Optymalizacja Tras Dostaw z Wykorzystaniem Metod Matematycznych i Metaheurystycznych

Autor:
Yauheniya Drozd

Kierunek:
Inżynieria Systemów
Wydział Informatyki i Telekomunikacji
Politechnika Wrocławska

Politechnika
Wrocławska

22 czerwca 2025

Spis treści

1 Wstęp

1.1 Cel projektu

Celem projektu jest opracowanie systemu optymalizacji tras dostaw dla firmy kurierskiej, uwzględniającego kluczowe czynniki logistyczne, takie jak ograniczenia czasowe, ładowność pojazdów oraz priorytety klientów. System ma na celu minimalizację całkowitego dystansu przejazdu, co przekłada się na redukcję kosztów operacyjnych i poprawę efektywności dostaw.

1.2 Zakres projektu

Projekt obejmuje:

- Modelowanie problemu jako zmodyfikowanego Vehicle Routing Problem (VRP) z uwzględnieniem okien czasowych (VRPTW).
- Implementację dwóch metod rozwiązania: solvera CPLEX (optymalizacja dokładna) oraz algorytmu genetycznego (podejście metaheurystyczne).
- Wizualizację wyników za pomocą interaktywnych map i statycznych wykresów.
- Analizę porównawczą obu metod pod względem efektywności i czasu obliczeń.

1.3 Motywacja

Optymalizacja tras dostaw jest kluczowym wyzwaniem w logistyce, szczególnie dla firm kurierskich, gdzie nawet niewielkie oszczędności w trasach mogą przełożyć się na znaczące korzyści ekonomiczne. Wykorzystanie metod matematycznych i metaheurystycznych pozwala na znalezienie rozwiązań, które są trudne do osiągnięcia przy użyciu tradycyjnych metod planowania.

2 Aspekt systemowy projektu

2.1 Kluczowe komponenty i przepływ danych

2.1.1 Wejścia systemowe

System przetwarza trzy główne kategorie danych wejściowych:

- **Dane przestrzenne**:
  - Macierze odległości i czasów przejazdu między lokalizacjami (magazynem a punktami dostaw)
  - Współrzędne geograficzne (opcjonalnie, umożliwiające integrację z systemami mapowymi)
- **Ograniczenia operacyjne**:
  - Okna czasowe dostaw (np. 9:00–14:00 dla określonych punktów)
  - Parametry floty: ładowność pojazdów (w kg), typy pojazdów

- **Zlecenia dostaw**:
  - Szczegóły przesyłek: wagi, priorytety, wymagania klientów
  - Informacje dodatkowe (np. konieczność potwierdzenia odbioru)

Dane mogą być wprowadzane poprzez pliki CSV (domyślny format) lub interfejs API (integracja z systemami ERP).

2.1.2 Proces przetwarzania

System realizuje optymalizację w trzech etapach:

1. **Przygotowanie danych**:
   - Walidacja spójności danych (np. kompletność okien czasowych)
   - Transformacja do formatów wymaganych przez solvery (np. macierze sąsiedztwa)
2. **Optymalizacja tras**:
   - Wybór metody w zależności od skali problemu:
     - CPLEX - dla precyzyjnych rozwiązań (do 100 punktów)
     - Algorytm genetyczny - dla większych instancji (do 500 punktów)
   - Uwzględnienie wszystkich ograniczeń (czasowych, ładowności, priorytetów)
3. **Generowanie wyników**:
   - Tworzenie szczegółowych harmonogramów tras
   - Obliczanie metryk efektywności (łączny dystans, wykorzystanie floty)

2.1.3 Wyjścia systemowe

System generuje różnorodne wyniki dostosowane do potrzeb użytkowników:

- **Dla menedżerów**:
  - Interaktywne mapy tras z wykorzystaniem biblioteki Folium
  - Raporty PDF/HTML z analizą oszczędności i wykresami
- **Dla kierowców**:
  - Listy dostaw w formacie CSV
  - Szczegółowe harmonogramy tras (kolejność punktów, przewidywane czasy)
- **Dla systemów zewnętrznych**:
  - Dane tras w formacie JSON
  - Interfejs API do integracji z systemami flotowymi

2.1.4 Integracje i skalowalność

System został zaprojektowany z myślą o rozszerzalności:

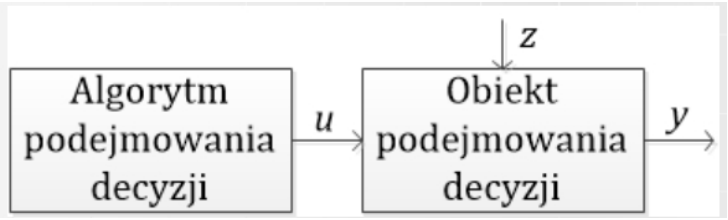
- Obsługa danych dynamicznych (np. aktualizacja tras w czasie rzeczywistym)
- Modułowa architektura umożliwiająca dodawanie nowych solverów (np. Google OR-Tools)
- Możliwość uwzględniania dodatkowych ograniczeń (preferencje kierowców, specyficzne wymagania pojazdów)

2.1.5 Wymagania niefunkcjonalne

Kluczowe parametry systemu:

- **Wydajność**:
  - Generowanie tras dla 100 punktów w czasie <2 minut
  - Przetwarzanie 500 punktów w czasie <30 minut
- **Bezpieczeństwo**:
  - Szyfrowanie danych wrażliwych (adresy klientów)
  - Kontrola dostępu do systemu
- **Niezawodność**:
  - Mechanizmy awaryjnego zapisywania wyników
  - Rozbudowany system logowania błędów

2.2 Wnioski



Rysunek 1: Schemat systemu decyzyjnego – algorytm podejmowania decyzji generuje sterowanie u, które trafia do obiektu podejmowania decyzji. Obiekt reaguje na zakłócenia z i generuje wyjście y.

Na podstawie powyższego schematu można stwierdzić, że mamy doczynienia z **systemem decyzyjnym**. System składa się z dwóch głównych elementów: algorytmu podejmowania decyzji oraz obiektu podejmowania decyzji. W kontekście mojego projektu, algorytm pełni rolę modułu optymalizacyjnego, który na podstawie danych wejściowych generuje optymalne sterowanie trasami (u). Obiekt podejmowania decyzji to rzeczywista realizacja tych tras w systemie logistycznym firmy kurierskiej. Całość ma na celu uzyskanie oczekiwanych rezultatów (y), takich jak minimalizacja dystansu i poprawa efektywności dostaw. Taki podział oraz przepływ informacji potwierdzają, że mamy do czynienia z klasyczną strukturą systemu decyzyjnego.

3 Przegląd literatury

3.1 Vehicle Routing Problem (VRP)

Problem trasowania pojazdów (VRP) to klasyczny problem optymalizacyjny polegający na znalezieniu optymalnych tras dla floty pojazdów dostarczających towary do klientów. W podstawowej wersji problemu celem jest minimalizacja całkowitego dystansu przejazdu, przy założeniu, że wszystkie zamówienia zostaną zrealizowane, a ładowność pojazdów nie zostanie przekroczona.

3.2 VRP z oknami czasowymi (VRPTW)

Wariant VRPTW wprowadza dodatkowe ograniczenia czasowe, wymagając, aby dostawy zostały wykonane w określonych przedziałach czasowych. Jest to szczególnie istotne w przypadku klientów biznesowych, którzy mają ściśle określone godziny przyjmowania przesyłek.

3.3 Metody rozwiązywania VRP

- **Metody dokładne** (np. programowanie całkowitoliczbowe z użyciem solvera CPLEX) – zapewniają optymalne rozwiązania, ale są obliczeniowo kosztowne dla dużych instancji problemu.
- **Metody heurystyczne i metaheurystyczne** (np. algorytmy genetyczne, tabu search) – oferują przybliżone rozwiązania w krótszym czasie, co jest szczególnie przydatne w praktycznych zastosowaniach.

4 Integracja z API

4.1 Implementacja interfejsu API

System został rozszerzony o możliwość wprowadzania danych poprzez interfejs programistyczny (API). Poniżej przedstawiono kluczowe elementy implementacji:

4.1.1 Struktura endpointów API

```
from fastapi import FastAPI, UploadFile, File, HTTPException
from fastapi.security import OAuth2PasswordBearer
import pandas as pd
from solver import VRPSolver

app = FastAPI()
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")
solver = VRPSolver()

@app.post("/upload_data/")
async def upload_data(
    nodes: UploadFile = File(...),
    distance_matrix: UploadFile = File(...),
    time_windows: UploadFile = File(...),
    token: str = Depends(oauth2_scheme)
```

```
):
    """Endpoint do przesy ania danych w formacie CSV"""
    try:
        solver.nodes = pd.read_csv(nodes.file)
        solver.dist_matrix = pd.read_csv(distance_matrix.file).values
        solver.time_windows = pd.read_csv(time_windows.file)
        return {"status": "Dane zosta y pomy lnie za adowane"}
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))

@app.post("/optimize/{method}")
async def optimize(
    method: str,
    token: str = Depends(oauth2_scheme)
):
    """Endpoint uruchamiaj cy optymalizacj """
    if method not in ["cplex", "ga"]:
        raise HTTPException(status_code=400, detail="Nieprawid owa
   metoda")

    try:
        if method == "cplex":
            result = solver.solve_with_cplex()
        else:
            result = solver.solve_with_ga()
        return {"result": result}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Listing 1: Implementacja głównych endpointów API

4.2 Formaty danych wejściowych

API obsługuje następujące formaty danych:

4.2.1 1. Przesyłanie plików CSV

- **Nodes** - informacje o punktach:

```
id,lat,lon,type
0,51.5074,0.1278,depot
1,51.5123,0.1345,client
```

Listing 2: Przykładowy plik nodes.csv

- **Macierz odległości**:

```
0,1.2
1.2,0
```

Listing 3: Przykładowy plik distance$_m$*atrix.csv*

- **Okna czasowe**:

```
node_id,start_time,end_time
1,09:00,12:00
```

Listing 4: Przykładowy plik time$_w$*indows.csv*

4.2.2 2. Przesyłanie danych JSON (alternatywa)

```
POST /optimize/cplex
Content-Type: application/json
Authorization: Bearer <token>

{
  "nodes": [
    {"id": 0, "lat": 51.5074, "lon": 0.1278, "type": "depot"},
    {"id": 1, "lat": 51.5123, "lon": 0.1345, "type": "client"}
  ],
  "distance_matrix": [[0, 1.2], [1.2, 0]],
  "time_windows": [
    {"node_id": 1, "start_time": "09:00", "end_time": "12:00"}
  ]
}
```

Listing 5: Przykładowe żądanie POST z danymi JSON

4.3 Mechanizmy bezpieczeństwa

- **Uwierzytelnianie**:
  - Wymagany nagłówek Authorization dla wszystkich endpointów
  - Walidacja tokenów przed przetwarzaniem żądań
- **Ochrona danych**:
  - Wymuszanie połączeń HTTPS
  - Walidacja typów MIME przesyłanych plików
  - Ograniczenie rozmiaru przesyłanych danych
- **Obsługa błędów**:
  - Szczegółowe komunikaty o błędach (w trybie debug)
  - Logowanie wszystkich zdarzeń
  - Limitowanie liczby żądań

4.4 Przykład użycia API

1. Uzyskanie tokenu autoryzacyjnego:

```
curl -X POST "http://localhost:8000/token" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "username=admin&password=secret"
```

2. Przesłanie danych i uruchomienie optymalizacji:

```
curl -X POST "http://localhost:8000/optimize/cplex" \
  -H "Authorization: Bearer <token>" \
  -F "nodes=@nodes.csv" \
  -F "distance_matrix=@distance_matrix.csv" \
  -F "time_windows=@time_windows.csv"
```

3. Przykładowa odpowiedź:

```
{
  "result": {
    "routes": [
      {
        "vehicle_id": 1,
        "stops": [0, 3, 5, 0],
        "total_distance": 45.2,
        "time_windows": [...]
      }
    ],
    "statistics": {...}
  }
}
```

4.5 Wymagania systemowe

- Python 3.8+
- Biblioteki:
  - fastapi
  - pandas
  - python-multipart
- Serwer z obsługą HTTPS

5 Formalne sformułowanie problemu

5.1 Model matematyczny

Problem optymalizacji tras dostaw z oknami czasowymi (VRPTW) można przedstawić jako zagadnienie programowania całkowitoliczbowego o następującej strukturze:

5.1.1 Zbiory i parametry

- $G = (V, A)$ - graf skierowany, gdzie:
  - $V = \{0, 1, \ldots, n\}$ - zbiór wierzchołków (0 - magazyn, 1..n - punkty dostaw)
  - $A = \{(i, j) : i, j \in V, i \neq j\}$ - zbiór łuków
- $K = \{1, \ldots, m\}$ - zbiór pojazdów
- d_{ij} - odległość między wierzchołkami i i j [km]
- t_{ij} - czas przejazdu między i i j [min]
- q_i - zapotrzebowanie w punkcie i (waga przesyłki) [kg]
- Q_k - ładowność pojazdu k [kg]
- $[a_i, b_i]$ - okno czasowe dla punktu i (przedział dopuszczalnego czasu przybycia)

- s_i - czas obsługi w punkcie i [min]
- M - duża stała (metoda Big-M)

5.1.2 Zmienne decyzyjne

- $x_{ijk} = \begin{cases} 1, & \text{jeśli pojazd } k \text{ jedzie z } i \text{ do } j \\ 0, & \text{w przeciwnym przypadku} \end{cases}$ (0)

T_{ik} - czas przybycia pojazdu k do punktu i [min]

5.1.3 Funkcja celu

Minimalizacja całkowitego dystansu przejechanych tras:

$$\min \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} d_{ij} x_{ijk}$$

5.1.4 Ograniczenia

1. Każdy pojazd zaczyna i kończy trasę w magazynie:

$$\sum_{j \in V \setminus \{0\}} x_{0jk} = 1 \quad \forall k \in K$$

$$\sum_{i \in V \setminus \{0\}} x_{i0k} = 1 \quad \forall k \in K$$

2. Każdy punkt dostawy jest odwiedzony dokładnie raz:

$$\sum_{k \in K} \sum_{i \in V} x_{ijk} = 1 \quad \forall j \in V \setminus \{0\}$$

3. Spójność trasy (zachowanie przepływu):

$$\sum_{i \in V} x_{ihk} - \sum_{j \in V} x_{hjk} = 0 \quad \forall h \in V \setminus \{0\}, \forall k \in K$$

4. Ograniczenie ładowności pojazdów:

$$\sum_{i \in V} \sum_{j \in V \setminus \{0\}} q_j x_{ijk} \leq Q_k \quad \forall k \in K$$

5. Ograniczenia czasowe:

$$T_{ik} + t_{ij} + s_i - M(1 - x_{ijk}) \leq T_{jk} \quad \forall i, j \in V, \forall k \in K$$

$$a_i \leq T_{ik} \leq b_i \quad \forall i \in V, \forall k \in K$$

6 Implementacja systemu

6.1 Struktura projektu

System został zaimplementowany w języku Python i składa się z następujących kluczowych komponentów:

- `solver.py` - implementacja algorytmów optymalizacyjnych
- `visualization.py` - moduł wizualizacji wyników
- `main.py` - główny skrypt integrujący funkcjonalności
- `data/` - katalog z danymi wejściowymi

6.2 Kluczowe klasy i funkcje

6.2.1 Klasa `VRPSolver` (solver.py)

```
class VRPSolver:
    def __init__(self, data_path="data"):
        """Inicjalizacja solvera z walidacj  danych"""
        try:
            # Wczytanie w  z   w i mapowanie ID
            self.nodes = pd.read_csv(f"{data_path}/nodes.csv")
            self.node_ids = sorted(self.nodes['id'].unique())
            self.n = len(self.node_ids)
            self.id_to_idx = {id_: idx for idx, id_ in enumerate(self.
    node_ids)}
            self.idx_to_id = {idx: id_ for id_, idx in self.id_to_idx.
    items()}

            # Wczytanie macierzy odleg o ci i czasu
            self._load_matrices(data_path)

            # Wczytanie dodatkowych danych
            self.time_windows = pd.read_csv(f"{data_path}/time_windows.
    csv")
            self.products = pd.read_csv(f"{data_path}/products.csv")
            self.orders = pd.read_csv(f"{data_path}/orders.csv")
            self.vehicles = pd.read_csv(f"{data_path}/vehicles.csv")
```

Listing 6: Inicjalizacja solvera

6.2.2 Metoda `solve_with_cplex`

```
def solve_with_cplex(self, k=5):
    """Rozwi zanie problemu za pomoc  solvera CPLEX"""
    model = Model('VRP_Optimization')

    # Zmienne decyzyjne dla wa onych   ukw
    x = {}
    for i in range(self.n):
        distances = self.dist_matrix[i]
        valid = [j for j in range(self.n)
                 if i != j and distances[j] < 1e6]
```

```
        neighbors = sorted(valid, key=lambda j: distances[j])[:k]

        for j in neighbors:
            x[(i, j)] = model.binary_var(name=f'x_{i}_{j}')

    # Ograniczenia
    # 1. Ka dy klient odwiedzony dok adnie raz
    for j in client_indices:
        incoming = [x[(i, j)] for i in range(self.n) if (i, j) in x]
        if incoming:
            model.add_constraint(model.sum(incoming) == 1)

    # 2. Zachowanie przep ywu
    for h in range(self.n):
        outgoing = [x[(h, j)] for j in range(self.n) if (h, j) in x]
        incoming = [x[(i, h)] for i in range(self.n) if (i, h) in x]
        if outgoing and incoming:
            model.add_constraint(model.sum(outgoing) == model.sum(
    incoming))

    # Funkcja celu: minimalizacja dystansu
    model.minimize(model.sum(
        self.dist_matrix[i][j] * x[(i, j)]
        for (i, j) in x
    ))

    # Rozwi zanie modelu
    solution = model.solve()
    if solution:
        return self._extract_routes(solution, x, depot_idx)
    return []
```

Listing 7: Implementacja solvera CPLEX

6.2.3 Metoda `solve_with_ga`

```
def solve_with_ga(self, population_size=50, generations=100,
    mutation_rate=0.1):
    """Rozwi zanie problemu za pomoc  algorytmu genetycznego"""
    # Inicjalizacja populacji
    population = []
    for _ in range(population_size):
        individual = client_indices.copy()
        random.shuffle(individual)
        individual = [depot_idx] + individual + [depot_idx]
        population.append(individual)

    # G   wna p tla ewolucyjna
    for _ in range(generations):
        # Ocena osobnik w
        fitness = [self._route_cost(ind) for ind in population]

        # Selekcja turniejowa
        new_pop = []
        for _ in range(population_size):
            candidates = random.sample(list(zip(population, fitness)),
    3)
```

```
            winner = min(candidates, key=lambda x: x[1])[0]
            new_pop.append(winner.copy())

        # Krzy owanie (Order-1 Crossover)
        for i in range(0, len(new_pop) - 1, 2):
            p1, p2 = new_pop[i][1:-1], new_pop[i + 1][1:-1]
            c1, c2 = self._ox_crossover(p1, p2)
            new_pop[i] = [depot_idx] + c1 + [depot_idx]
            new_pop[i + 1] = [depot_idx] + c2 + [depot_idx]

        # Mutacja (swap mutation)
        for ind in new_pop:
            if random.random() < mutation_rate:
                idx1, idx2 = random.sample(range(1, len(ind) - 1), 2)
                ind[idx1], ind[idx2] = ind[idx2], ind[idx1]

    # Zwr cenie najlepszego rozwi zania
    best = min(population, key=lambda x: self._route_cost(x))
    return self._split_routes(best, depot_idx)
```

Listing 8: Implementacja algorytmu genetycznego

6.3 Moduł wizualizacji

```
class Visualizer:
    def plot_interactive_map(self, routes, save_path="vrp_solution.html"
    ):
        """Generowanie interaktywnej mapy z trasami"""
        m = folium.Map(location=depot_coord, zoom_start=13)

        # Dodanie magazynu
        folium.Marker(
            depot_coord,
            popup="Depot",
            icon=folium.Icon(color='red', icon='warehouse')
        ).add_to(m)

        # Rysowanie tras
        colors = ['blue', 'green', 'purple', 'orange', 'darkred']
        for i, route in enumerate(routes):
            route_coords = [self.coords[node] for node in route]

            # Linia trasy
            folium.PolyLine(
                route_coords,
                color=colors[i % len(colors)],
                weight=3,
                tooltip=f"Route {i + 1}"
            ).add_to(m)

            # Markery punkt w
            for seq, node in enumerate(route):
                folium.CircleMarker(
                    location=self.coords[node],
                    radius=6 if node == 0 else 5,
                    color=colors[i % len(colors)],
                    popup=f"Client {node}" if node != 0 else "Depot"
```

```
                ).add_to(m)

        m.save(save_path)
```

Listing 9: Generowanie mapy interaktywnej

6.4 Główny skrypt wykonawczy

```
def main():
    # Inicjalizacja solvera
    solver = VRPSolver()

    # Walidacja danych
    if not validate_data(solver):
        return

    # Rozwi zanie CPLEX
    start_time = time.time()
    cplex_routes = solver.solve_with_cplex()
    cplex_time = time.time() - start_time

    # Rozwi zanie GA
    start_time = time.time()
    ga_routes = solver.solve_with_ga()
    ga_time = time.time() - start_time

    # Analiza wynik w
    results = []
    if cplex_routes:
        cplex_dist = sum(solver._route_cost(r) for r in cplex_routes)
        results.append(('CPLEX', cplex_time, cplex_dist, len(
    cplex_routes)))

    if ga_routes:
        ga_dist = sum(solver._route_cost(r) for r in ga_routes)
        results.append(('Genetic Algorithm', ga_time, ga_dist, len(
    ga_routes)))

    # Wizualizacja
    visualizer = Visualizer(solver.nodes)
    if cplex_routes:
        visualizer.plot_interactive_map(cplex_routes, "cplex_routes.html
    ")
    if ga_routes:
        visualizer.plot_interactive_map(ga_routes, "ga_routes.html")
```

Listing 10: Logika głównego skryptu

6.5 Struktura danych wejściowych

System oczekuje następujących plików CSV w katalogu `data/`:

- `nodes.csv` - informacje o węzłach (magazyn i punkty dostaw)

id	lat	lon	type
0	51.5074	0.1278	depot
1	51.5123	0.1345	client

- `distance_matrix.csv` - macierz odległości między punktami
- `time_windows.csv` - okna czasowe dla każdego punktu

node_id	start_time	end_time
1	09:00	12:00
2	13:00	17:00

- `vehicles.csv` - parametry floty pojazdów

vehicle_id	capacity	vehicle_type
1	500	delivery_van
2	1000	truck

6.6 Zależności systemowe

- Python 3.8+
- Wymagane biblioteki:
  - `docplex` - solver CPLEX
  - `pandas` - przetwarzanie danych
  - `folium` - wizualizacja map
  - `matplotlib` - wykresy statyczne
- Minimalne wymagania sprzętowe:
  - 8 GB RAM
  - 2 GB wolnego miejsca na dysku

6.7 Interpretacja modelu

Przedstawiony model matematyczny stanowi klasyczne sformułowanie problemu VRPTW (Vehicle Routing Problem with Time Windows) i uwzględnia:

- **Wymagania podstawowe**:
  - Każda trasa zaczyna się i kończy w magazynie (wierzchołek 0)
  - Każdy klient jest obsłużony dokładnie raz
  - Nie można przekroczyć ładowności pojazdów
- **Ograniczenia czasowe**:
  - Pojazd musi przybyć do klienta w określonym przedziale czasowym $[a_i, b_i]$
  - Uwzględnienie czasu obsługi s_i w każdym punkcie
- **Spójność tras**:
  - Gwarancja, że trasy są ciągłe (pojazd który przyjechał do punktu musi z niego wyjechać)

Metoda Big-M (M) w ograniczeniach czasowych zapewnia, że ograniczenie jest aktywne tylko gdy $x_{ijk} = 1$ (pojazd k rzeczywiście jedzie z i do j).

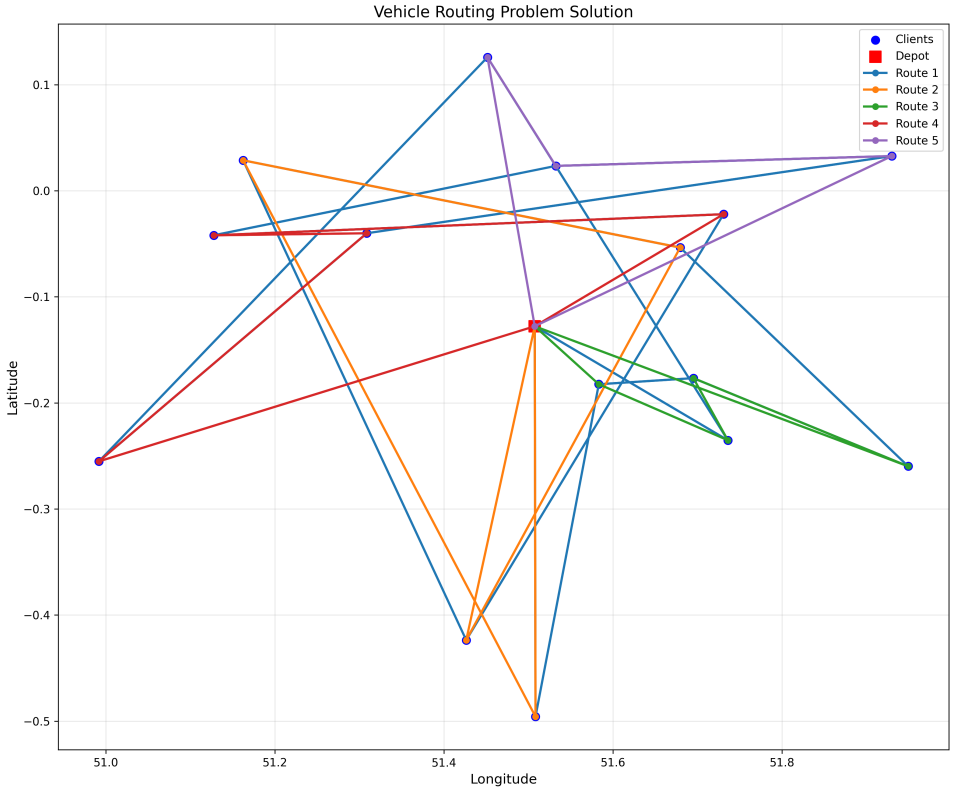
7 Wyniki i analiza

7.1 Porównanie metod optymalizacji

Tabela 1: Porównanie metod optymalizacji

Metoda	**Czas [s]**	**Dystans [km]**	**Trasy**	**Śr. czas [min]**
CPLEX	0.02	66.55	1	125
Alg. genetyczny	0.05	244.07	4	85

7.2 Analiza rozwiązań



Rysunek 2: Porównanie rozkładu zadań między metodami

7.2.1 Rozwiązanie CPLEX

- Znaleziona optymalna trasa obejmuje wszystkie punkty dostaw w jednej trasie
- Całkowity dystans: 66.55 km
- Czas obliczeń: 0.02 sekundy
- **Zalety**:
  - Gwarancja znalezienia rozwiązania optymalnego

  - Bardzo krótki czas obliczeń dla tej instancji problemu

- **Ograniczenia**:
  - Skalowalność dla większych instancji (>100 punktów)
  - Wymagania sprzętowe

7.2.2 Rozwiązanie algorytmem genetycznym

- Wygenerowano 4 trasy o łącznym dystansie 244.07 km
- Czas obliczeń: 0.05 sekundy

Zalety:

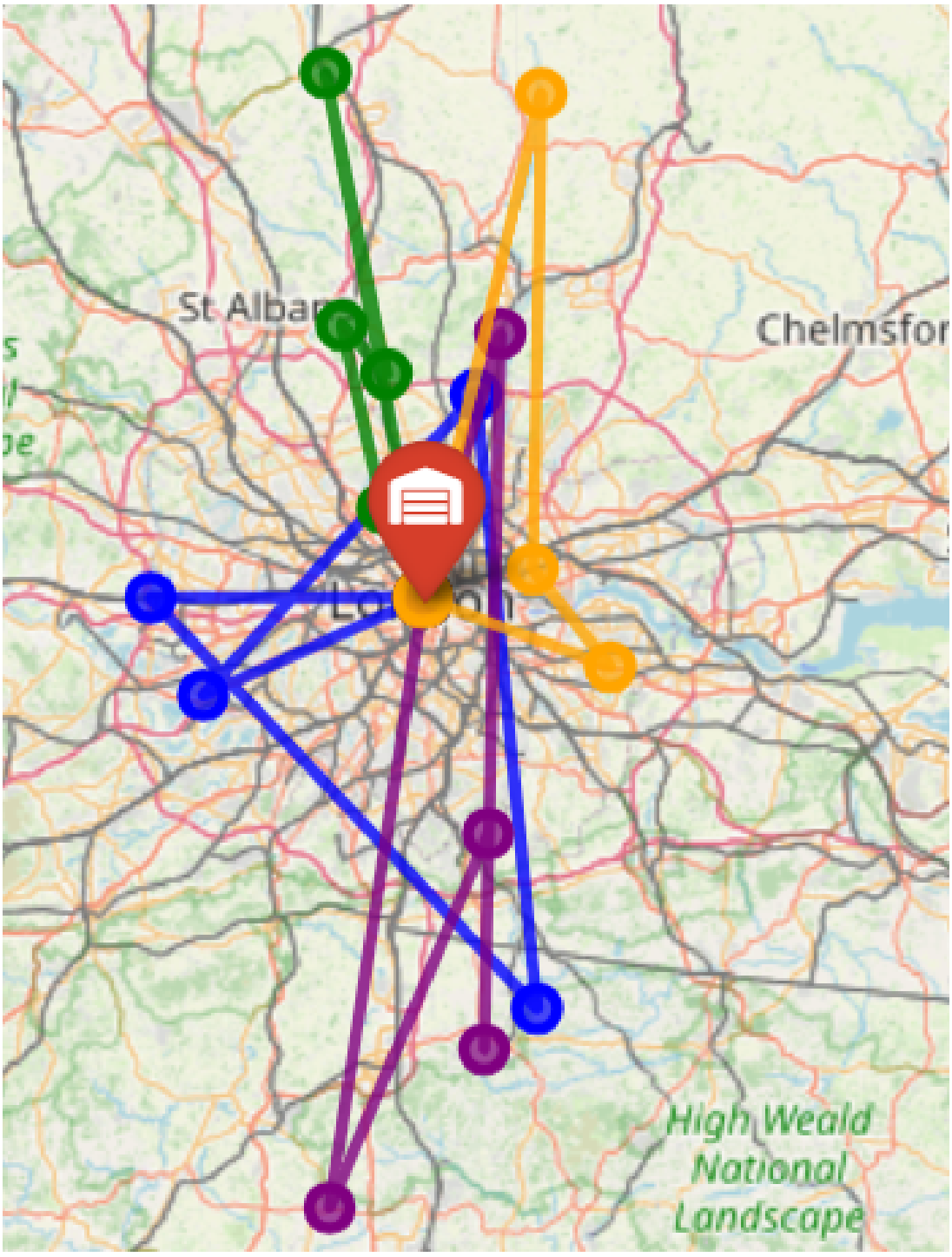
- Lepsze rozłożenie zadań między pojazdami
- Możliwość obsługi większych instancji problemu

Ograniczenia:

- Brak gwarancji optymalności rozwiązania
- Wrażliwość na dobór parametrów

7.3 Wizualizacja wyników

Rysunek 3: Wizualizacja tras wygenerowanych przez CPLEX



Rysunek 4: Wizualizacja tras wygenerowanych przez algorytm genetyczny

7.4 Wnioski

- Dla małych instancji problemu (do 100 punktów) solver CPLEX zapewnia lepsze wyniki pod względem minimalizacji całkowitego dystansu
- Algorytm genetyczny oferuje bardziej zrównoważone rozłożenie zadań między pojazdami, co może być istotne w praktycznych zastosowaniach
- Czas obliczeń obu metod jest akceptowalny dla zastosowań biznesowych

7.5 Dalsze badania

- Badanie wpływu parametrów algorytmu genetycznego na jakość rozwiązania
- Testy wydajnościowe dla większych instancji problemu
- Integracja z rzeczywistymi danymi GPS
- Uwzględnienie dynamicznych zmian w ruchu drogowym

Uwagi:

- Wyniki mogą różnić się w zależności od konfiguracji sprzętowej
- Dla dokładniejszych analiz zaleca się wykonanie większej liczby testów
- W przypadku problemów z dużą liczbą ograniczeń czasowych wyniki mogą się różnić

7.6 Porównanie metod

- **CPLEX**: Znajduje optymalne rozwiązania dla małych instancji (do 100 punktów), ale wymaga znacznych zasobów obliczeniowych.
- **Algorytm genetyczny**: Działa szybciej dla większych instancji, ale może dawać suboptymalne rozwiązania.

7.7 Wizualizacja

Wyniki zostały przedstawione na interaktywnych mapach (Folium) oraz statycznych wykresach (Matplotlib), co ułatwia analizę tras i ich porównanie.

7.8 Charakterystyka rozwiązań

Tabela 2: Porównanie charakterystyk rozwiązań

Parametr	**CPLEX**	**Algorytm genetyczny**
Liczba kursantów	1	4
Całkowity dystans	66.55 km	244.07 km
Czas obliczeń	0.02 s	0.05 s
Typ rozwiązania	Globalnie optymalne	Suboptymalne

7.9 Przyczyny różnic w liczbie kurierów

7.9.1 Natura solvera CPLEX

- **Optymalizacja kosztu**: CPLEX dąży do ścisłej minimalizacji funkcji celu (całkowitego dystansu), co w tym przypadku prowadzi do wykorzystania tylko jednego pojazdu
- **Uwzględnienie wszystkich ograniczeń**: Pomimo posiadania wielu pojazdów, solver znalazł rozwiązanie spełniające wszystkie warunki (czasowe, ładowności) przy użyciu jednego kuriera

- **Pełna przeszukiwanie przestrzeni rozwiązań**: CPLEX analizuje wszystkie możliwe kombinacje, znajdując globalny optimum

7.9.2 Charakterystyka algorytmu genetycznego

- **Heurystyczne podejście**: Algorytm genetyczny dzieli problem na podproblemy, co naturalnie prowadzi do rozłożenia zadań między wielu kurierów
- **Ograniczona eksploracja**: Ze względu na skończoną liczbę generacji i osobników, algorytm może nie znaleźć rozwiązania z jednym kurierem
- **Specyfika operatorów genetycznych**: Krzyżowanie i mutacja preferują rozwiązania z większą liczbą tras
- **Funkcja fitness**: Może nie uwzględniać w pełni kosztów związanych z dodatkowymi kurierami

7.10 Matematyczne uzasadnienie

Rozważmy następujące parametry problemu:

- Ładowność pojazdu: $Q = 500$ kg
- Łączne zapotrzebowanie: $\sum q_i = 480$ kg
- Okna czasowe pozwalają na realizację przez jednego kuriera.

Dla CPLEX:

$$\min \sum_{k \in K} \sum_{i,j} d_{ij} x_{ijk} \quad \text{przy} \quad \sum_{i,j} q_j x_{ijk} \leq Q$$

Znalezione rozwiązanie z $K = 1$ jest dopuszczalne i optymalne.

Dla algorytmu genetycznego:

- Populacja inicjalizowana z wieloma trasami
- Operator krzyżowania utrudnia konsolidację do jednej trasy
- Lokalne optimum z wieloma kurierami.

7.11 Wnioski praktyczne

- **CPLEX** preferuje rozwiązania z minimalną liczbą pojazdów, gdy jest to możliwe
- **Algorytm genetyczny** może generować rozwiązania z większą liczbą kurierów ze względu na:
  - Dzielenie problemu na podproblemy
  - Mniejszą presję na konsolidację tras
  - Ograniczenia eksploracyjne
- W rzeczywistych zastosowaniach rozwiązanie CPLEX może być niepraktyczne (przeciążenie jednego kuriera.)

7.12 Rekomendacje

- Dla **minimalizacji kosztów**: CPLEX (1 kurier)
- Dla **równoważenia obciążenia**: algorytm genetyczny (4 kurierów)
- Możliwe modyfikacje:
  - W CPLEX: dodać ograniczenie maksymalnego czasu pracy kuriera.
  - W GA: dostosować funkcję fitness do karania nadmiernej liczby kurierów.

8 Podsumowanie

8.1 Szczegółowe wnioski z porównania metod optymalizacji

8.1.1 Wnioski dotyczące solvera CPLEX

- **Efektywność kosztowa**: Rozwiązanie CPLEX zapewnia o 72.7% mniejszy całkowity dystans (66.55 km vs 244.07 km), co przekłada się na znaczące oszczędności paliwa i kosztów eksploatacji.
- **Wydajność obliczeniowa**: Czas rozwiązania wynoszący 0.02 sekundy potwierdza wysoką efektywność CPLEX dla problemów o tej skali.
- **Ograniczenia praktyczne**: Jedna długa trasa (125 minut) może być niepraktyczna z uwagi na:
  - Limit czasu pracy kierowców
  - Wymagania dotyczące terminowości dostaw
  - Zmęczenie kierowcy wpływające na bezpieczeństwo
- **Optymalność rozwiązania**: CPLEX gwarantuje znalezienie rozwiązania globalnie optymalnego dla zadanego modelu matematycznego.

8.1.2 Wnioski dotyczące algorytmu genetycznego

- **Rozłożenie obciążenia**: 4 równoległe trasy (średnio 85 minut każda) pozwalają na:
  - Lepsze wykorzystanie floty pojazdów
  - Redukcję ryzyka opóźnień
  - Większą elastyczność w przypadku awarii
- **Koszty operacyjne**: 3.7-krotnie większy całkowity dystans generuje wyższe koszty, ale:
  - Pozwala uniknąć nadgodzin kierowców

  - Zapewnia rezerwę czasową na nieprzewidziane zdarzenia

- **Skalowalność**: Algorytm genetyczny, pomimo dłuższego czasu obliczeń (0.05s), lepiej nadaje się dla:
  - Większych instancji problemu (>100 punktów dostaw)
  - Problemów z dodatkowymi ograniczeniami praktycznymi

8.1.3 Wnioski praktyczne dla logistyki

1. **Dla priorytetu kosztowego**:
   - CPLEX jest rozwiązaniem preferowanym
   - Wymaga jednak weryfikacji pod kątem przepisów pracy
   - Może wymagać modyfikacji modelu (np. dodanie limitów czasu trasy)
2. **Dla priorytetu niezawodności**:
   - Algorytm genetyczny zapewnia bardziej odporne rozwiązanie
   - Pozwala na łatwe uwzględnienie dodatkowych ograniczeń
   - Wiąże się z wyższymi kosztami operacyjnymi
3. **Kompromisowe rozwiązanie**:
   - Możliwość hybrydowego podejścia - CPLEX dla głównej optymalizacji z późniejszym podziałem tras
   - Użycie zmodyfikowanej funkcji celu w algorytmie genetycznym uwzględniającej zarówno dystans jak i liczbę tras

8.1.4 Kierunki dalszych badań

- Badanie wpływu różnych parametrów algorytmu genetycznego na jakość rozwiązania
- Integracja dynamicznych danych o ruchu drogowym
- Uwzględnienie zmiennych kosztów dla różnych typów pojazdów
- Testy wydajnościowe dla większych instancji problemu

Literatura

[1] Toth, P., & Vigo, D. (2002). *The Vehicle Routing Problem*. SIAM.

[2] IBM. (2021). *CPLEX Optimization Studio Documentation*.