Боян Дончев
Фн: 72029
Системи основани на знания

Домашна работа №1

Задача за осемте царици

В задачата се иска да се напише програма, която решава задачата за осемте царици с помощта на генетичен алгоритъм. Задачата е да се намери такова разпределение на 8 царици върху стандартна шахматна дъска, че нито една двойка царици да не се застрашават.

За решение на проблема ще използваме генетичен алгоритъм, който използва поколения от състояния, математическа функция която изчислява колко близо до крайната цел е дадено състояние, метод за кръстосване на състояния за съставяне на ново поколение и метод за мутиране на състояния.

В рамките на нашата програма ще използваме следните термини:

Състояние ще наричаме масив с осем целочислени елемента, вариращи от 1 до 8, описващи на кой ред от съответната колона на шахматната дъска има царица. Тъй като фактът че всяка царица трябва да е в различна колона от дъската, за да е изпълнено условието на задачата, е тривиален, ще го приемем за даденост, което ще спести памет и време за изпълнение на програмата.

Поколение ще наричаме списък от състояние, с краен брой елементи.

Фитнес функция ще наричаме математическата функция, пресмятаща колко е вероятно дадено състояние да участва в съставянето на следващото поколение.

Мутация ще наричаме малка промяна по дадено състояние.

Родител ще наричаме всяко състояние което ще участва в съставянето на следващото поколение.

Наследник ще наричаме всяко състояние от следващото поколение.

Решение ще наричаме такова състояние, което отговаря на изискванията на заданието.

Ще използваме програмния език JavaScript за реализация на решението. Единствената външна библиотека която ще се наложи да използваме, ще ни помогне да пишем файлове, в които ще записваме целия процес на генетичния алгоритъм за намиране на решение.

```
1 const fs = require('fs');
```

В началото на кода ще дефинираме няколко константи, с които можем да променяме някои от характеристиките на алгоритъма, според наши предпочитания.

```
const POPULATION_SIZE = 10;
const NUM_OF_QUEENS = 8;
const MUTATION_RATE = 0.05;
const PARENTS_RATIO = 0.5;
const MAX_FITNESS =
  NUM_OF_QUEENS * (NUM_OF_QUEENS-1) / 2;
```

Стойностите които ще използваме за демонстрация са:

Размер на поколението = 10

Брой царици = 8 (има такава променлива, защото алгоритъмът е приложим и за различна от стандартната шахматна дъска)

Коефициент на мутация = 0,05 (тази стойност отговаря за това какъв е шансът всяка една от целочислените стойности в масива на дадено състояние да се промени с друга случайна)

Съотношение на родители = 0,5 (тази стойност отговаря за това каква част от поколението ще бъдат родители на следващото поколение)

Ще имаме една глобална променлива, която ще съдържа настоящето поколение.

```
var population = [];
```

Програмата ще започне със съставяне на началното поколение, в което има състояния със случайни стойности. За всяко от състоянията се пресмята фитнеса му. След като всички състояния са генерирани, се сортират по стойността на фитнеса им.

```
function initialPopulation() {
  for (let p = 0; p < POPULATION_SIZE; p++) {
    var state = { seq: [] };

    for (let i = 0; i < NUM_OF_QUEENS; i++) {
      var num = Math.floor(
        Math.random()*NUM_OF_QUEENS
      ) + 1;
      state.seq.push(num);
    }

    state.fitness = fitness(state.seq);
    population.push(state);
  }
  population.sort(
    (a, b) => (a.fitness < b.fitness ? 1 : -1)
  );
}
```

Фитнес функцията пресмята за всяко състояние колко са двойките царици които не се застрашават взаимно. Колкото по-голяма е стойността пресметната от фитнес функцията, толкова по близко до правилното решение е това състояние.

```
function fitness(seq) {
  var score = 0;

  for (let row = 0; row < NUM_OF_QUEENS; row++) {
      col = seq[row];

      for (let other_row = 0;
        other_row < NUM_OF_QUEENS;
        other_row++) {
          if (other_row == row)
            continue;
          if (seq[other_row] == col)
            continue;
          if (other_row + seq[other_row] == row + col)
            continue;
          if (other_row - seq[other_row] == row - col)
            continue;
          score++;
      }
  }

  return score/2;
}
```

Докато няма състояние в поколението, което е достигнало максималната фитнес стойност, се създава следващо поколение. За съставяне на следващо поколение се избират родители от сегашното, след това се съставят двойки от родители, които се кръстосват и създават наследници, върху които се прилага мутираща функция. Накрая се избират най-добрите наследници, които формират следващото поколение.

```
function nextGeneration() {
  const parents = getParents();
  const couples = getCouples(parents);

  const nextGen = []

  couples.forEach(couple => {
    var offsprings = crossover(couple);
    offsprings.forEach(offspring => {
      nextGen.push(offspring);
    })
  });

  nextGen.sort(
    (a, b) => (a.fitness < b.fitness ? 1 : -1)
  );
  population = nextGen.slice(0, POPULATION_SIZE);
}
```

Родителите се избират като, от поколението се взимат най-добрите състояния спрямо фитнес функциите им, като броят им се определя от константата за **Съотношение на родители**

```
function getParents() {
  const parents = population.slice(
    0,
    POPULATION_SIZE*PARENTS_RATIO
  );

  return parents;
}
```

За двойки родители се взимат всички комбинации от двама родити

```
function getCouples(parents) {
  var couples = []
  parents.map(
      (v, i) => parents.slice(i+1).map( w => {
        let curr = [];
        curr.push(v);
        curr.push(w);
        couples.push(curr);
      } )
  );

  return couples;
}
```

Кръстосването на две състояния става като се избере случайна точка на кръстосване, в която се преплитат двата масива и получаваме нови два наследника, на които се прилага мутиращата и фитнес функциите.

```
function crossover(couple) {
  var cross_point = Math.floor(
    Math.random()*NUM_OF_QUEENS
  );
  offsprings = [];

  offsprings.push(
    couple[0].seq.slice(0,cross_point).concat(
      couple[1].seq.slice(cross_point)
    )
  );
  offsprings.push(
    couple[1].seq.slice(0,cross_point).concat(
      couple[0].seq.slice(cross_point)
    )
  );

  offsprings = offsprings.map(
    offspr => {
      let seq = mutate(offspr);
      return {
        seq,
        fitness: fitness(seq)
      }
    }
  );

  return offsprings;
}
```

```
function mutate(seq) {
  for (let row = 0; row < seq.length; row++) {
      if (Math.random() < MUTATION_RATE) {
        seq[row] = Math.floor(
          Math.random()*NUM_OF_QUEENS
        ) + 1;
      }
  }
  return seq;
}
```

По време на изпълнение на цялата програма се записва всяка стъпка в изходен файл, като на последния ред седи намереното решение.

```
function findSolution() {
  let writeStream = fs.createWriteStream(
    `solutions/EightQueensSolution_${Date.now()}.txt`
  );

  initialPopulation()

  let counter = 0;
  while (population[0].fitness < MAX_FITNESS) {
    writeStream.write(
      `>> generation: ${counter} - fitness: ${population
[0].fitness} - current state:\n`
    );
    writeStream.write(population[0].seq + '\n\n');

    nextGeneration();
    counter++;
  }

  writeStream.write(`\n\nSOLUTION:\n`);
  writeStream.write(population[0].seq + '\n');
  writeStream.end();
}
```

Източници:

- Презентации от кура 'Системи основани на знания' за специалност Информационни Системи 3 курс, зимен семестър
- https://medium.com/nerd-for-tech/genetic-algorithm-8-queens-problem-b01730e673fd