozwiązań przez SA.
- **Algorytmy hybrydowe:** połączenie szybkich metod zachłannych (np. Nearest Neighbor) z algorytmami optymalizacji lokalnej (np. Simulated Annealing lub Tabu Search) może dać korzystny kompromis między jakością a czasem działania.

Podsumowując, algorytm Nearest Neighbor — mimo swojej prostoty — ogólnie uzyskał najlepsze wyniki. Jednak przy odpowiednim dostrojeniu, algorytm Simulated Annealing posiada potencjał, by osiągać rozwiązania wyższej jakości, zwłaszcza dla większych i bardziej złożonych instancji problemu komiwojażera.