Rozwiązanie problemu komiwojażera z wykorzystaniem algorytmu genetycznego

Opis Techniczny Projektu

Tomasz Kawiak Piotr Karaś Mateusz Mazur

01.12.2024

Spis treści

1 Przegląd projektu

Temat: Rozwiązanie problemu komiwojażera z wykorzystaniem algorytmu genetycznego

Cele:

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

Stos technologiczny: Python, Numpy, Matplotlib

2 Opis Techniczny

2.1 Struktura Kodu

Kod projektu znajduje się w folderze `src` i składa się z następujących głównych elementów:

- **Klasa `TravelGraph`** znajdująca się w pliku `travel_graph.py`:
  - Odpowiada za przeprowadzenie algorytmu genetycznego.
  - Przeprowadza inicjalizację, selekcję, krzyżowanie, mutację, dywersyfikację
  - Przechowuje najlepszą ścieżkę oraz jej wyniki, a także dane dotyczące przebiegu algorytmu.
- **Elementy algorytmu genetycznego** znajdujące się w plikach w folderze `ga_utils`:
  - `crossover_functions.py` - zawiera funkcje krzyżowania.
  - `distance_functions.py` - zawiera funkcje obliczające odległość między miastami.
  - `diversification_functions.py` - zawiera funkcje zwiększające różnorodność populacji.
  - `fitness_functions.py` - zawiera funkcje oceny przystosowania.
  - `mutation_functions.py` - zawiera funkcje mutacji.
  - `selection_functions.py` - zawiera funkcje selekcji.
- **Funkcje pomocnicze** znajdujące się w plikach:
  - `driver.py` - obsługują główną logikę wywoływania algorytmu.
  - `genetic_experiment_conductor.py` - przeprowadzają eksperymenty z różnymi parametrami algorytmu genetycznego.
  - `plotter.py` - odpowiadają za wizualizację wyników.
  - `solution_exporter.py` - eksportują wyniki do plików.
  - `parser/*` - parsują pliki z modelami miast.

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.
  - Lub, w przypadku plików określających odległości między miastami macierzą odległości, lista indeksów miast.

2.2.2 Algorytm Genetyczny

- **Inicjalizacja**. Tworzenie populacji rozwiązań reprezentujących różne ścieżki między miastami.
- **Funkcja Dopasowania (`fitness_function`)**. Obliczanie sumarycznej odległość dla danej ścieżki (rozwiązania).

  - `fitness_classic` - oblicza sumę odległości między miastami. Im mniejsza odległość, tym lepsze rozwiązanie (wartość minimalizowana).
- **Selekcja Rodziców**. Wybór rodziców wybraną metoda selekcji:
  - `SelectionElitism` - wybiera najlepsze rozwiązania z populacji.
  - `SelectionRouletteWheel` - wybiera rozwiązania na podstawie ich przystosowania.
  - `SelectionTournament` - wybiera rozwiązania metodą turniejową.
- **Krzyżowanie**. Tworzenie potomstwa na podstawie wybranego rodzaju krzyżowania:
  - `CrossoverGenesPMX` - krzyżowanie PMX.
  - `CrossoverGenesEdgeRecombination` - krzyżowanie metodą rekombinacji krawędzi.
- **Mutacja**:. Mutowanie potomstwa wybraną metodą mutacji:
  - `MutationGenePerCity` - mutacja każdego genu z określonym prawdopodobieństwem.
  - `MutationGeneDisplacement` - mutacja przemieszczenia podciągu genów.
  - `MutationRandomMutation` - losowa mutacja z wybranych.
- **Dywersyfikacja**. Zwiększanie różnorodności populacji:
  - `diversification_random` - zwiększa różnorodność populacji poprzez dodanie losowych rozwiązań.
  - `diversification_roulette_wheel` - zwiększa różnorodność populacji poprzez dodanie rozwiązań wybranych metodą ruletki.
- Zastępowanie starej populacji nową i powtarzanie procesu dla kolejnych generacji.

2.3 Parametry Algorytmu i Optymalizacja

Algorytm jest elastyczny i łatwy w modyfikacji, w tym oferuje proste określanie parametrów wybranych operacji genetycznych (m.in. dzięki zastosowaniu podejścia OOP).

Uruchamianie algorytmu odbywa się poprzez wywołanie funkcji `find_shortest_path` klasy `TravelGraph` z odpowiednimi parametrami:

- Inicjalizacja obiektu klasy `TravelGraph` podając następujące parametry:
  - `nodes` - lista miast.
  - `distance_function` - funkcja obliczająca odległość między miastami.
  - `fitness_function` - funkcja oceny przystosowania.
  - `selection` - obiekt klasy selekcji.
  - `diversification_function` - funkcja zwiększająca różnorodność populacji.
  - `crossover` - obiekt klasy krzyżowania.
  - `mutation` - obiekt klasy mutacji.

- Wywołanie metody `find_shortest_path` klasy `TravelGraph` z następującymi parametrami:
  - `population_size` - wielkość populacji.
  - `generations` - liczba generacji.
  - `diversity_factor` - ułamek populacji, który ma zostać wygenerowany z użyciem funkcji zwiększającej różnorodność.
  - `diversity_factor_change` - zmiana wartości `diversity_factor` w kolejnych generacjach.
  - `patience` - liczba generacji bez poprawy, po której algorytm zakończy działanie.
  - `patience_factor` - współczynnik określający minimalną różnicę między najlepszymi rozwiązaniami w populacji, aby uznać, że doszło do poprawy.
  - `verbose` - tryb wyświetlania informacje o postępie algorytmu.

2.4 Wynik i Analiza Rozwiązania

Algorytm kończy swoje działanie po osiągnięciu maksymalnej liczby generacji lub po przekroczeniu wartości `patience`. Dostęp do wyników i analizy rozwiązania uzyskujemy poprzez następujące metody klasy `TravelGraph`.

- `get_solution()` - zwraca najlepsze rozwiązanie.
- `get_fitness()` - zwraca wartość funkcji przystosowania dla najlepszego rozwiązania.
- `get_convergence()` - zwraca listę wartości funkcji przystosowania w kolejnych generacjach.

2.5 Przeprowadzanie eksperymentów

Proces przeprowadzenia eksperymentu z wykorzystaniem algorytmu genetycznego realizuje funkcja `run_genetic_experiment` znajdująca się wp pliku `genetic_experiment_conductor.py`. Proces ten składa się z następujących kroków:

- Wczytanie miast i optymalnego rozwiązania z pliku .tsp na bazie argumentu `problem_name`. Na ich bazie zdefiniowanie listy miast `nodes` oraz funkcji obliczającej odległość między nimi `distance_function`.
- Wizualizacja miast oraz optymalnego rozwiązania problemu (poprzez zapis do pliku).
- Stworzenie obiektu klasy solvera na bazie argumentu `travel_graph_class` (domyślnie `TravelGraph`) oraz przekazanie mu odpowiednich parametrów (`nodes`, `distance_function`, `selection`, `diversification_function`, `crossover`, `mutation`).
- Uruchomienie algorytmu genetycznego z wykorzystaniem metody `find_shortest_path` z odpowiednimi parametrami (wszystkimi pozostałymi, które nie zostały zdefiniowane w kroku 2, a zostały przekazane jako argumenty funkcji).

- Zmierzenie czasu wykonania algorytmu.
- Zapis wyników do pliku.
- Wizualizacja wyników (poprzez zapis do pliku).
- Wizualizacja zbieżności algorytmu (poprzez zapis do pliku).

2.6 Uruchomienie Programu

Po uruchomieniu programu, wywołana zostaje funkcja `main`, która:

- Pyta o wybór nazwy problemu.
- Definiuje parametry algorytmu genetycznego,
- Tworzy obiekty klas reprezentujących operacje genetyczne na bazie wybranych parametrów.
- Uruchamia funkcję `run_genetic_experiment` przeprowadzając eksperyment z wybranymi parametrami na wybranym problemie.