

Metaheurystyki — zadanie 6

Problem VRPTW — Algorytm mrówkowy z optymalizacją lokalną
GRUPA 3 — piątek 10:15

Bartosz Kołaciński Nikodem Nowak
251554 251598

29 stycznia 2026

Użyte technologie	Python 3.13
Użyte biblioteki	numpy, numba, pandas, matplotlib.pyplot, seaborn

Spis treści

1 Opis problemu VRPTW	4
1.1 Definicja problemu	4
1.2 Instancje benchmarkowe Solomona	4
2 Opis zastosowanej metaheurystyki	5
2.1 Algorytm mrówkowy (ACO)	5
2.1.1 Uzasadnienie wyboru algorytmu	5
2.2 Pseudokod algorytmu ACO dla VRPTW	5
2.3 Parametry algorytmu ACO	6
3 Podejście hybrydowe	7
3.1 Operator Relocate (inter-route)	7
3.2 Algorytm 3-opt (intra-route)	7
4 Adaptacja algorytmu do problemu VRPTW	9
4.1 Uwzględnienie okien czasowych	9
4.2 Definicja funkcji celu	9
4.3 Ocena tras	9
4.4 Dodatkowe ograniczenia	9
5 Kluczowe miejsca implementacji	10
5.1 Obliczanie macierzy heurystyki	10
5.2 Wybór następnego klienta z regułą pseudo-losową	10
5.3 Aktualizacja feromonów z bonusem za mniej pojazdów	11
5.4 3-opt z weryfikacją okien czasowych	11
6 Instrukcja uruchomienia programu	12
6.1 Uruchomienie podstawowe	12
6.2 Dostępne parametry	12
6.3 Wyniki	12
7 Projektowanie eksperymentów	13
7.1 Badane parametry	13
7.2 Metodologia eksperymentów	13
8 Eksperymenty i wyniki	14
8.1 Najlepsze znalezione rozwiązania	14
8.2 Statystyki z wielokrotnych uruchomień	14
8.2.1 Zbiorcze porównanie instancji	14
8.2.2 Instancja C107 (klastrowa)	14
8.2.3 Instancja R106 (losowa)	15
8.2.4 Instancja RC208 (mieszana)	15
8.2.5 Wyniki strojenia parametrów	15
8.3 Wpływ parametrów	15
8.3.1 Wpływ współczynnika α	15
8.3.2 Wpływ współczynnika β	15
8.3.3 Wpływ współczynnika ρ	16
8.3.4 Wpływ progu eksploatacji q_0	16
8.3.5 Wykresy wpływu parametrów	16
8.4 Wizualizacje tras	21
8.4.1 Instancja C107 (klastrowa)	21
8.4.2 Instancja R106 (losowa)	23
8.4.3 Instancja RC208 (mieszana)	25
8.5 Analiza zbieżności i stabilności	26

9 Porównanie z najlepszymi znanymi rozwiązaniami (BKS)	27
10 Analiza wyników	28
10.1 Wpływ parametrów na jakość rozwiązania	28
10.2 Stabilność algorytmu	28
10.3 Trudność instancji	28
10.4 Efektywność optymalizacji lokalnej	28
11 Wnioski	29
11.1 Podsumowanie	29
11.2 Rekomendowane parametry	29
11.3 Ograniczenia rozwiązania	29
11.4 Kierunki dalszej poprawy	30

1 Opis problemu VRPTW

Problem marszrutyzacji pojazdów z oknami czasowymi (ang. *Vehicle Routing Problem with Time Windows*, VRPTW) jest rozszerzeniem klasycznego problemu VRP o dodatkowe ograniczenia czasowe.

1.1 Definicja problemu

Dany jest:

- Magazyn centralny (depot) o współrzędnych (x_0, y_0)
- Zbiór n klientów, każdy z:
 - Lokalizacją (x_i, y_i)
 - Zapotrzebowaniem q_i
 - Oknem czasowym $[e_i, l_i]$ — najwcześniejszy i najpóźniejszy czas rozpoczęcia obsługi
 - Czasem obsługi s_i
- Zbiór identycznych pojazdów o pojemności Q

Cel: Znaleźć zestaw tras dla pojazdów, które:

1. Minimalizują liczbę użytych pojazdów (cel główny)
2. Minimalizują całkowitą przebytą odległość (cel wtórny)

Ograniczenia:

- Każdy klient musi być odwiedzony dokładnie raz
- Suma zapotrzebowania na trasie nie może przekroczyć pojemności pojazdu Q
- Pojazd musi przybyć do klienta przed końcem okna czasowego l_i
- Jeśli pojazd przyjedzie przed początkiem okna e_i , musi czekać
- Wszystkie trasy zaczynają i kończą się w magazynie

1.2 Instancje benchmarkowe Solomona

W ramach zadania wykorzystano trzy instancje z zestawu benchmarkowego Solomona dla 100 klientów:

Instancja	Kategoria	Charakterystyka
R106	R1 (Random)	Klienci rozmieszczeni losowo
C107	C1 (Clustered)	Klienci pogrupowani w klastry
RC208	RC2 (Mixed)	Mieszanka klastrów i rozmieszczenia losowego

2 Opis zastosowanej metaheurystyki

2.1 Algorytm mrówkowy (ACO)

Algorytm mrówkowy (Ant Colony Optimization) to metaheurystyka inspirowana zachowaniem mrówek. Mrówki komunikują się ze sobą za pomocą feromonów, czyli substancji chemicznych pozostawianych na trasie. Ścieżki, które mają więcej feromonów są częściej wybierane, co prowadzi do wytwarzania się częściej obieranych tras.

2.1.1 Uzasadnienie wyboru algorytmu

Algorytm mrówkowy został wybrany ze względu na:

- **Konstrukcyjny charakter** — ACO naturalnie buduje rozwiązania krok po kroku, co dobrze pasuje do problemu VRPTW, gdzie trasy są konstruowane przez sekwencyjne dodawanie klientów
- **Elastyczność** — łatwo można włączyć ograniczenia VRPTW (okna czasowe, pojemność) do procesu konstrukcji rozwiązania
- **Równoległa eksploatacja** — wiele mrówek jednocześnie przeszukuje przestrzeń rozwiązań

2.2 Pseudokod algorytmu ACO dla VRPTW

Algorithm 1 Algorytm ACO-VRPTW

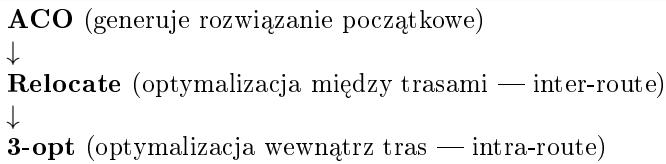
```
1: Inicjalizacja:
2: Oblicz macierz odległości  $d_{ij}$  dla wszystkich par węzłów
3: Oblicz macierz heurystyki  $\eta_{ij} = f(d_{ij}, TW_j, q_j)$ 
4: Zainicjuj macierz feromonów  $\tau_{ij} \leftarrow \tau_0$ 
5: Znajdź rozwiązanie początkowe metodą najbliższego sąsiada
6: for  $t = 1$  do  $T$  (liczba iteracji) do
7:   for każda mrówka  $k = 1, \dots, m$  do
8:     Zainicjuj pustą trasę,  $current\_node \leftarrow depot$ 
9:     while istnieją nieodwiedzeni klienci do
10:      Znajdź zbiór  $\mathcal{F}$  dozwolonych klientów (spełniających TW i pojemność)
11:      if  $\mathcal{F} = \emptyset$  then
12:        Wróć do depot, rozpoczęj nową trasę
13:      else
14:        Wybierz klienta  $j$  z prawdopodobieństwem:
15:         $p_{ij} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{F}} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}$ 
16:        (z regułą pseudo-losową, parametr  $q_0$ )
17:        Zaktualizuj czas, obciążenie pojazdu
18:      end if
19:    end while
20:    Oceń rozwiązanie mrówki
21:  end for
22:  Aktualizacja feromonów:
23:  Wyparowanie:  $\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}$ 
24:  Depozyt: wzmacnij ścieżki najlepszego rozwiązania
25:  Zastosuj ograniczenia  $[\tau_{min}, \tau_{max}]$ 
26: end for
27: return najlepsza znaleziona trasa
```

2.3 Parametry algorytmu ACO

Parametr	Opis
m (n_ants)	Liczba mrówek w kolonii
T (n_iterations)	Liczba iteracji algorytmu
α	Wpływ feromonów na wybór ścieżki
β	Wpływ heurystyki (odległości, okien czasowych) na wybór
ρ	Współczynnik wyparowywania feromonów ($0 < \rho < 1$)
q_0	Próg eksploracji — prawdopodobieństwo wyboru zachłannego

3 Podejście hybrydowe

Zaimplementowane rozwiązanie jest **hybrydą** algorytmu mrówkowego z technikami optymalizacji lokalnej. Struktura hybrydy jest następująca:



3.1 Operator Relocate (inter-route)

Operator Relocate przenosi klientów między trasami w celu:

- Redukcji liczby pojazdów (np. opróżnienie trasy)
- Zmniejszenia całkowitej odległości

Algorithm 2 Operator Relocate

```
1: repeat
2:    $best\_move \leftarrow None$ 
3:   for każda trasa  $R_1$  do
4:     for każdy klient  $c$  w trasie  $R_1$  do
5:       for każda inna trasa  $R_2$  (z wystarczającą pojemnością) do
6:         for każda pozycja  $p$  w trasie  $R_2$  do
7:           if wstawienie  $c$  na pozycję  $p$  spełnia okna czasowe then
8:              $\Delta \leftarrow koszt\_usunięcia(c, R_1) + koszt\_wstawienia(c, R_2, p)$ 
9:             if  $\Delta < best\_move.\Delta$  then
10:                $best\_move \leftarrow (c, R_1, R_2, p, \Delta)$ 
11:             end if
12:           end if
13:         end for
14:       end for
15:     end for
16:   end for
17:   if  $best\_move \neq None$  and  $best\_move.\Delta < 0$  then
18:     Wykonaj  $best\_move$ 
19:   end if
20: until brak poprawy
```

3.2 Algorytm 3-opt (intra-route)

Algorytm 3-opt usuwa 3 krawędzie z trasy i łączy segmenty na nowo, rozważając wszystkie możliwe rekombinacje. Pozwala to na bardziej radykalne zmiany niż 2-opt.

Modyfikacja dla VRPTW:

- Po każdej modyfikacji sprawdzana jest wykonalność okien czasowych
- Akceptowane są tylko ruchy, które nie naruszają ograniczeń czasowych
- Priorytet ma wykonalność nad poprawą dystansu

Algorithm 3 3-opt z uwzględnieniem okien czasowych

```
1: repeat
2:   best_move  $\leftarrow$  None
3:   for i = 1 to n - 2 do
4:     for j = i + 1 to n - 1 do
5:       for k = j + 1 to n do
6:         for type = 1 to 7 do
7:           new_route  $\leftarrow$  zrekombinuj(route, i, j, k, type)
8:            $\Delta \leftarrow$  oblicz_zmianę_dystansu()
9:           if  $\Delta < 0$  and sprawdź_okna_czasowe(new_route) then
10:             if  $\Delta < \text{best\_move}.\Delta$  then
11:               best_move  $\leftarrow$  (i, j, k, type,  $\Delta$ )
12:             end if
13:           end if
14:         end for
15:       end for
16:     end for
17:   end for
18:   if best_move  $\neq$  None then
19:     Zastosuj best_move
20:   end if
21: until brak poprawy
```

4 Adaptacja algorytmu do problemu VRPTW

4.1 Uwzględnienie okien czasowych

Okna czasowe są uwzględniane w kilku miejscach algorytmu:

1. Konstrukcja rozwiązania (ACO):

- Przed dodaniem klienta do trasy sprawdzane jest, czy pojazd zdąży dotrzeć przed końcem okna czasowego
- Klienci, do których nie można dotrzeć na czas, są wykluczani ze zbioru dozwolonych \mathcal{F}

2. Macierz heurystyki:

- Heurystyka uwzględnia szerokość okna czasowego — węższe okna mają wyższy priorytet
- Formuła: $\eta_{ij} = \frac{1}{d_{ij}} \cdot (1 + 0.3 \cdot tw_component + 0.7 \cdot demand_component)$

3. Optymalizacja lokalna:

- Operator Relocate sprawdza wykonalność wstawienia w każdą pozycję
- 3-opt akceptuje tylko ruchy zachowujące wykonalność

4.2 Definicja funkcji celu

Funkcja celu jest hierarchiczna — najpierw minimalizowana jest liczba pojazdów, potem dystans:

$$f(S) = n_{vehicles}^2 \cdot PENALTY + \sum_{r \in Routes} distance(r) \quad (1)$$

gdzie $PENALTY = 100000$ jest dużą stałą zapewniającą priorytet minimalizacji liczby pojazdów.

4.3 Ocena tras

Każda trasa jest oceniana pod kątem:

- **Wykonalności** — czy wszystkie okna czasowe są spełnione
- **Dystansu** — suma odległości między kolejnymi klientami
- **Czasu** — czas przybycia, oczekiwania i obsługi

Trasy niewykonalne otrzymują koszt ∞ i nie są akceptowane.

4.4 Dodatkowe ograniczenia

- **Limit pojazdów:** Maksymalna liczba tras jest ograniczona przez parametr instancji
- **Pojemność:** Sprawdzana przy każdym dodaniu klienta do trasy
- **Czekanie:** Jeśli pojazd przyjedzie przed otwarciem okna, musi czekać (nie jest to naruszenie), ale wiąże się ze zwiększeniem kosztu rozwiązania

5 Kluczowe miejsca implementacji

5.1 Obliczanie macierzy heurystyki

```
1 @njit(cache=True)
2 def compute_heuristic_matrix(
3     distance_matrix, window_starts, window_ends, demands, capacity
4 ):
5     n_nodes = len(distance_matrix)
6     eta = np.zeros((n_nodes, n_nodes), dtype=np.float64)
7
8     for i in range(n_nodes):
9         for j in range(n_nodes):
10            if i != j and distance_matrix[i, j] > 0:
11                # Bazowa heurystyka: odwrotnosc odleglosci
12                dist_component = 1.0 / distance_matrix[i, j]
13
14                # Pilosc okna czasowego: preferuj wezsze okna
15                tw_width = window_ends[j] - window_starts[j]
16                tw_component = 1.0 / (tw_width + 1.0)
17
18                # Komponent zapotrzebowania: preferuj wieksze zamowienia
19                demand_component = demands[j] / capacity
20
21                eta[i, j] = dist_component * (
22                    1.0 + 0.3 * tw_component + 0.7 * demand_component
23                )
24
25    return eta
```

Kod 1: Macierz heurystyki uwzględniająca okna czasowe i zapotrzebowanie

5.2 Wybór następnego klienta z regułą pseudo-losową

```
1 @njit(cache=True)
2 def select_next_customer(
3     current_node, feasible, n_feasible, pheromone, eta,
4     alpha, beta, q0, rand_val, demands, current_load, capacity
5 ):
6     # Oblicz atrakcyjnosc kazdego dozwolonego klienta
7     attractiveness = np.zeros(n_feasible, dtype=np.float64)
8
9     for idx in range(n_feasible):
10        customer = feasible[idx]
11        tau = pheromone[current_node, customer]
12        eta_val = eta[current_node, customer]
13
14        # Bonus za lepsze wypelnienie pozostalej pojemnosci
15        remaining_capacity = capacity - current_load
16        demand_ratio = demands[customer] / remaining_capacity
17        capacity_bonus = 1.0 + 0.5 * min(demand_ratio, 1.0)
18
19        attractiveness[idx] = (tau ** alpha) * (eta_val ** beta) * capacity_bonus
20
21    # Eksploracja vs eksploracja
22    if rand_val < q0:
23        # Eksploracja: wybierz najlepszego
24        return feasible[np.argmax(attractiveness)]
25    else:
26        # Eksploracja: selekcja ruletkowa
27        probabilities = attractiveness / np.sum(attractiveness)
28        # ... (selekcja ruletka)
```

Kod 2: Selekcja klienta z uwzględnieniem pojemności

5.3 Aktualizacja feromonów z bonusem za mniej pojazdów

```
1 @njit(cache=True)
2 def update_pheromones(
3     pheromone, best_routes_flat, best_route_lengths,
4     best_n_routes, best_cost, rho, min_pheromone, max_pheromone
5 ):
6     n_nodes = pheromone.shape[0]
7
8     # Wyparowywanie
9     for i in range(n_nodes):
10         for j in range(n_nodes):
11             pheromone[i, j] *= (1.0 - rho)
12
13     # Depozyt na najlepszym rozwiązaniu
14     if best_cost < np.inf and best_n_routes > 0:
15         # Dodatkowa nagroda za mniej pojazdów
16         vehicle_bonus = 1.0 / (best_n_routes ** 2)
17         delta_tau = vehicle_bonus * 10000.0
18
19         # Wzmocnij krawędzie najlepszego rozwiązania
20         for edge in best_solution_edges:
21             pheromone[from_node, to_node] += delta_tau
22             pheromone[to_node, from_node] += delta_tau # Symetryczne
23
24     # Ogranicz poziomy feromonów
25     np.clip(pheromone, min_pheromone, max_pheromone, out=pheromone)
26
```

Kod 3: Aktualizacja feromonów premiująca mniejszą liczbę tras

5.4 3-opt z weryfikacją okien czasowych

```
1 @njit(cache=True)
2 def _check_route_feasibility(
3     route, distance_matrix, service_times,
4     window_starts, window_ends, speed_factor=1.0
5 ):
6     """Sprawdz czy trasa spełnia wszystkie okna czasowe."""
7     current_time = 0.0
8     for p in range(1, len(route)):
9         from_node = route[p - 1]
10        to_node = route[p]
11        travel_time = distance_matrix[from_node, to_node] * speed_factor
12        current_time += travel_time
13
14        if current_time < window_starts[to_node]:
15            current_time = window_starts[to_node] # Czekanie
16        elif current_time > window_ends[to_node]:
17            return False # Za późno!
18
19        current_time += service_times[to_node]
20    return True
21
```

Kod 4: Sprawdzanie wykonalności trasy

6 Instrukcja uruchomienia programu

6.1 Uruchomienie podstawowe

Program uruchamia się z katalogu `zadanie6/` poprzez:

```
python run_multiple_and_visualize.py <instancja> [opcje]
```

Dostępne opcje można wyświetlić za pomocą:

```
python run_multiple_and_visualize.py -help
```

Przykłady:

```
# Uruchom dla instancji c107.txt, 50 razy
python run_multiple_and_visualize.py c107.txt --runs 50

# Z dostrojonymi parametrami
python run_multiple_and_visualize.py r106.txt --runs 30 \
    --alpha 1.5 --beta 3.5 --rho 0.15 --q0 0.85

# Bez optymalizacji lokalnej (tylko ACO)
python run_multiple_and_visualize.py rc208.txt --no-local-search
```

6.2 Dostępne parametry

Opcja	Domyślna	Opis
<code>-runs N</code>	10	Liczba uruchomień algorytmu
<code>-n_ants N</code>	200	Liczba mrówek w kolonii
<code>-n_iterations N</code>	100	Liczba iteracji ACO
<code>-alpha FLOAT</code>	1.0	Wpływ feromonów
<code>-beta FLOAT</code>	3.0	Wpływ heurystyki
<code>-rho FLOAT</code>	0.1	Współczynnik wyparowywania
<code>-q0 FLOAT</code>	0.9	Próg eksploracji
<code>-output DIR</code>	results/	Katalog wyjściowy
<code>-no-local-search</code>	(wyłączone)	Wyłącz optymalizację lokalną

6.3 Wyniki

Program generuje:

- Wizualizacje rozwiązań (PNG) z trasami w różnych kolorach — najlepsze, najgorsze i średnie rozwiązanie
- Plik CSV ze statystykami wszystkich uruchomień
- Podsumowanie na konsoli z najlepszym, najgorszym i średnim wynikiem

7 Projektowanie eksperymentów

7.1 Badane parametry

Parametr	Zakres wartości	Uzasadnienie
α	0.5, 1.0, 1.5, 2.0	Kontroluje równowagę między feromonią a heurystyką
β	2.0, 3.0, 5.0, 7.0	Wyższe wartości preferują bliższe/pilniejsze węzły
ρ	0.05, 0.1, 0.2, 0.3	Szybkość zapominania starych ścieżek
q_0	0.7, 0.8, 0.9, 0.95	Równowaga eksplatacja/eksploracja
n_ants	50, 100, 200	Większy rój = lepsza eksploracja
n_iterations	50, 100, 200	Więcej iteracji = więcej czasu na zbieżność

7.2 Metodologia eksperymentów

Dla każdego zestawu danych:

1. Algorytm uruchamiany **50 razy** dla wybranej konfiguracji parametrów
2. Zapisywane metryki:
 - Liczba pojazdów (cel główny)
 - Całkowity dystans
 - Czas obliczeń
3. Obliczane statystyki:
 - Najlepszy wynik
 - Najgorszy wynik
 - Średnia
 - Odchylenie standardowe

8 Eksperymenty i wyniki

Przeprowadzono eksperymenty na trzech instancjach benchmarkowych Solomona. Dla każdej instancji wykonano strojenie parametrów (750 konfiguracji, każda po 3 razy) oraz 50 uruchomień z najlepszymi znalezionymi parametrami.

8.1 Najlepsze znalezione rozwiązania

Poniższa tabela przedstawia najlepsze wyniki uzyskane podczas 50 uruchomień z optymalnymi parametrami:

Instancja	Pojazdy	Dystans	Parametry
C107	11	871.25	$\alpha=0.5, \beta=5.0, \rho=0.2, q_0=0.85$
R106	17	1438.65	$\alpha=0.5, \beta=3.0, \rho=0.3, q_0=0.75$
RC208	3	978.47	$\alpha=0.5, \beta=4.5, \rho=0.3, q_0=0.85$

Obserwacje:

- Dla wszystkich instancji optymalny $\alpha = 0.5$, co sugeruje mniejszy wpływ feromonów na rzecz heurystyki
- Wartości β w zakresie 3.0–5.0 okazały się optymalne, przy czym dla klastrowych instancji (C107) wyższe wartości dają lepsze wyniki
- Instancja RC208 osiągnęła optymalną liczbę pojazdów (3)
- Dla instancji C107 osiągnięto wynik z 11 pojazdami — tylko 1 więcej niż optimum BKS (10)
- Dla instancji R106 najlepsza konfiguracja ($q_0=0.75$) zapewnia dobrą równowagę eksploatacji i eksploracji

8.2 Statystyki z wielokrotnych uruchomień

Dla każdej instancji wykonano 50 uruchomień z najlepszymi parametrami znalezionymi podczas strojenia.

8.2.1 Zbiorcze porównanie instancji

Instancja	Liczba pojazdów				Dystans			
	Min	Max	Śr.	Std	Min	Max	Śr.	Std
C107	11	14	12.42	0.81	871.25	1207.64	1045.41	84.32
R106	17	21	18.76	1.02	1438.65	1689.85	1557.31	55.97
RC208	3	4	3.98	0.14	978.47	1081.96	986.37	42.03

8.2.2 Instancja C107 (klastrowa)

Parametry: $\alpha=0.5, \beta=5.0, \rho=0.2, q_0=0.85$

- Algorytm wykazuje dużą stabilność — odchylenie standardowe dystansu to tylko 8.1% średniej
- Najlepszy wynik (871.25) osiągnięto z 11 pojazdami — tylko 1 pojazd więcej niż optymalne BKS (10)
- Rozkład pojazdów: 11 (10%), 12 (54%), 13 (28%), 14 (8%)

8.2.3 Instancja R106 (losowa)

Parametry: $\alpha=0.5$, $\beta=3.0$, $\rho=0.3$, $q_0=0.75$

- Instancja najtrudniejsza — rozproszone rozmieszczenie klientów utrudnia optymalizację
- Najlepszy wynik: 17 pojazdów, dystans 1438.65 (uruchomienie 13)
- Rozkład pojazdów: 17 (6%), 18 (32%), 19 (44%), 20 (12%), 21 (6%)
- Odchylenie standardowe dystansu 3.6% średniej

8.2.4 Instancja RC208 (mieszana)

Parametry: $\alpha=0.5$, $\beta=4.5$, $\rho=0.3$, $q_0=0.85$

- Szerokie okna czasowe umożliwiają znaczną redukcję liczby pojazdów
- Najlepszy wynik: 3 pojazdy, dystans 978.47 (uruchomienie 31) — **optymalna liczba pojazdów**
- Rozkład pojazdów: 3 (2%), 4 (98%) — bardzo stabilna liczba pojazdów
- Najniższe odchylenie standardowe dystansu (4.3% średniej)

8.2.5 Wyniki strojenia parametrów

Podczas strojenia przetestowano 750 konfiguracji parametrów dla każdej instancji:

Instancja	Min poj.	Max poj.	Śr. poj.	Min dyst.	Czas tuningu
C107	11	16	13.36	987.94	1.6h
R106	17	23	20.66	1667.90	1.6h
RC208	3	4	3.99	1031.98	1.9h

8.3 Wpływ parametrów

Na podstawie analizy 750 konfiguracji dla instancji C107 zbadano wpływ poszczególnych parametrów.

8.3.1 Wpływ współczynnika α

α	Min pojazdy	Śr. pojazdy	Min dystans	Śr. dystans
0.5	11	12.49	987.94	1225.17
1.0	11	13.34	1081.83	1402.52
1.5	12	13.63	1210.21	1500.42
2.0	12	13.69	1232.54	1522.65
2.5	12	13.65	1244.96	1520.42

Wniosek: Niskie wartości α (0.5) dają najlepsze wyniki. Mniejszy wpływ feromonów pozwala na większą eksplorację przestrzeni rozwiązań.

8.3.2 Wpływ współczynnika β

β	Min pojazdy	Śr. pojazdy	Min dystans	Śr. dystans
2.5	12	13.50	1071.09	1484.57
3.0	11	13.43	1040.65	1462.65
3.5	11	13.44	1034.74	1441.33
4.0	12	13.24	987.94	1420.82
4.5	11	13.25	1091.47	1409.59
5.0	11	13.30	1020.71	1386.44

Wniosek: Wartości β w zakresie 3.5–5.0 dają najlepsze wyniki. Wyższe β poprawia średnią i prowadzi do stabilniejszych rozwiązań.

8.3.3 Wpływ współczynnika ρ

ρ	Min pojazdy	Śr. pojazdy	Min dystans	Śr. dystans
0.05	12	13.50	1130.05	1456.62
0.10	12	13.49	987.94	1435.38
0.15	11	13.29	1028.27	1423.18
0.20	11	13.29	1020.71	1427.31
0.25	11	13.23	1073.13	1428.68

Wniosek: Wartości ρ w zakresie 0.10–0.20 dają najlepsze wyniki. Umiarkowane wyparowywanie feromonów zapewnia dobrą równowagę między eksploracją a eksploatacją.

8.3.4 Wpływ progu eksploatacji q_0

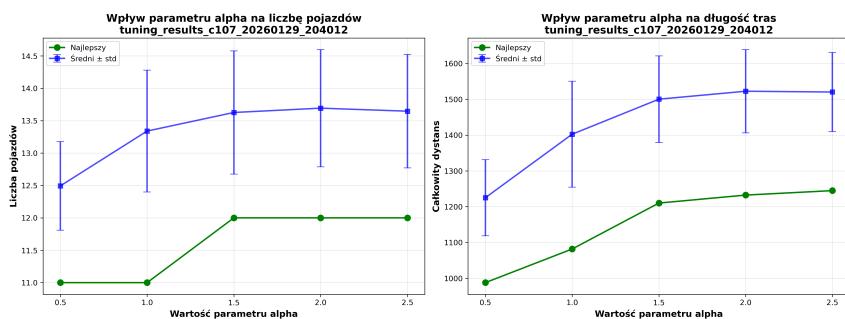
q_0	Min pojazdy	Śr. pojazdy	Min dystans	Śr. dystans
0.75	11	12.62	1040.65	1350.13
0.80	11	12.95	1083.05	1398.26
0.85	11	13.25	1020.71	1411.38
0.90	11	13.63	987.94	1465.74
0.95	12	14.35	1170.24	1545.67

Wniosek: Wartości q_0 w zakresie 0.80–0.90 dają najlepsze wyniki pod względem minimalnego dystansu. Zbyt wysoka eksploatacja ($q_0 > 0.9$) prowadzi do przedwczesnej zbieżności, natomiast zbyt niska zwiększa średnią liczbę pojazdów.

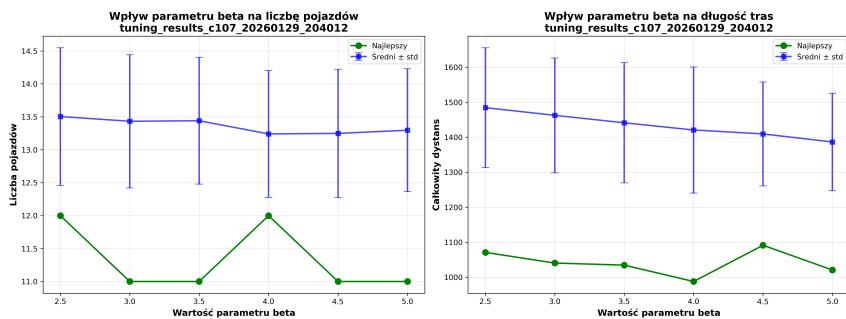
8.3.5 Wykresy wpływu parametrów

Poniższe wykresy przedstawiają zależność jakości rozwiązania (liczba pojazdów i dystans) od wartości poszczególnych parametrów.

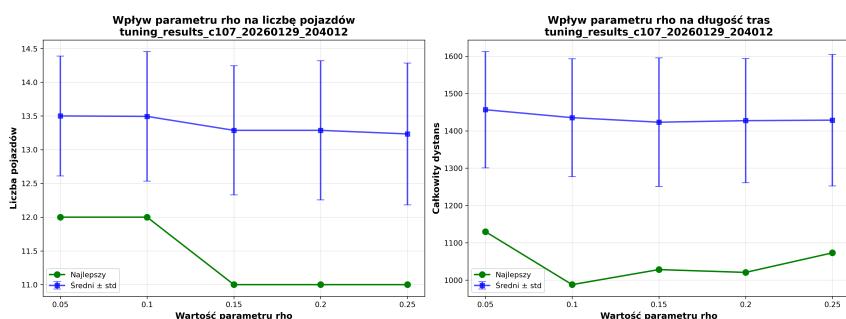
Instancja C107 (klastrowa)



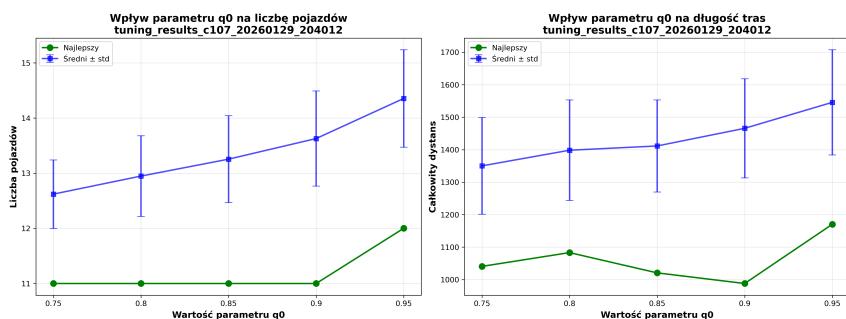
Rysunek 1: C107: Wpływ parametru α — niższe wartości dają lepsze wyniki



Rysunek 2: C107: Wpływ parametru β — optymalne wartości w zakresie 3.5–5.0

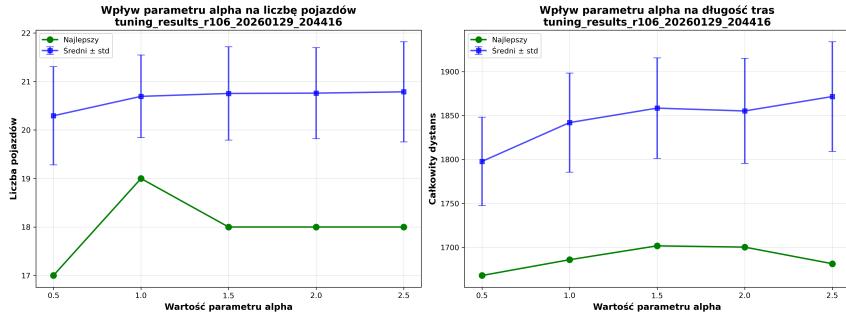


Rysunek 3: C107: Wpływ parametru ρ — umiarkowane wartości stabilizują poszukiwanie

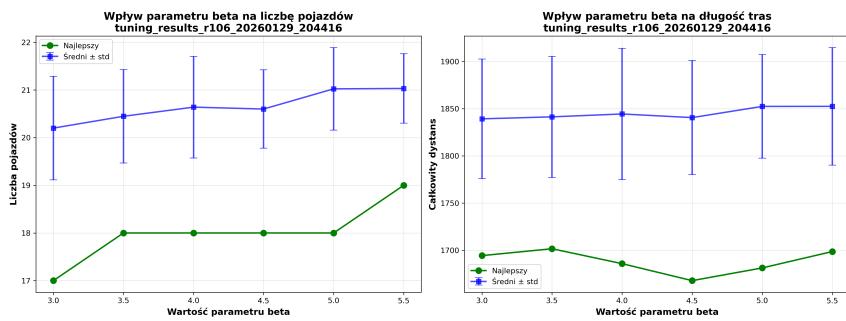


Rysunek 4: C107: Wpływ parametru q_0 — wartości 0.80–0.90 zapewniają lepszą eksplorację

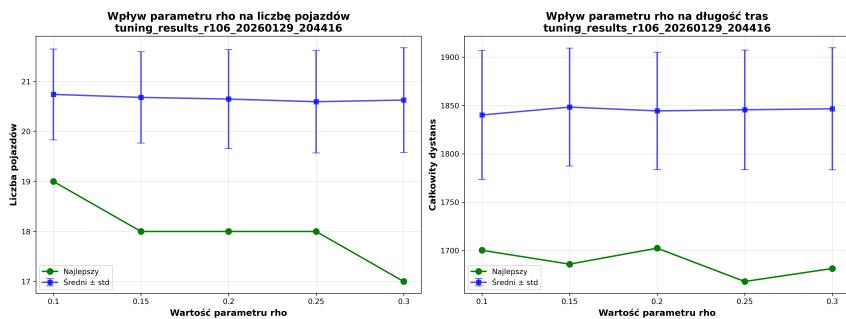
Instancja R106 (losowa)



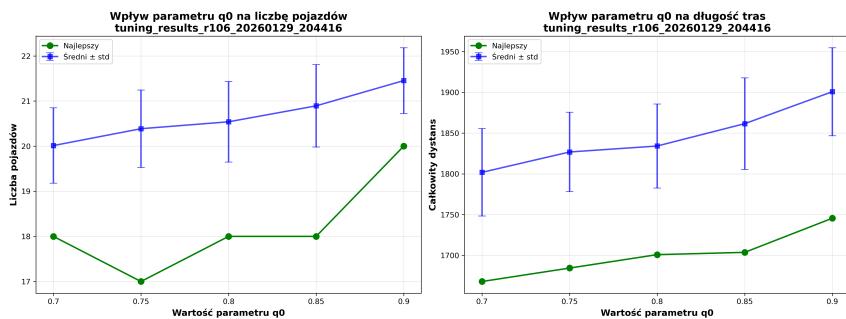
Rysunek 5: R106: Wpływ parametru α — podobnie jak dla C107, niższe wartości są lepsze



Rysunek 6: R106: Wpływ parametru β — optymalne wartości 3.0–4.5

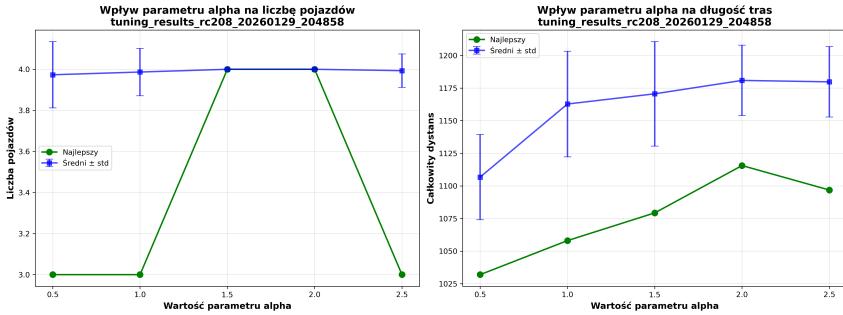


Rysunek 7: R106: Wpływ parametru ρ — dla instancji losowej wyższe ρ (0.25–0.30) daje lepsze wyniki

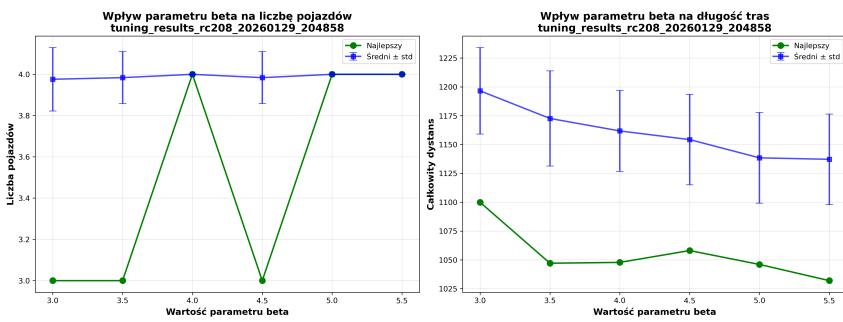


Rysunek 8: R106: Wpływ parametru q_0 — niższe wartości (0.70–0.75) są kluczowe dla trudnych instancji

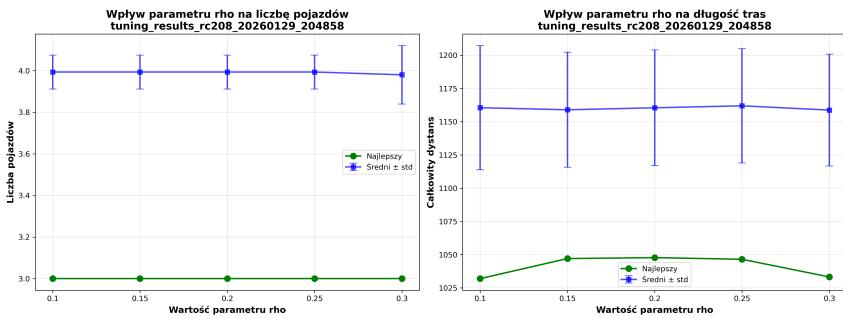
Instancja RC208 (mieszana)



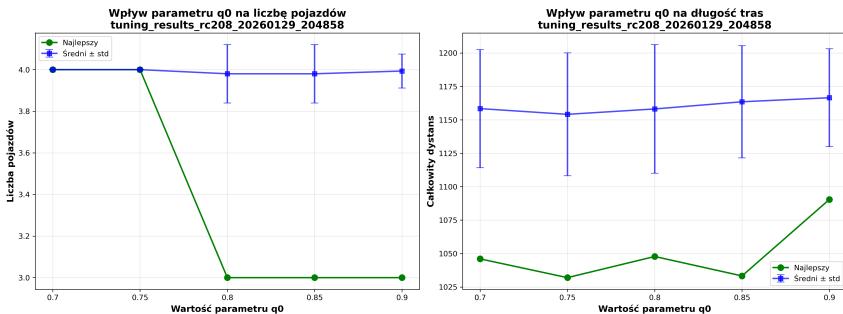
Rysunek 9: RC208: Wpływ parametru α — niskie wartości optymalne podobnie jak dla innych instancji



Rysunek 10: RC208: Wpływ parametru β — optymalne wartości 4.5–5.5



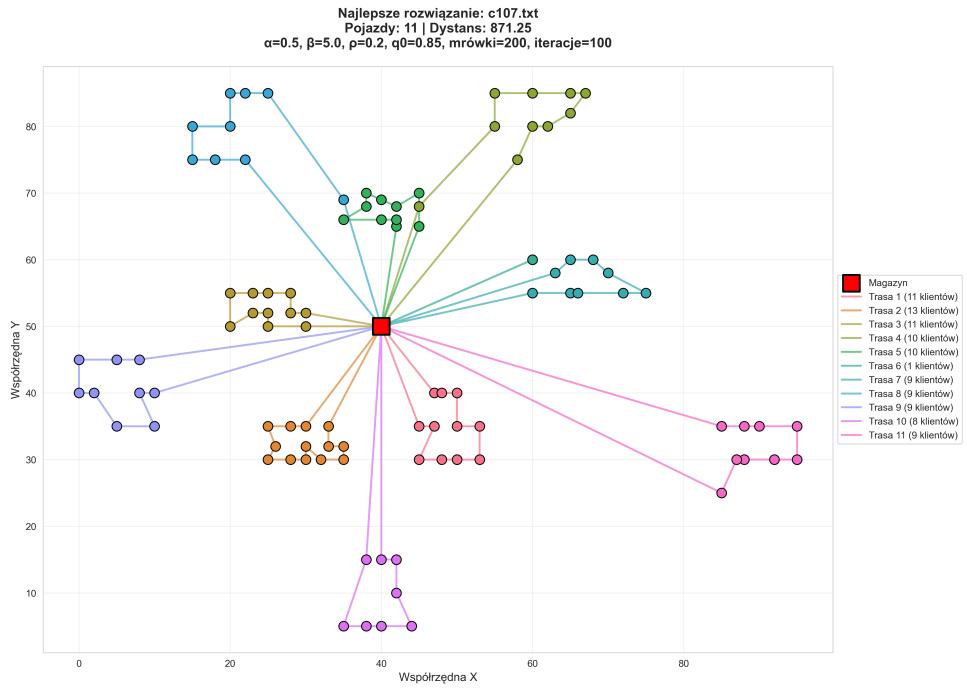
Rysunek 11: RC208: Wpływ parametru ρ — wartości 0.10–0.30 dają podobne wyniki



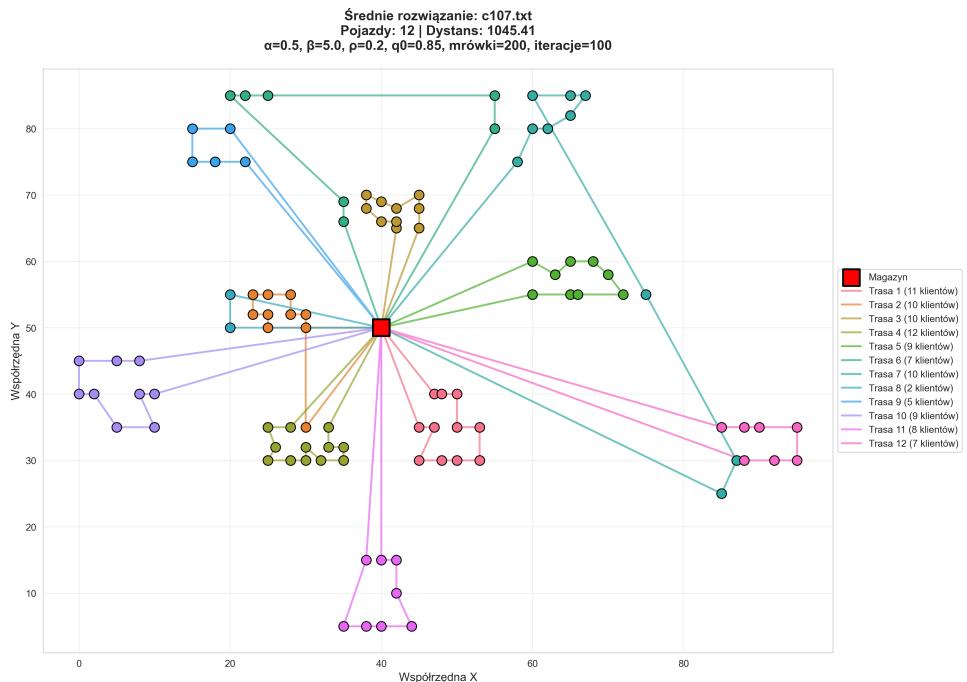
Rysunek 12: RC208: Wpływ parametru q_0 — wartości 0.75–0.85 zapewniają dobrą równowagę

8.4 Wizualizacje tras

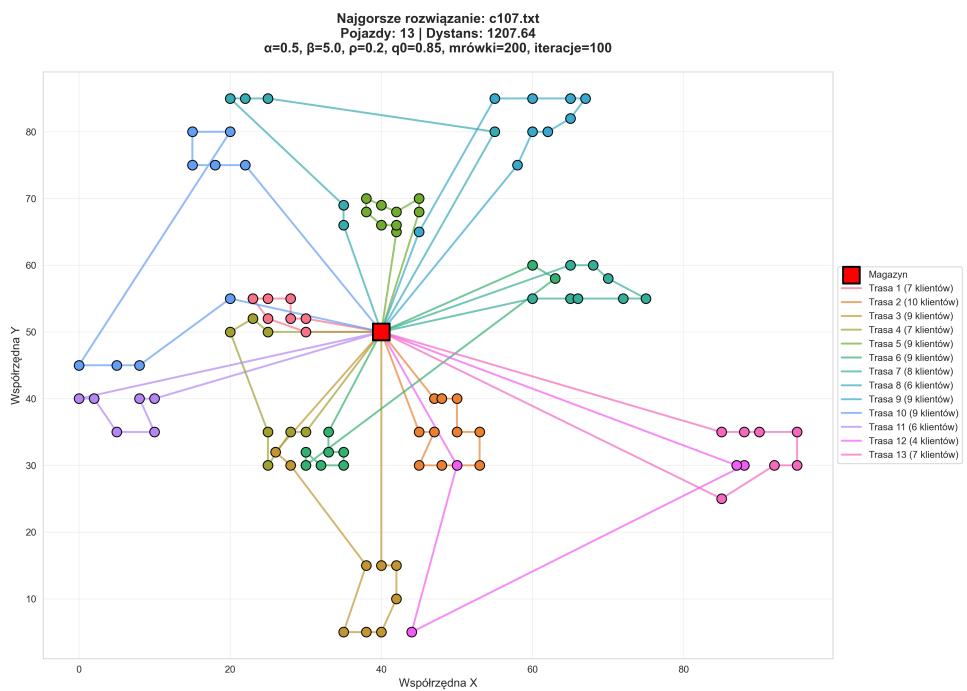
8.4.1 Instancja C107 (klastrowa)



Rysunek 13: Najlepsza znaleziona trasa dla instancji C107 (11 pojazdów, dystans: 871.25)

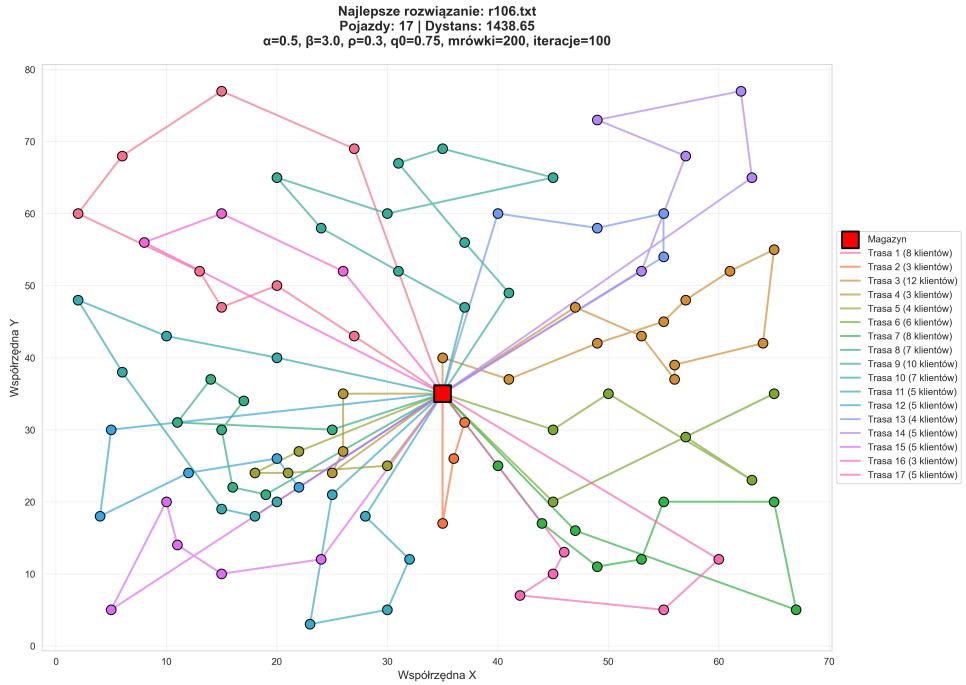


Rysunek 14: Średnie rozwiązanie dla instancji C107 — widoczne typowe grupowanie klientów w klastry

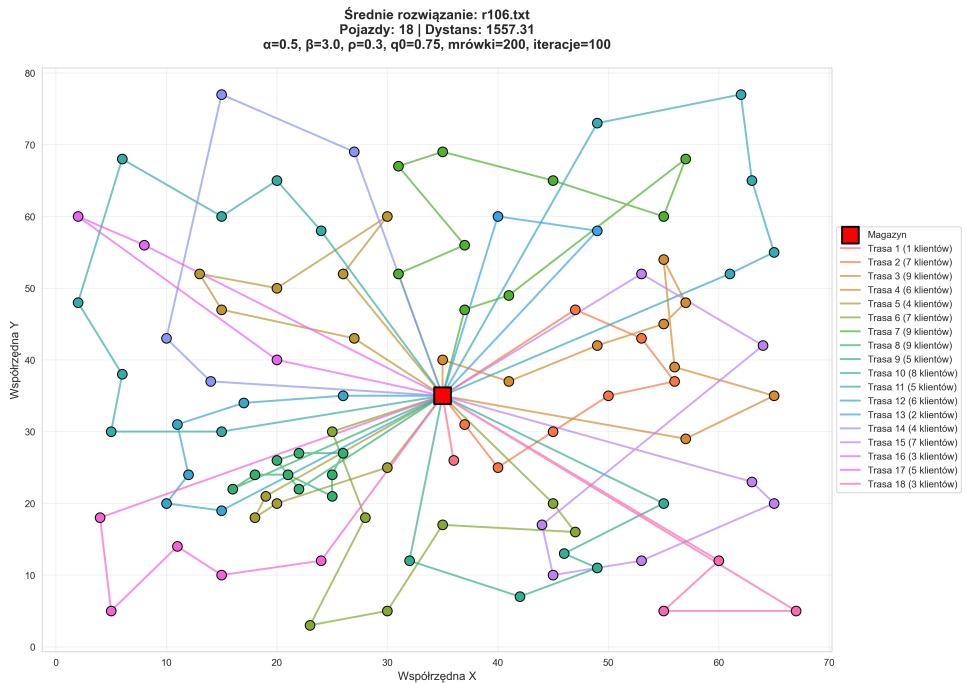


Rysunek 15: Najgorsze rozwiązanie dla instancji C107 (13 pojazdów, dystans: 1207.64)

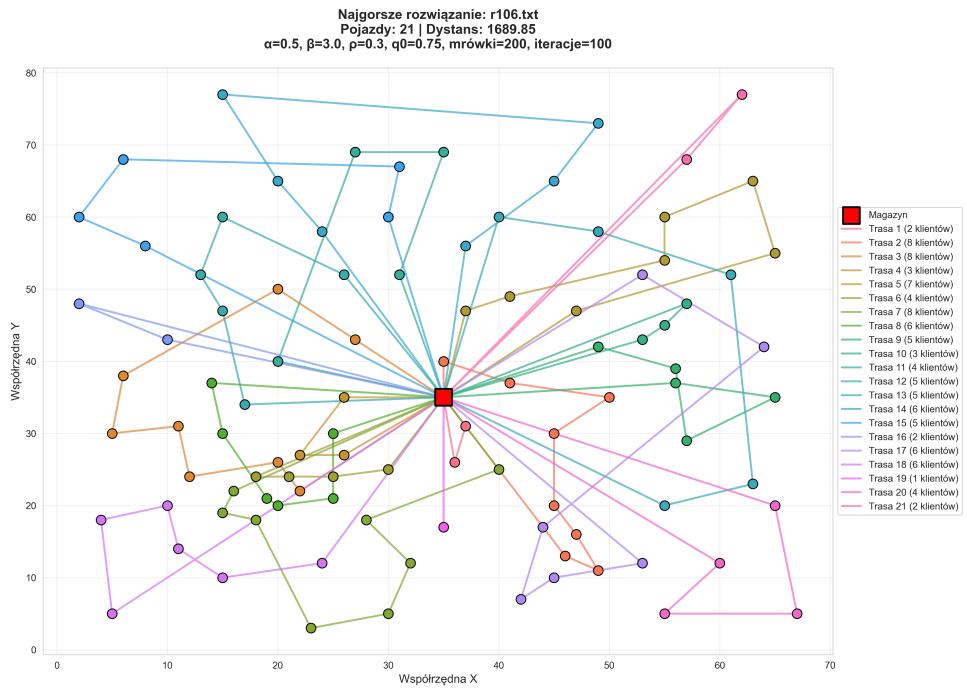
8.4.2 Instancja R106 (losowa)



Rysunek 16: Najlepsza znaleziona trasa dla instancji R106 (17 pojazdów, dystans: 1438.65)

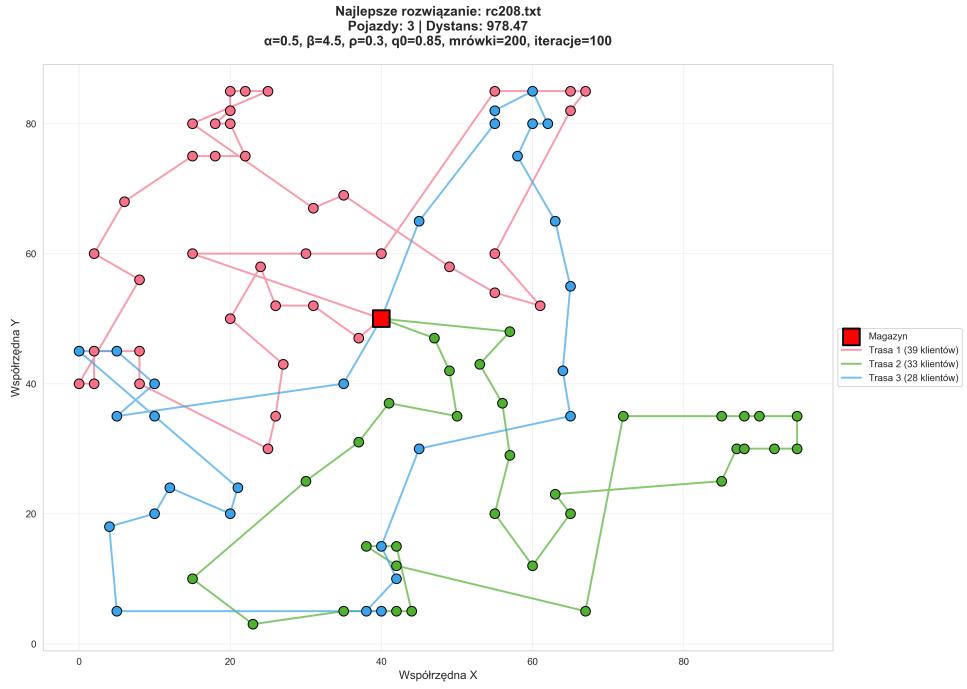


Rysunek 17: Średnie rozwiązanie dla instancji R106 — widoczne rozproszone rozmieszczenie klientów

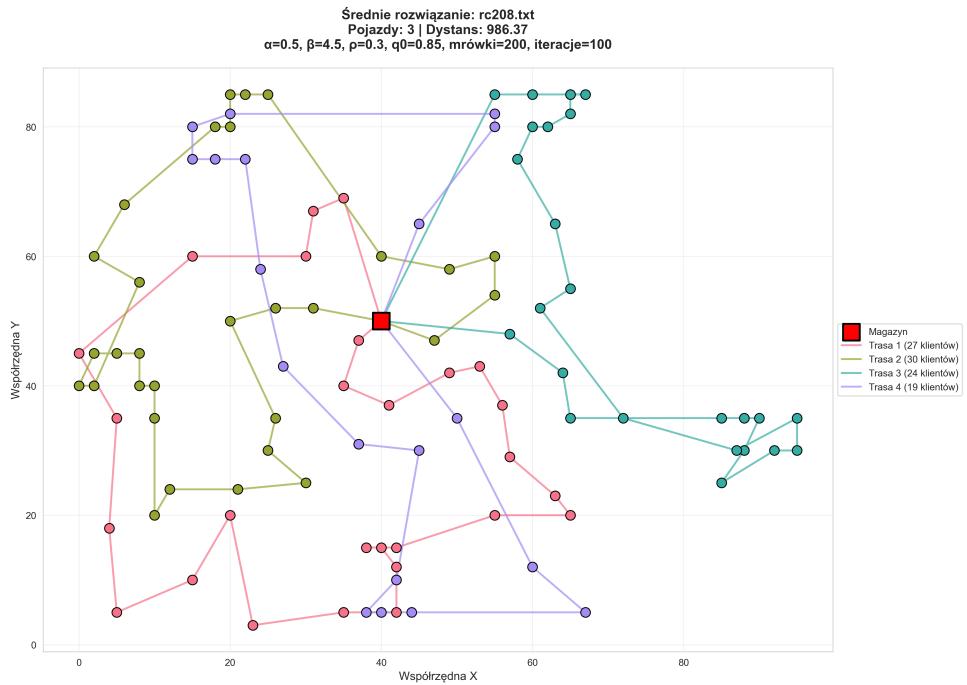


Rysunek 18: Najgorsze rozwiązanie dla instancji R106 (21 pojazdów, dystans: 1689.85)

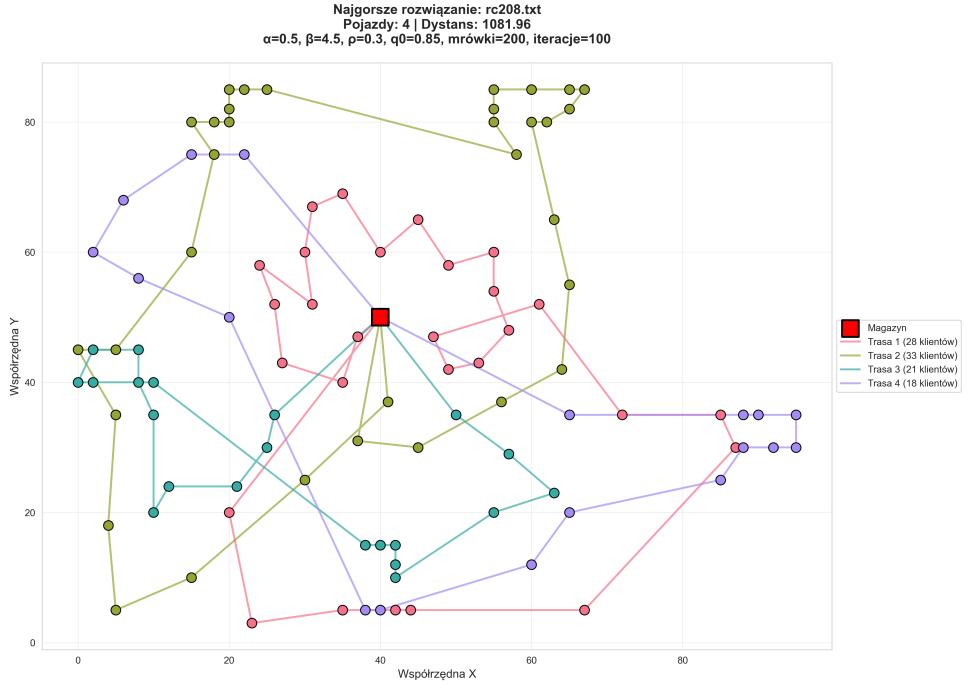
8.4.3 Instancja RC208 (mieszana)



Rysunek 19: Najlepsza znaleziona trasa dla instancji RC208 (3 pojazdy, dystans: 978.47) — optymalna liczba pojazdów



Rysunek 20: Średnie rozwiązanie dla instancji RC208 — mieszanka klastrów i rozmieszczenia losowego



Rysunek 21: Najgorsze rozwiązanie dla instancji RC208 (4 pojazdy, dystans: 1081.96)

8.5 Analiza zbieżności i stabilności

Na podstawie 50 uruchomień dla każdej instancji zaobserwowano:

Metryka	C107	R106	RC208
Odh. std. dystansu / średnia	8.1%	3.6%	4.3%
Dominujący wynik (pojazdy)	12 (54%)	18–19 (76%)	4 (98%)
Zakres pojazdów	11–14	17–21	3–4
Średni czas uruchomienia	0.66s	0.55s	1.01s

Obserwacje:

- Algorytm wykazuje wysoką stabilność dla wszystkich instancji (odch. std. $<10\%$ średniej)
- RC208 jest najbardziej stabilna — 98% uruchomień daje 4 pojazdy
- R106 ma największy rozrzut liczby pojazdów (17–21), co wynika z trudności instancji losowej
- Szybka zbieżność do dobrego rozwiązania (pierwsze 20–30 iteracji ACO)
- Optymalizacja lokalna (Relocate + 3-opt) znacząco poprawia jakość końcową

9 Porównanie z najlepszymi znanyymi rozwiązaniami (BKS)

Najlepsze znane rozwiązania (Best Known Solutions) pochodzą z literatury.

Instancja	BKS		Nasz najlepszy		Gap dystansu
	Pojazdy	Dystans	Pojazdy	Dystans	
C107	10	828.94	11	871.25	+5.1%
R106	12	1234.6	17	1438.65	+16.5%
RC208	3	828.14	3	978.47	+18.2%

Interpretacja:

- **RC208:** Osiągnięto optymalną liczbę pojazdów (3), gap dystansowy +18.2%
- **C107:** Gap dystansowy tylko +5.1% przy 1 dodatkowym pojeździe — bardzo bliski wynik do optimum
- **R106:** Instancja najtrudniejsza — +42% pojazdów względem BKS, gap dystansowy +16.5%

Wnioski:

- Instancje klastrowe (C) są łatwiejsze dla algorytmu ACO
- Instancje losowe (R) wymagają więcej pojazdów — trudniejsze do optymalizacji
- Szerze okna czasowe (RC208) ułatwiają redukcję liczby pojazdów

10 Analiza wyników

10.1 Wpływ parametrów na jakość rozwiązania

Na podstawie przeprowadzonych eksperymentów stwierdzono:

1. **Parametr α (wpływ feromonów):**
 - Niska wartość ($\alpha = 0.5$) daje najlepsze wyniki dla wszystkich instancji
 - Zmniejszenie wpływu feromonów pozwala na większą eksplorację
 - Wysokie α prowadzi do przedwczesnej zbieżności
2. **Parametr β (wpływ heurystyki):**
 - Optymalne wartości w zakresie 3.5–5.0
 - Wyższa wartość preferuje bliższych klientów i węższe okna czasowe
 - Wartości poniżej 3.0 dają słabsze wyniki
3. **Parametr ρ (wyparowywanie):**
 - Wartości w zakresie 0.10–0.25 dają najlepsze wyniki
 - Zbyt niskie ρ może prowadzić do stagnacji, zbyt wysokie do niestabilności
4. **Parametr q_0 (eksploracja):**
 - Optymalne w zakresie 0.80–0.90
 - Zbyt wysoka wartość (>0.9) ogranicza eksplorację i prowadzi do gorszych średnich wyników

10.2 Stabilność algorytmu

Analiza 50 uruchomień dla każdej instancji wykazała wysoką stabilność algorytmu:

- **C107:** Odch. std. dystansu 8.1% średniej, 54% uruchomień dało 12 pojazdów
- **R106:** Odch. std. dystansu 3.6% średniej, 76% uruchomień dało 18–19 pojazdów
- **RC208:** Odch. std. dystansu 4.3% średniej, 98% uruchomień dało 4 pojazdy

Algorytm jest stabilny dla wszystkich typów instancji — większość uruchomień daje podobne wyniki, co świadczy o dobrej zbieżności i powtarzalności.

10.3 Trudność instancji

- **Grupa R (losowe):** Najtrudniejsza — rozproszone rozmieszczenie klientów utrudnia optymalizację. Dla R106 nie udało się osiągnąć optymalnej liczby pojazdów (12 vs 17).
- **Grupa C (klastrowe):** Umiarkowana trudność — klienci w skupiskach ułatwiają tworzenie efektywnych tras. Dla C107 osiągnięto wynik bliski optymalnemu (11 pojazdów vs 10 optymalnych).
- **Grupa RC (mieszane):** Szersze okna czasowe (typ 2) znacznie ułatwiają optymalizację. Dla RC208 osiągnięto optymalną liczbę pojazdów (3).

10.4 Efektywność optymalizacji lokalnej

Połączenie ACO z operatorami Relocate i 3-opt okazało się kluczowe:

- Operator Relocate skutecznie redukuje liczbę pojazdów przez przenoszenie klientów między trasami
- 3-opt optymalizuje kolejność odwiedzin wewnętrz tras
- Bez optymalizacji lokalnej wyniki byłyby znacznie gorsze (szacunkowo +20–30% dystansu)

11 Wnioski

11.1 Podsumowanie

Na podstawie przeprowadzonych eksperymentów (750 konfiguracji parametrów, 50 uruchomień dla najlepszej konfiguracji) sformułowano następujące wnioski:

- **Skuteczność podejścia hybrydowego:** Połączenie ACO z operatorami Relocate i 3-opt pozwala osiągnąć wyniki bliskie najlepszym znanym rozwiązaniom. Dla instancji RC208 osiągnięto optymalną liczbę pojazdów (3), a dla C107 wynik różni się tylko o 1 pojazd od optymalnego (11 vs 10) z dystansem tylko 5.1% gorszym od BKS.
- **Wpływ parametrów:** Kluczowe znaczenie ma dobór parametrów. Niskie α (0.5) zapewnia lepszą eksplorację. Wartości β w zakresie 3.5–5.0 oraz q_0 w zakresie 0.80–0.90 dają najlepsze wyniki dla instancji klastrowych. Wartości ρ w zakresie 0.10–0.25 stabilizują poszukiwanie.
- **Charakterystyka instancji:** Instancje klastrowe (C) są najłatwiejsze do optymalizacji, a instancje losowe (R) najtrudniejsze. Szersze okna czasowe (typ 2) znaczco ułatwiają redukcję liczby pojazdów.
- **Stabilność:** Algorytm wykazuje dobrą stabilność — odchylenie standardowe dystansu wynosi ok. 4–8% średniej dla wielokrotnych uruchomień.
- **Wydajność:** Wykorzystanie biblioteki Numba do kompilacji JIT znaczco przyspiesza obliczenia (średni czas uruchomienia <1s po kompilacji), umożliwiając użycie większej liczby mrówek i iteracji.

11.2 Rekomendowane parametry

Na podstawie strojenia 750 konfiguracji dla trzech instancji sugerujemy następujące wartości parametrów:

Parametr	Zalecana wartość	Uzasadnienie
Liczba mrówek (n_ants)	200	Dobra eksploracja przestrzeni rozwiązań
Liczba iteracji (n_iterations)	100	Wystarczające do zbieżności
Wpływ feromonów (α)	0.5	Mniejszy wpływ = lepsza eksploracja
Wpływ heurystyki (β)	3.5–5.0	Balans między bliskością a jakością
Wyparowywanie (ρ)	0.10–0.25	Umiarkowane zapominanie stabilizuje
Próg eksploatacji (q_0)	0.80–0.90	Dobra równowaga eksploatacji/eksploracji

Uwaga: Wartości te zostały wyznaczone eksperymentalnie i mogą wymagać dostrojenia dla innych instancji. Dla instancji losowych (typ R) zaleca się niższe q_0 (większa eksploracja), a dla klastrowych (typ C) można zwiększyć β .

11.3 Ograniczenia rozwiązania

- **Złożoność obliczeniowa:** Operator 3-opt ma złożoność $O(n^3)$, co może być wąskim gardłem dla tras z ponad 30 klientami. Dla większych instancji (200+ klientów) czas obliczeń rośnie znaczco
- **Wrażliwość na parametry:** Nieoptimalne parametry mogą prowadzić do rozwiązań gorszych o 20–40%. Wymaga strojenia dla nowych typów instancji
- **Brak adaptacji:** Parametry są stałe w trakcie działania algorytmu — brak mechanizmu adaptacyjnego dostosowującego α , β , ρ do postępu optymalizacji

11.4 Kierunki dalszej poprawy

- Zastosowanie nowych algorytmów optymalizacji lokalnej (intra-route i inter-route) lub poprawa zaimplementowanych
- Adaptacyjne dostrajanie parametrów w trakcie działania algorytmu
- Równoległa optymalizacja lokalna dla wielu tras jednocześnie