Universidade de Pernambuco

Escola Politécnica da Universidade de Pernambuco

Curso de Engenharia da Computação

Problema TSP utilizando algoritmo genético

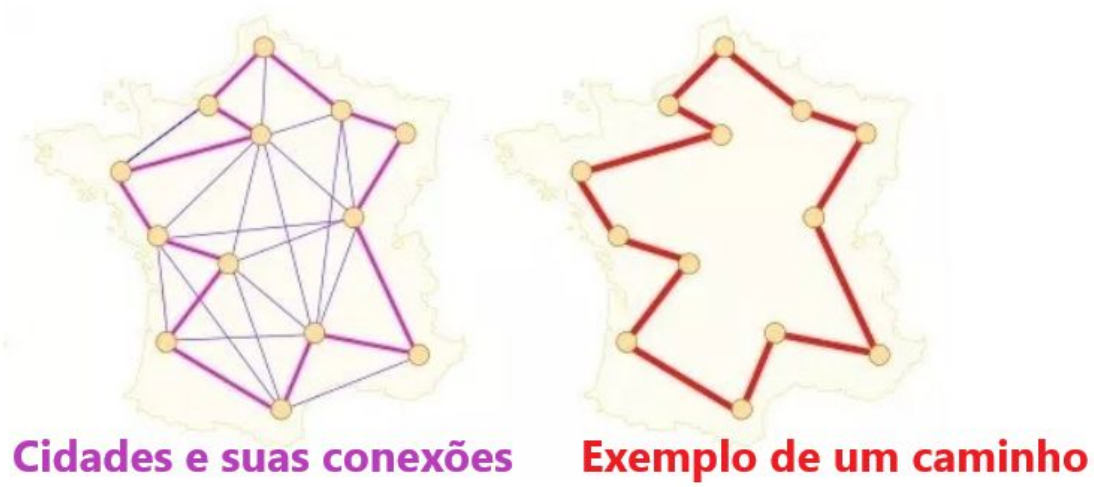
Thiago Lourenço C. Bezerra

Recife
2019

Sumário

1. Caracterização do Problema

O problema do Caixeiro Viajante consiste em encontrar a menor rota que passa por N cidades conectadas entre elas. Nessa implementação estamos utilizando o problema berlin52 da biblioteca TSPLIB para definir os valores das cidades. Cada cidade deve ser visitada apenas uma vez e no fim o caixeiro deve está na cidade de origem.



2. Implementação

O projeto foi desenvolvido na linguagem Python utilizando a técnica de algoritmo genético.

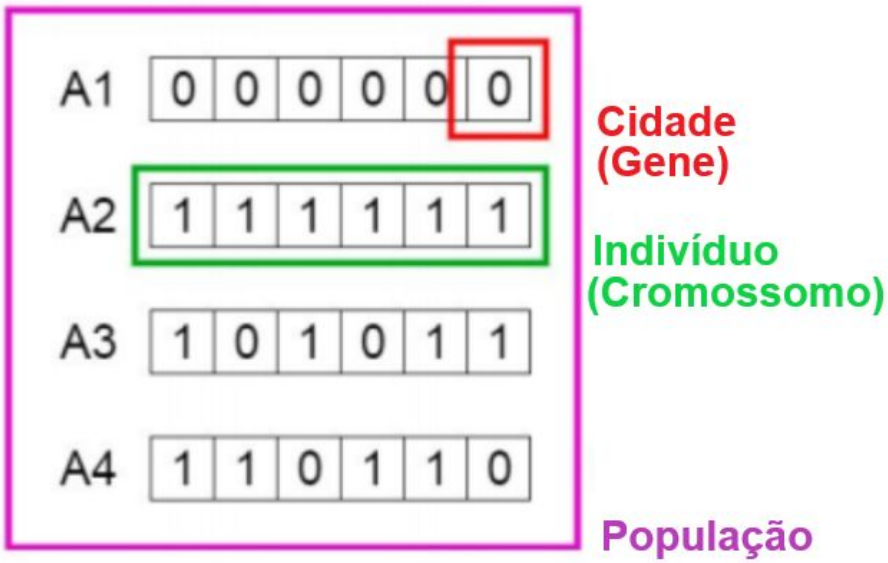
A implementação inicia com um conjunto de indivíduos que contem em seus genes as cidades que devem ser visitadas. Esse conjunto é a população inicial e o objetivo é melhorar a população em cada iteração a fim de que se chegue a um melhor valor fitness, nesse caso, a menor distância percorrida. Esses indivíduos ficam armazenados em uma lista do heap, que guarda os elementos com um ordem de prioridade, além disso, o tamanho da população é de 50 indivíduos e 20 interações. Cada um dos indivíduos possui um caminho e um valor fitness. O caminho representa as cidades que ele percorreu e o valor fitness é o inverso da distância entre as cidades, ou seja:

$$\sqrt{(\textbf{coordenada Xa} - \textbf{coordenada Xb})^2 + (\textbf{coordenada Ya} - \textbf{coordenada Yb})^2}$$

As cidades são representadas por vértices, no qual cada um deles possui um número, a coordenada X e a coordenada Y, que são as características utilizadas para as cidade.

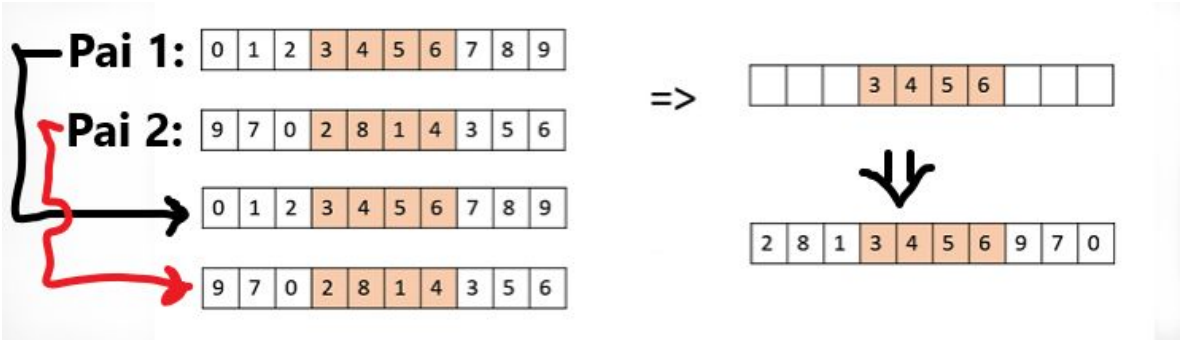
Foi implementado as etapas de: geração, o crossover, a mutação e seleção da população. No crossover foi utilizado o método OX1, na mutação o swap mutation e na seleção o método do torneio.

O objetivo é que cada indivíduo percorra um caminho diferente (soluções) que será melhorado quando forem surgindo novas gerações.

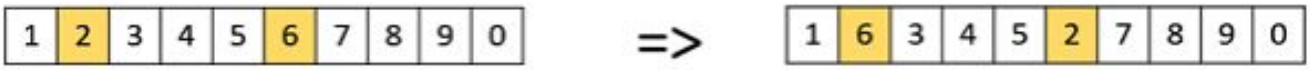


3. Métodos utilizados

3.1. Crossover OX1: Cria-se dois pontos de cruzamento aleatórios no pai essa parte do vetor (parte do gene) é colocada diretamente no filho. Depois copie os números não utilizados do primeiro pai que estão no segundo pai para o filho, agrupando a lista. Vale ressaltar novamente que o tamanho da população que foi usado é de 1000 indivíduos.



3.2. Mutação Swap: Seleciona-se duas posições no cromossomo aleatoriamente e trocamos os valores de posição.



3.3. Seleção por Torneio: selecionamos dois indivíduos aleatórios da população. A escolha desse método foi pela facilidade de implementação e por ser um dos métodos mais populares e utilizados.

4. Código

O código segue a ordem de um algoritmo genético, ou seja, primeiro geramos a população, depois cruzamos os indivíduos para gerar novos, passamos pela fase da mutação e depois a seleção dos melhores, nesse caso. O Algoritmo implementado passa por essas etapas até chegar ao resultado final que queremos.

(1) initialise population;
(2) evaluate population;
(3) **while** (*!stopCondition*) **do**
(4) select the best-fit individuals for reproduction;
(5) breed new individuals through crossover and mutation operations;
(6) evaluate the individual fitness of new individuals;
(7) replace least-fit population with new individuals;

```
251     #Inicialiso a população
252     populacao = geraPopulacao()
253
254     while(melhorSolucao):
255
256         #Avalio os indivíduos e faço a seleção.
257         individuoUm = selecaoIndividuo(populacao)
258         individuoDois = selecaoIndividuo(populacao)
259
260         #Cruzamentos entre os individuos escolhidos. E no mesmo método faço a mutação do filho gerado
261         #No método crossover eu gero um lista de novos individuos.
262         novosIndividuosGerados = crossoverIndividuo(individuoUm, individuoDois)
263
264         #Faço a avaliação do novo indivíduo e se valer a pena adiciono ele na população.
265         populacao = atualizaPopulacao(populacao, novosIndividuosGerados)
266
```

O código comentado é mostrado em anexo. Ele é explicado nas suas próprias linhas.

5. Resultados

Nos testes iniciais o resultado estava alto, por volta de 23.000. Após realizar ajustes na seleção e no crossover conseguimos abaixar o valor fitness para 12877. O melhor indivíduo foi: [33, 51, 11, 13, 27, 4, 15, 5, 48, 44, 34, 25, 6, 46, 16, 20, 50, 28, 12, 41, 8, 9, 10, 43, 40, 37, 24, 38, 39, 45, 19, 35, 36, 49, 32, 17, 7, 2, 42, 21, 31, 18, 22, 1, 23, 30, 29, 47, 14, 52, 26, 3] e o seu valor fitness está mostrado abaixo.

```
Melhor indivíduo:(Menor distância) 12877.9177
```

6. Conclusão

Concluímos que o que praticamos neste projeto é de suma importância, pois vimos um pouco sobre as características e o comportamento do algoritmo genético num problema real, além de observarmos toda a teoria estudada em sala em prática. Além disso, tivemos oportunidade de testar e aprender várias técnicas além das que foram mostradas em sala de aula.

Os resultados obtidos no projeto foram satisfatório, porque os objetivos do projeto como encontrar o menor caminho, gerar, cruzar, mutar e selecionar indivíduos utilizando técnicas para resolver o TSP e entender como o algoritmo genético foram cumpridos.

Claro que o que aprendemos é o básico, percebemos que as possibilidades de utilização desse algoritmo é enormes, dependendo das suas limitações e do conhecimento de quem o utiliza.

7. Referências bibliográficas

[1] Slide de Algoritmo genético mostrado em sala. **https://classroom.google.com/u/1/c/Mzc3MDgwMzEwNTFa/a/Mzc3MjAxNjI1OTla/details.** Acessado em 14/09/2019

[2]Algoritmo Genético. **http://conteudo.icmc.usp.br/pessoas/andre/research/genetic/#links.** Acessado em 14/09/2019

[3] Slideplayer - Algoritmo genético. https://slideplayer.com.br/slide/359174. Acessado em 15/09/2019

[4] Heap queue algorithm. https://docs.python.org/3/library/heapq.html. Acessado em 16/09/2019.

[5] Tutorial Point. https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm. Acessado em 17/09/2019