Rozwiązanie problemu komiwojażera z wykorzystaniem algorytmu genetycznego

Opis Techniczny Projektu

Tomasz Kawiak Piotr Karaś Mateusz Mazur

07.11.2024

Spis treści

1 Przegląd projektu

Temat: Rozwiązanie problemu komiwojażera z wykorzystaniem algorytmu genetycznego

Cele:

- Opracowanie bliskiego optimum rozwiązania problemu komiwojażera
- Implementacja algorytmu w Pythonie przy użyciu PyGAD i wizualizacja wyników
- Ocena i walidacja rozwiązania, porównanie z innymi metodami optymalizacji

Stos technologiczny: Python, PyGAD

2 Opis Techniczny

2.1 Struktura Kodu

Kod projektu składa się z następujących głównych elementów:

- **Klasa `TravelGraph`**:
  - Odpowiada za zarządzanie miastami oraz implementację metod związanych z wyszukiwaniem najkrótszej ścieżki.
- **Metody algorytmu genetycznego**:
  - `pygad.GA` jest używany do przeprowadzenia procesu optymalizacji.
- **Funkcje pomocnicze**:
  - `generate_random_distance_matrix` generuje macierz odległości
  - `main` obsługuje główną logikę wywoływania algorytmu.

2.2 Szczegóły Techniczne i Algorytmiczne

2.2.1 Reprezentacja Danych

- **Lista Miast (`nodes`)**:
  - Zawiera listę par x i y, które symbolizują miasta odwiedzane w ramach ścieżki podróży.

2.2.2 Algorytm Genetyczny

Parametry mogą ulec zmianie w czasie rozwoju projektu, w zależności od optymalizacji i testów.

- **Inicjalizacja**:
  - Tworzona jest populacja rozwiązań reprezentujących różne ścieżki między miastami.
- **Funkcja Dopasowania (`fitness_function`)**:
  - Oblicza sumaryczną odległość dla danej ścieżki (rozwiązania). Im mniejsza odległość, tym lepsze rozwiązanie (wartość minimalizowana).
- **Selekcja Rodziców**:
  - Typ selekcji *SSS* (Steady-State Selection) pozwala wybrać najlepszych rodziców z każdej generacji, którzy są przekazywani do następnych pokoleń.
- **Krzyżowanie**:
  - Wybór krzyżowania jednopunktowego (`single_point`) pozwala losowo łączyć sekwencje genów dwóch rodziców w celu stworzenia nowych rozwiązań.
- **Mutacja**: `mutation_type=żandom"`
  - Modyfikuje losowo wybrane geny w populacji potomków, co zapobiega wpadaniu algorytmu w lokalne minima.

2.3 Parametry Algorytmu i Optymalizacja

- **Liczba Generacji (`num_generations`)**:
  - Określa maksymalną liczbę iteracji, przez które przechodzi algorytm.
- **Wielkość Populacji (`sol_per_pop`)**:
  - Odpowiada za liczbę potencjalnych rozwiązań w każdej generacji.

- **Przestrzeń Genów (`gene_space`)**:
  - Zapewnia, że geny przyjmują wartości tylko w zakresie dostępnych indeksów miast, bez powtórzeń w ścieżce.

2.4 Wynik i Analiza Rozwiązania

Algorytm kończy swoje działanie, zwracając:

- **Najlepszą Ścieżkę**:
  - Lista indeksów podanych miast.
- **Odległość Najlepszej Ścieżki**:
  - Całkowita długość tej trasy.

2.5 Przykładowe Uruchomienie

Po uruchomieniu programu, wywołana zostaje funkcja `main`, która: - Odczytuje miasta z pliku .tsp - Tworzy instancję klasy `TravelGraph`. - Wywołuje metodę `find_shortest_path`, która uruchamia algorytm genetyczny i wyświetla najkrótszą znalezioną ścieżkę oraz jej odległość.