Katarzyna Kanicka, 331388

WSI Sprawozdanie 2 – Algorytmy ewolucyjne

Treść polecenia:

Zaimplementować algorytm ewolucyjny z mutacją, selekcją ruletkową, krzyżowaniem oraz sukcesją generacyjną.

Użyć zaimplementowany algorytm do wyznaczenia najbliższej trasy w problemie komiwojażera na zbiorze polskich miast (data/cities.csv).

Znaleźć ustawienie hiperparametrów algorytmu, które zwraca przyzwoite wyniki.

Zbadać wpływ następującego hiperparametru na proces optymalizacji (jak dobre rozwiązanie zwraca algorytm, jak szybko algorytm dochodzi do dobrego rozwiązania):

- prawdopodobieństwo krzyżowania

Wypisać najlepszą znalezioną przez algorytm trasę. W sprawozdaniu zawrzeć zrzut ekranu z wizualizacji trasy na mapie Polski używając np. Google Maps

Analiza i wstępne założenia badań

Ustalona para miast (początek i koniec) to : Siedlce -> Warszawa.

Użyta implementacja mutacji

- **swap mutation** - zmienia rozwiązanie poprzez losowy wybór dwóch pozycji w reprezentacji i zamianę ich wartości.

Użyta implementacja krzyżowania

- **order crossover** – potomek przyjmuje zakres allei od pierwszego rodzica, a pozostałe wartości są umieszczane w kolejności w jakiej pojawiają się u drugiego rodzica.

Instrukcja do używania skryptu

Po pobraniu repozytorium należy:

- przejść do głownego katalogu – lab1

- `$ cd lab2`
- stworzyć środowisko wirtualne i pobrać requirements.txt

```
$ python3 -m venv .venv
$ source .venv/bin/activate
$ pip install -r requirements.txt
```

- uruchomić plik main_gd.py, w terminalu za pomocą:

```
$ python3 main.py [argumenty]
```

Możliwe argumenty:

- **cities_path** – ścieżka to cities.csv
- **problem-size** – uruchom algorytm na pełnej lub uproszczonej konfiguracji problemu
- **start** – miasto początkowe
- **finisz** – miasto końcowe
- **seed** – ziarno
- **visualize** – generowanie wykresu zmiany najlepszego rozwiązania na przestrzeni pokoleń
- **plot_filename** – nazwa pliku generowanego wykresu
- **run_tests** - generuje tabele ze średnimi najlepszymi parametrami oraz wpływ prawdopodobieństwa krzyżowania finalne rozwiązanie
- **find_solution** – przeprowadza algorytm podaną ilość razy i wybiera najlepsze z otrzymanych rozwiązań

Dodatkowo w pliku ‘config.json’ znajdują się parametry algorytmu: wielkość populacji, prawodobodobieństwo mutacji, krzyżowania oraz limit pokoleń.

Wyniki analiz w formie tabel i wykresów

Średnie najlepsze parametry algorytmu

	mean	std
population_size	88.0	10.95
mutation_p	0.22	0.04
crossover_p	0.42	0.31

Wpływ hiperparametru na optymalizację

Ustalone reszta parametrów:

- rozmiar populacji – 20
- prawdopodobieństwo mutacji – 0.3
- llimit pokoleń – 500

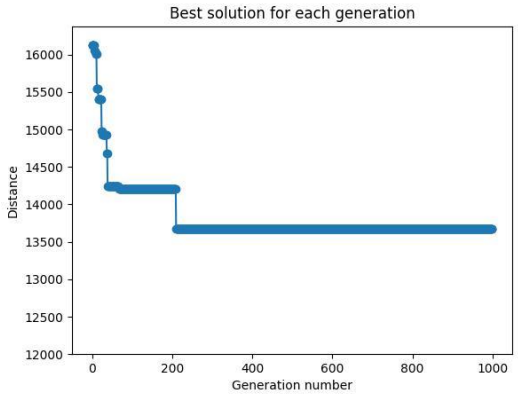
crossover_p	grade_mean	grade_std
0.1	14129.4799	455.58365833229107
0.2	14702.775099999999	426.83567629981786
0.3	14593.1886	678.5689925388244
0.4	14290.226	592.9596767403332
0.5	14397.6816	322.6969365117138
0.6	14714.738500000001	461.13493431249475
0.7	14281.7687	482.3314834482035
0.8	14671.7157	408.5778800506423
0.9	14525.931699999997	439.3157100418424
1.0	14620.151300000001	570.594862825154

Najlepsza wartość wystąpiła dla crossover_prob = 0.1 (14129.4799). Wtedy algorytm głównie polega na mutacji, a krzyżowanie jest znikome.

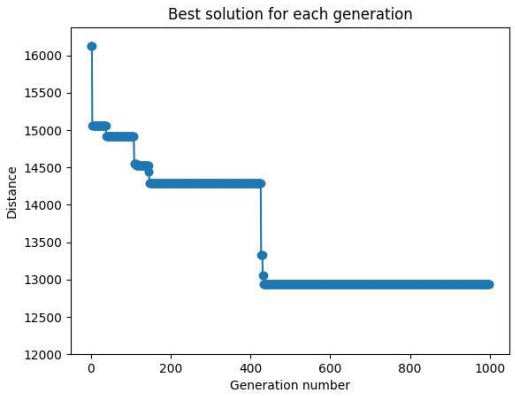
Równie dobre wartości dla prawdopodobieństwa 0.4 i 0.7. Taka wartość pozwala zarówno na eksplorację jak i eksploatację w celu znalezienia rozwiązania.

Jednak najmniejsze odchylenie standardowe odnotowano dla wartości parametru 0.5

crossover_prob = 0.2



crossover_prob = 0.4



Crossover_prob = 0.2

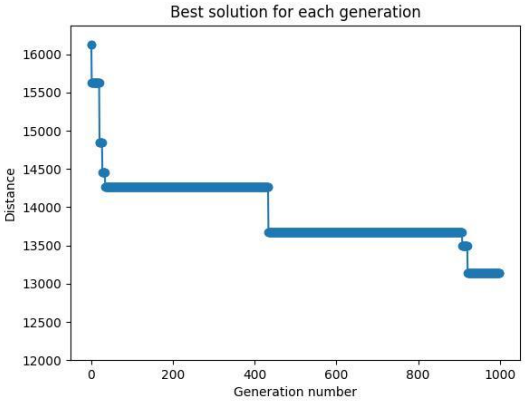
- w pierwszych 100 pokoleniach widać wyraźną poprawę rozwiązania, jednak przy 200 pokoleniu zatrzymuje się już w znalezionym minimum lokalnym
- eksploracja w pierwszych pokoleniach wynika głównie z mutacji
- za małą wartość prawdopodobieństwa krzyżowania, aby przekroczyć znalezione minimum

Crossover_prob = 0.4

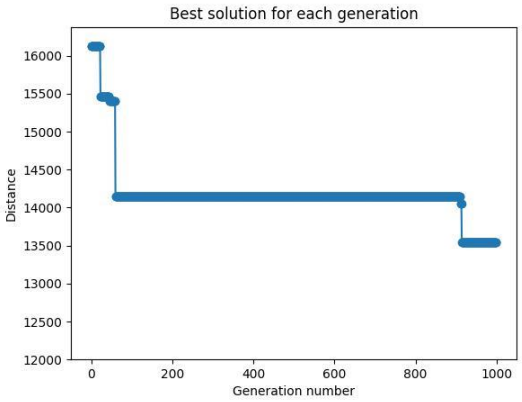
- na początku również widać dużą eksplorację i poprawę rozwiązania

- w pokoleniu ok. 150 algorytm się zatrzymał na minimum lokalnym, ale wydostaje się z niego po 400 pokoleniach i znajduje dużo lepsze rozwiązanie
- większa wartość parametru pozwala na większą eksploatację dobrych rozwiązań

crossover_prob = 0.8



crossover_prob = 1.0



Crossover_prob = 0.8

- na początku widać dużą eksplorację i poprawę rozwiązania
- krzyżowanie umożliwiło większą eksploatację, przez co algorytm chwilę zatrzymuję się na znalezionym minimum
- niewielkie poprawy rozwiązania występują w kolejnych pokoleniachs

Crossover_prob = 1.0

- bardzo mała eksploracja
- trudność w wydostaniu się z znalezionego minimum lokalnego, nadmierna eksploatacja

Najlepsze znalezione rozwiązanie

grade (distance)	solution
11621.506999999998	'Siedlce', 'Gdańsk', 'Olsztyn', 'Ciechanów', 'Częstochowa', 'Wrocław', 'Wałbrzych', 'Sieradz', 'Łódź', 'Jelenia Góra', 'Kalisz', 'Zielona Góra', 'Ostrołęka', 'Słupsk', 'Gorzów Wielkopolski', 'Poznań', 'Konin', 'Bydgoszcz', 'Włocławek', 'Toruń', 'Kielce', 'Leszno', 'Piła', 'Koszalin', 'Szczecin', 'Elbląg', 'Białystok', 'Legnica', 'Piotrków Trybunalski', 'Skierniewice', 'Tarnów', 'Krosno', 'Kraków', 'Katowice', 'Chełm', 'Przemyśl', 'Rzeszów', 'Nowy Sącz', 'Cieszyn', 'Bielsko - Biała', 'Tarnobrzeg', 'Zakopane', 'Opole', 'Płock', 'Radom', 'Łomża', 'Suwałki', 'Zamość', 'Lublin', 'Biała Podlaska', 'Warszawa'

Wnioski końcowe

- Najlepsze rezultaty uzyskano przy prawdopodobieństwach krzyżowania w zakresie 0.1 - 0.4. Niższe wartości sprzyjają eksploracji (dominacja mutacji), co pozwala uniknąć lokalnych minimów, podczas gdy umiarkowane wartości (np. 0.4) oferują dobry balans między eksploracją i eksploatacją.
- Wyniki są najbardziej stabilne dla prawdopodobieństwa krzyżowania równego 0.5
- Nadmierne krzyżowanie (np. 0.8–1.0) prowadziło do wpadania w lokalne minima