Задача за Търговския Пътник

Решение с Генетичен Алгоритъм

Николай Пашов

17 април 2022 г.

Съдържание

1 Описание на решението

Връх в графа се кодира с ASCII символи - букви от латинската азбука. Път в графа се изразява като символен низ. Пример: *'ABCDA'*. Генетичният алгоритъм е имплементиран на езика Python. Входни данни за програмата са:

- Матрица на съседство. Различните върхове от пътя са кодирани с ASCII символи.
- Брой поколения.
- Брой агенти.
- Функция за размножаване на два индивида.
- Функция за селекция.
- Функция за мутация.
- Начален/краен връх.

Алгоритъмът е имплементиран по такъв начин, че позволява лесно променяне на функциите за размножаване, селекция и мутация. Предаденият код идва с примерни имплементирани такива.

1.1 Фитнес функция

Имплементира фитнес функция като отрицателната сума на пътя, зададен от решението на агента. Изчислява се, използвайки матрицата на съседство. Сумата е отрицателна, за да важи правилото, че по-голяма фитнес функция представлява по-добро решение.

1.2 Функция за размножаване

Имплементира One-point crossover на две решения. В случаите, в които наследникът има дублиращи се букви, дублираните букви се заместват със случайни неизползвани. Пример:

$$crossover('ABCDA', 'ADCBA', 2) \rightarrow 'ABCBA' \rightarrow 'ABCDA'$$

.

1.3 Функция за селекция

Имплементира селекция по следния принцип:

1. Сортира агентите в популацията в нарастващ ред по фитнеса на всеки агент.
2. Добавя към селектираните агенти последните 40% агента от сортирания масив.
3. Добавя към селектираните агенти случайно избрани 10% от първите 60% от сортирания масив.

В този си вид функцията за селекция филтрира 50% от агентите. Самите проценти са хиперпараметри, които подлежат на промяна и експериментация.

1.4 Функция за мутация

Имплементира функция за мутация, която разменя местата на две случайно избрани букви от решението на агент. Пример:

$$'ABCDA' \rightarrow 'ADCBA'$$

В настоящата имплементация агент има 5% вероятност да мутира.

1.5 Изпълнение

1. Получаване на входни данни
2. Създаване на популация от агенти със случайни решения. Размерът на популацията е зададен от входните данни.
3. Итериране толкова пъти, колкото е зададено от входните данни
   (а) Селекция
   (б) Размножаване, докато размерът на новата популация стане толкова, колкото е броят агенти от входните данни
   (в) Мутация

(г) Извеждане на стандартния изход данни за поколението: минимална фитнес функция, максимална фитнес функция, средна фитнес функция

4. Извеждане на стандартния вход 10-те най-добри агента от последното поколение и техните фитнес функции.

5. Терминация

1.6 Псевдокод

input ← input_data
population ← generate_population()
iteration ← 0
while it ≠ input.max_iterations **do**
    population ← input_data.selection_func(population)
    population ← input_data.crossover_func(population)
    population ← input_data.mutation_func(population)
end while
    present_results(population)

2 Инструкции за компилиране

Имплементацията е реализирана на Python 3.10. Очаква се да бъде изпълнима и на по-стари версии, заради backward compatibility, но не е тествана и не се гарантира от автора. Изисква се инсталиран Python и python командата в терминала да извиква интерпретатор на Python 3.10. За изпълнение на програмата:

cd <code_directory>
python <filename>

В кода са въведени три различни по големина матрици на съседство. За да се пусне експеримент с някоя от тях, трябва променливата adj_table на ред 209 и 212 в конструктора на клас *Experiment* да се замени с името на съответната матрица на съседство. Може да се направят и нови матрици на съседство под формата на речник от речници. Хиперпараметри като брой поколения, брой агенти, функции за мутация, селекция,

```
adj_table = {
    'A': {'B': 5, 'C': 10, 'D': 3},
    'B': {'A': 5, 'C': 12, 'D': 8},
    'C': {'A': 10, 'B': 12, 'D': 5},
    'D': {'A': 3, 'B': 8, 'C': 5}}
```

Фигура 1: Матрица на съседство за експеримент 1.

размножаване, фитнес функция и др. могат също да се променят през викането на конструктора на клас *Experiment* на редове 204-214.

3 Примерни резултати

3.1 Експеримент 1

Резултати за графът, който е даден в условието на задачата. Матрицата на съседство е изобразена на фиг. 1.

Изход на програмата за 20 агента и 5 поколения:

```
epoch 0
max fitness function: -25
min fitness function: -33
average fitness: -26.5
epoch 1
max fitness function: -25
min fitness function: -33
average fitness: -25.7
epoch 2
max fitness function: -25
min fitness function: -28
average fitness: -25.15
epoch 3
max fitness function: -25
min fitness function: -25
average fitness: -25
```

```
adj_table2 = {
'A': {'B': 5, 'C': 10, 'D': 3, 'E': 2, 'F': 10},
'B': {'A': 5, 'C': 12, 'D': 8, 'E': 5, 'F': 12},
'C': {'A': 10, 'B': 12, 'D': 5, 'E': 10, 'F': 3},
'D': {'A': 3, 'B': 8, 'C': 5, 'E': 4, 'F': 5},
'E': {'A': 2, 'B': 5, 'C': 10, 'D': 4, 'F': 8},
'F': {'A': 10, 'B': 12, 'C': 3, 'D': 5, 'E': 8}}
```

Фигура 2: Матрица на съседство за експеримент 2.

```
epoch 4
max fitness function: -25
min fitness function: -33
average fitness: -25.4
TOP 10 SOLUTIONS AND THEIR PATH LENGTH
[('ABCDA', 25), ('ABCDA', 25), ('ADCBA', 25),
('ADCBA', 25), ('ABCDA', 25), ('ADCBA', 25),
('ABCDA', 25), ('ADCBA', 25), ('ADCBA', 25),
('ADCBA', 25)]
```

3.2 Експеримент 2

Втори експеримент с по-голяма матрица на съседство (фиг. 2) и $6! = 720$ възможни решения.

Експериментът е изпълнен с 200 агента и 50 поколения. Изходът на програмата е твърде голям, за да бъде показан. Фитнес функцията намалява монотонно първите 10 поколения, след което започва да варира между -29 и -31. Финалното решение на алгоритъма е, че най-краткият път е 29.

4 Методи на тестване

Коректността на алгоритъма е тествана по следните начини:

- Тестове на коректността на отделните компоненти - класове, методи и т.н. Това намалява риска от грешки в решението.

- Тестване на предложените от алгоритъма решения на ръка за по-малки и прости задачи (като експеримент 1).
- Следене на промяната на фитнес функцията за различните епохи. Очаква се, че намаляваща средна фитнес функция индицира подобрение на решението