Projekt: Problem Komiwojażera

Algorytm Genetyczny + Wizualizacja Streamlit

Cel projektu

Celem projektu jest zaimplementowanie algorytmu genetycznego rozwiązującego problem komiwojażera (TSP - Traveling Salesman Problem) oraz stworzenie interaktywnej aplikacji webowej w Streamlit umożliwiającej:

- Interaktywną modyfikację parametrów algorytmu
- Wizualizację najlepszej trasy w czasie rzeczywistym
- Obserwację zbieżności algorytmu na wykresie
- Porównanie różnych konfiguracji
- Eksport wyników

Opis problemu TSP

Problem komiwojażera (Traveling Salesman Problem) polega na znalezieniu najkrótszej trasy przechodzącej przez wszystkie miasta dokładnie raz i powracającej do miasta startowego.

Jest to problem NP-trudny, co oznacza że dla dużej liczby miast znalezienie optymalnego rozwiązania metodami dokładnymi jest praktycznie niemożliwe. Algorytm genetyczny jest heurystyką, która pozwala znaleźć bardzo dobre (choć nie zawsze optymalne) rozwiązanie w rozsądnym czasie.

Datasety do wyboru

Projekt można zrealizować na jednym z następujących zbiorów danych:

1. **ATT48** - 48 stolic stanów USA
   - Rozmiar: 48 miast
   - Znane optimum: 10628
   - Typ odległości: ATT (pseudo-euklidesowa)
2. **Berlin52** - 52 lokacje w Berlinie
   - Rozmiar: 52 miasta
   - Znane optimum: 7542
   - Typ odległości: EUC_2D (euklidesowa)

Źródło danych: TSPLIB - klasyczna biblioteka problemów TSP
Link do pobrania: `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/`
Można też użyć biblioteki Python `tsplib95`:

```
pip install tsplib95
```

Wymagania projektu

Część 1: Implementacja algorytmu genetycznego

Algorytm genetyczny musi zawierać następujące komponenty:

1. **Reprezentacja chromosomu**
   - Permutacja numerów miast: [0, 5, 12, 3, 8, ...]
   - Każde miasto występuje dokładnie raz
2. **Funkcja fitness**
   - Oblicza całkowitą długość trasy
   - Im krótsza trasa, tym lepszy osobnik
3. **Operatory selekcji** (minimum 2 metody)
   - Tournament selection (selekcja turniejowa)
   - Roulette wheel selection (selekcja ruletki)
   - Rank selection (selekcja rangowa - opcjonalnie)
4. **Operatory krzyżowania** (minimum 2 metody)
   - OX (Order Crossover)
   - PMX (Partially Mapped Crossover)
   - Edge Recombination (opcjonalnie)
5. **Operatory mutacji** (minimum 2 metody)
   - Swap mutation (zamiana 2 miast)
   - Inversion mutation (odwrócenie fragmentu)
   - Scramble mutation (losowe przestawienie fragmentu - opcjonalnie)
6. **Mechanizm elitaryzmu**
   - Zachowanie najlepszych osobników w następnej generacji
   - Opcjonalnie: adaptacyjne prawdopodobieństwa mutacji

Część 2: Aplikacja Streamlit

Aplikacja musi zawierać:

1. **Sidebar z parametrami:**
   - Rozmiar populacji (slider: 50-500)

- Liczba generacji (slider: 100-2000)
- Prawdopodobieństwo mutacji (slider: 0.01-0.2)
- Prawdopodobieństwo krzyżowania (slider: 0.5-1.0)
- Wybór datasetu (ATT48 / Berlin52)
- Wybór metody selekcji (dropdown)
- Wybór metody krzyżowania (dropdown)
- Wybór metody mutacji (dropdown)
- Rozmiar elity (slider: 0-20% populacji)
- Przycisk START

2. **Główny obszar aplikacji:**
   - **Dwie kolumny obok siebie:**
     - Lewa: Mapa najlepszej trasy (aktualizacja na żywo)
     - Prawa: Wykres zbieżności (aktualizacja na żywo)
   - **Metryki w czasie rzeczywistym:**
     - Aktualna najlepsza odległość
     - Różnica od optimum
     - Numer generacji
     - Czas wykonania
     - Poprawa względem startu (%)
   - **Progress bar** pokazujący postęp
   - **Status text** z informacjami
3. **Funkcje dodatkowe:**
   - Podsumowanie końcowe po zakończeniu
   - Historia porównań (opcjonalnie)

Część 3: Sprawozdanie

Projekt musi zawierać sprawozdanie PDF (8-12 stron) z następującymi sekcjami:

1. **Strona tytułowa**
   - Tytuł projektu
   - Imię i nazwisko autora/autorów
   - Numer indeksu
   - Data wykonania
   - **Jeśli projekt w zespole:** koniecznie zaznaczyć wszystkie osoby (do 4 osób)
2. **Wprowadzenie**
   - Opis problemu TSP
   - Dlaczego TSP jest trudny (złożoność NP)
   - Zastosowania praktyczne TSP
3. **Implementacja**

- Opis reprezentacji chromosomu
- Funkcja fitness
- Szczegółowy opis każdego zaimplementowanego operatora

4. **Aplikacja Streamlit**
   - Architektura aplikacji
   - Opis funkcjonalności
   - Screenshoty interfejsu
5. **Eksperymenty i wyniki**
   - Przetestowane konfiguracje (minimum 5)
   - Tabela z wynikami
   - Wykresy porównawcze
   - Analiza wpływu parametrów na wyniki
6. **Wnioski**
   - Najlepsza znaleziona konfiguracja
   - Które operatory działały najlepiej
   - Trudności napotkane podczas implementacji
   - Możliwe ulepszenia
   - Wnioski końcowe
7. **Bibliografia**
   - Źródła o TSP
   - Źródła o algorytmach genetycznych
   - Dokumentacja bibliotek

Struktura projektu

Projekt powinien mieć następującą strukturę katalogów:

```
tsp_genetic_algorithm/
|
|-- app.py                    # Glowna aplikacja Streamlit
|-- genetic_algorithm.py      # Klasa GeneticAlgorithm
|-- tsp_problem.py            # Klasa TSP, wczytywanie danych
|-- operators.py              # Operatory genetyczne
|-- visualization.py          # Funkcje do wykresow
|-- utils.py                  # Funkcje pomocnicze
|
|-- data/
|   |-- att48.tsp             # Dataset ATT48
|   |-- berlin52.tsp          # Dataset Berlin52
|
|-- results/
|   |-- best_route.json       # Najlepsza znaleziona trasa
```

```
|   |-- experiments.csv      # Wyniki eksperymentow
|   |-- screenshots/         # Screenshoty aplikacji
|
|-- requirements.txt         # Zaleznosci Python
|-- README.md               # Instrukcja uruchomienia
|-- sprawozdanie.pdf        # Sprawozdanie koncowe
```

Wskazówki implementacyjne

Wczytywanie danych TSPLIB

Pliki TSPLIB mają specyficzny format. Należy:

- Parsować nagłówek (NAME, DIMENSION, EDGE_WEIGHT_TYPE)
- Wczytać współrzędne z sekcji NODE_COORD_SECTION
- Obliczyć macierz odległości zgodnie z typem (EUC_2D lub ATT)

Funkcje odległości:

- **EUC_2D:** Standardowa odległość euklidesowa zaokrąglona do najbliższej liczby całkowitej
- **ATT:** Pseudo-euklidesowa używana w ATT48, szczególny wzór opisany w dokumentacji TSPLIB

Operatory genetyczne dla TSP

Uwaga: TSP wymaga specjalnych operatorów zachowujących permutację!

- **Order Crossover (OX):** Kopiuje fragment z jednego rodzica, resztę uzupełnia w kolejności z drugiego rodzica
- **Partially Mapped Crossover (PMX):** Wymienia fragmenty między rodzicami z zachowaniem mapowania
- **Swap mutation:** Zamienia miejscami dwa losowe miasta
- **Inversion mutation:** Odwraca kolejność miast w losowym fragmencie trasy

Wizualizacja w Streamlit

Ważne techniki:

- `st.empty()` - tworzenie placeholder'ów do aktualizacji na żywo
- `st.columns()` - podział ekranu na kolumny
- `st.progress()` - progress bar
- `plt.close(fig)` - ważne! Zamykanie figur by uniknąć wycieku pamięci
- Aktualizacja co 10-20 generacji dla płynności

Typowe błędy do uniknięcia

1. **Niepoprawna permutacja po krzyżowaniu/mutacji**
   - Sprawdź czy każde miasto występuje dokładnie raz
   - Testuj operatory osobno przed użyciem w GA
2. **Brak zamknięcia pętli w obliczaniu dystansu**
   - Pamiętaj o dodaniu odległości z ostatniego miasta do pierwszego
3. **Za rzadkie lub za częste aktualizacje w Streamlit**
   - Update co 10-20 generacji jest optymalny
   - Za częste = wolne, za rzadkie = brak wrażenia "live"
4. **Memory leak w matplotlib**
   - Zawsze wywołuj `plt.close(fig)` po `st.pyplot(fig)`
5. **Selekcja bez uwzględnienia fitness**
   - W TSP mniejsza wartość = lepiej (minimalizacja!)
   - Selekcja ruletki wymaga odwrócenia lub przekształcenia fitness
6. **Brak elitaryzmu**
   - Bez elitaryzmu można stracić najlepsze rozwiązanie
   - Zalecane: 5-10% najlepszych osobników przechodzi bez zmian

Dodatkowe funkcje (opcjonalne)

Opcjonalne rozszerzenia projektu dla chętnych:

1. **Adaptacyjne prawdopodobieństwa**
   - Prawdopodobieństwo mutacji zmienia się w trakcie ewolucji
   - Na początku wysokie (eksploracja), później niskie (eksploatacja)
2. **Local search (2-opt)**
   - Zastosowanie lokalnego przeszukiwania do ulepszania rozwiązań
   - Znacząco poprawia wyniki końcowe
3. **Animacja GIF**
   - Zapisanie ewolucji trasy jako animowany GIF
   - Możliwość obejrzenia całego procesu ewolucji
4. **Testy jednostkowe**
   - Testy dla operatorów genetycznych
   - Użycie pytest lub unittest
   - Weryfikacja poprawności permutacji
5. **Wizualizacja różnorodności populacji**
   - Wykres pokazujący jak zmienia się różnorodność w czasie
   - Pomaga zrozumieć konwergencję algorytmu

Przykładowe wyniki do osiągnięcia

ATT48 (optimum: 10628)

- **Podstawowa implementacja:** 11000-12000
- **Po tuningu parametrów:** 10700-11000
- **Bardzo dobry wynik:** 10628-10800
- **Z 2-opt local search:** często dokładne optimum 10628

Berlin52 (optimum: 7542)

- **Podstawowa implementacja:** 8000-8500
- **Po tuningu parametrów:** 7600-7800
- **Bardzo dobry wynik:** 7542-7650

Uwaga: Osiągnięcie dokładnego optimum nie jest wymagane! Ważniejsza jest jakość implementacji i przeprowadzona analiza.

Praca w zespołach

- Projekt można realizować **indywidualnie** lub **w zespołach** (maksymalnie 4 osoby)
- W przypadku pracy w zespole:
  - **Koniecznie zaznaczyć wszystkie osoby** na stronie tytułowej sprawozdania
  - Wszystkie osoby muszą podać imię, nazwisko i numer indeksu
  - Wystarczy napisać jedno sprawozdanie
  - **Wszystkie osoby muszą osobno oddać projekt na platformie Moodle**
  - W komentarzu przy oddawaniu zaznaczyć: "Projekt w zespole: [imiona i nazwiska wszystkich członków]"

Wymagania techniczne

Instalacja zależności

requirements.txt:

```
streamlit>=1.29.0
numpy>=1.24.3
matplotlib>=3.7.2
pandas>=2.0.3
tsplib95>=0.7.1
```

Instalacja:

```
pip install -r requirements.txt
```

Uruchomienie aplikacji

```
streamlit run app.py
```

Aplikacja otworzy się automatycznie w przeglądarce pod adresem `http://localhost:8501`

Pobranie danych

Opcja 1: Ręczne pobranie

```
# ATT48
wget http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
    tsp/att48.tsp.gz
gunzip att48.tsp.gz
mv att48.tsp data/

# Berlin52
wget http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
    tsp/berlin52.tsp.gz
gunzip berlin52.tsp.gz
mv berlin52.tsp data/
```

Opcja 2: Użycie biblioteki tsplib95

Biblioteka `tsplib95` pozwala na automatyczne ładowanie i parsowanie problemów TSP.

Format oddania projektu

Pliki do oddania

Projekt należy oddać jako **archiwum ZIP** zawierające:

1. Wszystkie pliki źródłowe projektu (.py)
2. Dane (att48.tsp lub berlin52.tsp)
3. **Sprawozdanie w formacie PDF**
4. **Wypełniony formularz od dydaktyka**

Konwencja nazewnictwa

Archiwum ZIP:

```
nazwisko_imie_nrindeksu_tsp_ga.zip
```

Przykład:

```
kowalski_jan_123456_tsp_ga.zip
```

Dla zespołu (2-4 osoby):

Należy wymienić wszystkie osoby w nazwie pliku, oddzielone podkreślnikami:

`nazwisko1_nazwisko2_nrindeksu1_nrindeksu2_tsp_ga.zip`

Przykłady:
Zespół 2-osobowy:

`kowalski_nowak_123456_234567_tsp_ga.zip`

Zespół 3-osobowy:

`kowalski_nowak_malinowski_123456_234567_345678_tsp_ga.zip`

Zespół 4-osobowy:

`kowalski_nowak_malinowski_wisniewski_123456_234567_345678_456789_tsp_ga.zip`

Formularz dydaktyka

Koniecznie należy wypełnić załączony przez dydaktyka formularz i dołączyć go do archiwum ZIP.

Ocena projektu

Projekt oceniany jest za:

- **Poprawność implementacji** - czy algorytm działa zgodnie z wymaganiami
- **Jakość kodu** - czytelność, organizacja, komentarze
- **Funkcjonalność aplikacji Streamlit** - czy wszystkie wymagane elementy są zaimplementowane
- **Sprawozdanie** - kompletność, analiza, wnioski
- **Eksperymenty** - różnorodność testów, analiza wyników
- **Całokształt projektu** - spójność wszystkich elementów
- **Terminowość** - oddanie w wyznaczonym terminie

Pytania i odpowiedzi (FAQ)

Pytanie: Czy mogę użyć gotowej biblioteki do GA?

Odpowiedź: Nie. Należy zaimplementować algorytm genetyczny samodzielnie. Można używać standardowych bibliotek (numpy, matplotlib, pandas), ale nie bibliotek implementujących GA (jak DEAP, PyGAD).

Pytanie: Czy mogę użyć innego datasetu niż ATT48/Berlin52?

Odpowiedź: Tak, ale musi to być dataset z TSPLIB i musi mieć znane optimum. Wybór innego datasetu należy skonsultować z prowadzącym.

Pytanie: Jak często powinien się aktualizować wykres w Streamlit?

Odpowiedź: Zaleca się co 10-20 generacji. Za częste aktualizacje spowalniają aplikację, za rzadkie zmniejszają efekt wizualizacji "na żywo".

Pytanie: Co jeśli nie osiągnę optimum?

Odpowiedź: To normalne! Algorytm genetyczny jest heurystyką i nie gwarantuje znalezienia optimum. Ważniejsza jest jakość implementacji, przeprowadzona analiza i wyciągnięte wnioski.

Pytanie: Czy mogę pracować w zespole?

Odpowiedź: Tak! Projekt można realizować indywidualnie lub w zespole (maksymalnie 4 osoby). W przypadku pracy w zespole:

- Koniecznie zaznaczyć wszystkie osoby w sprawozdaniu
- Wszystkie osoby muszą osobno oddać projekt na Moodle
- Jeden wspólny kod i jedno sprawozdanie wystarczy
- Nazwa pliku ZIP musi zawierać wszystkie nazwiska i numery indeksów

Pytanie: Ile eksperymentów powinienem przeprowadzić?

Odpowiedź: Minimum 5 różnych konfiguracji parametrów. Więcej eksperymentów pozwoli na lepszą analizę wpływu parametrów na wyniki.

Pytanie: Czy muszę załączyć formularz?

Odpowiedź: Tak, wypełniony formularz od dydaktyka jest **obowiązkowy** i musi być załączony do archiwum ZIP.

Pytanie: Co z terminem oddania?

Odpowiedź: Terminowość jest brana pod uwagę w ocenie. Dokładny termin zostanie podany przez prowadzącego.

Zasoby dodatkowe

Literatura

1. **Algorytmy genetyczne:**
   - D.E. Goldberg - "Algorytmy genetyczne i ich zastosowania"
   - M. Mitchell - "An Introduction to Genetic Algorithms"
   - Z. Michalewicz - "Algorytmy genetyczne + struktury danych = programy ewolucyjne"
2. **Problem TSP:**
   - Gutin, Punnen - "The Traveling Salesman Problem and Its Variations"
   - Applegate et al. - "The Traveling Salesman Problem: A Computational Study"

3. **Operatory dla TSP:**
    - Larranaga et al. (1999) - "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators"

Linki

- TSPLIB: `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`
- Streamlit docs: `https://docs.streamlit.io`
- tsplib95 library: `https://pypi.org/project/tsplib95/`
- Matplotlib docs: `https://matplotlib.org/`
- NumPy docs: `https://numpy.org/doc/`