СОФИЙСКИ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ”
Факултет по математика и информатика

Домашна работа №1
Constraint Satisfaction Problem

Изготвила: Весела Илиянова Стоянова
Дисциплина: Системи, основани на знания
III курс, Информационни системи
Факултетен номер: 71949

1. Описание на задачата и описание на използваните методи за нейното разрешаване

Представено е решение на задачата за удовлетворяване на ограниченията (Constraint Satisfaction Problem, CSP) за натоварване на кораби. Целта е да се натоварят множество товари по начин, който максимизира сумарната им стойност без да нарушава капацитета на кораба и да позволи да се носят определен брой товари с минимална сумарна тежест. Решаването на задачата се свежда до използването на генетичен алгоритъм – вариант на стохастично търсене в лъч, при което новите състояния се генерират чрез комбиниране на двойки родителски състояния вместо чрез модифициране на текущото състояние. В решението на задачата са използвани основните принципи на генетичните алгоритми:

- Състоянията се представят като низове от нули и единици;
- Имаме оценяваща функция (fitness function), която оценява близостта до целта на съответното състояние;
- Алгоритъмът започва с популация от k случайно генерирани състояния;
- Имплементирани и използвани са принципите за получаване на състоянията от следващите поколения: селекция, кръстосване и мутация.

2. Описание на реализацията с псевдокод

В програмата са реализирани следните класове:

- **Class Cargo**, който представлява самия товар. Има член-данни value (стойност) и weight (тегло). Методите getFirstCargos() и getSecondCargos() се използват за четене от файл, в който се съдържат съответните данни.
- **Class Data** и **Class Datum** се използват във връзка с четенето от файл.
- **Class Genome** се съдържат данни за различните индивиди. Има член-данна ArrayList<Integer> places, в който се съдържа информация за това на кои места имаме 1 и на кои 0.
- **Class GenomeTest** съдържа методи, в които се тестват всички методи от class Genome.
- **Class Population** съдържа методи за генериране на нова популация.

- **Class PopulationTest** съдържа методи, в които се тестват всички методи от class Population.

Основните принципи и функции, използвани в програмата са:

- **Генетичен алгоритъм**

Function Genetic-Algorithm(population, fitness-function)

returns an individual

repeat

initialize new-population **with** null

for I = 1 to size(population) **do**

y = random-select(population, fitness-function)

x = random-select(population, fitness-function)

child = cross-over(x, y)

mutate(child) with a small random probability

add child to new-population

population = new-population

until some individual is fit enough or enough time has elapsed

return the best individual in population

- **Fitness function** – оценява индивида в неговото декодирано състояние, а не когато е битов низ. Има по-големи стойности за по-добрите състояния.
- **Кръстосване** – избират се двойка родителски състояния и се определя точката на кръстосването им. В програмата е реализирано кръстосване в две точки, тоест са избрани две точки на кръстосване. Полученият низ - резултатът от началото до първата точка на кръстосване е копие на съответната част от първия родител, частта от първата точка на кръстосване до втората точка на кръстосване е копие на съответната част на втория родител и частта от втората точка на кръстосване до края е копие на оставащата след втората точка на кръстосване част от първия родител.
- **Мутация** – извършване на случайни промени в случайно избрана малка част от новата популация с цел да се осигури възможност за достигане на

всяка точка от пространството на състоянията и да се избегне опасността от попадане в локален екстремум.

```
Function Mutation
FOR I = 1 to N
        IF rand <= p(i)
        j = collisionpoint (1, dim)
        mutrange = [upper _Bound(j) – low_Bound(j)] * p(i)
        XMij = Xij + randn * mutrange * (1 – t / T max)
        XNij = Xij  - randn * mutrange * (1 – t/ T max)
        ENDIF
        IF f(XMij) < f(XNij)
                Xij = XMij
        ELSE Xij = XNij
        ENDIF
ENDFOR
```

- **Селекция** – в генерирането на състоянията от следващото поколение участват някои от най-добрите представители на текущото поколение (съгласно оценяващата функция), избрани на случаен принцип.

Програмна реализация на програмата:

При стартиране на програмата се изисква от потребителя да въведе броя на индивидите и броя на епохите. След това се изисква да въведе име на файла, от който да бъдат прочетени данните. Към проекта има 2 файла с данни – cargos.json и moreCargos.json. След като потребителят въведе необходимите входни данни, то се генерират нови популации, обхождат се епохите и се намира най-доброто решение. На конзолата се принтират популациите, най-доброто решение, стойността на оценяващата функция.

3. Инструкции за компилиране на програмата

- Тъй като в програмата четенето на файлове е реализирано чрез JSON, то е необходимо да имате инсталиран JSON reader.
- Ако нямате, то следвайте инструкциите:

File -> Project Structure -> Libraries -> + -> From Maven -> *Type com.google.code.gson:gson:2.8.5* -> Click OK -> Apply -> OK.

- След това при стартиране на програмата ще трябва да въведете 2 числа (int) и един низ (String).
- Накрая на конзолата ще видите резултата.

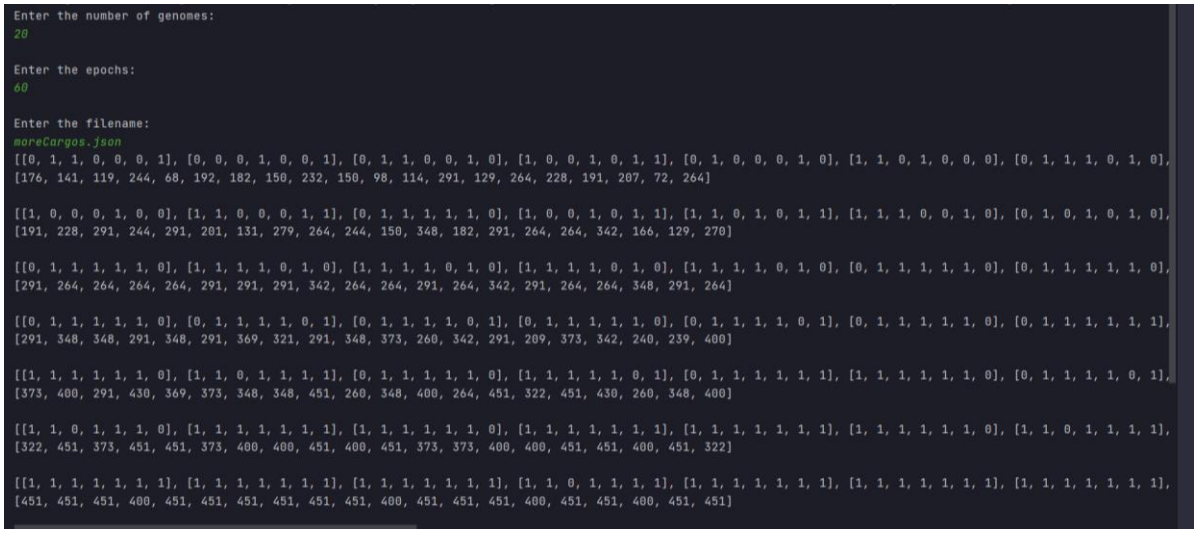
4. Примерни резултати

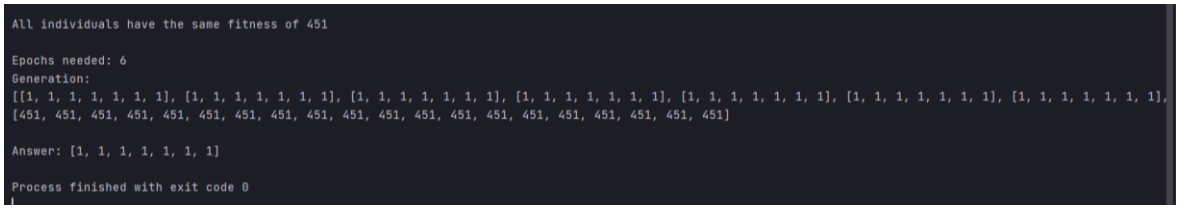
- **Пример 1**

```
Enter the number of genomes:
4

Enter the epochs:
20

Enter the filename:
cargos.json
[[0, 0, 0, 0, 0, 1, 0], [0, 1, 0, 0, 0, 1, 0], [0, 1, 1, 0, 1, 1, 0], [1, 1, 1, 1, 1, 1, 1]]
[21, 68, 228, 451]

[[0, 1, 1, 0, 1, 1, 1], [1, 1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1, 0], [0, 1, 1, 0, 1, 1, 1]]
[306, 373, 373, 306]

All individuals have the same fitness of 373

Epochs needed: 1
Generation:
[[1, 1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1, 0]]
[373, 373, 373, 373]

Answer: [1, 1, 1, 1, 1, 1, 0]

Process finished with exit code 0
```

За файла cargos.json, за 4 индивида и 20 епохи получаваме, че ни е необходима 1 епоха за достигане на резултата.

- **Пример 2**





За файла moreCargos.json за 20 индивида за 60 епохи получаваме, че ще са ни необходими 6 епохи за достигане на резултата.