Aplicando algoritmos de busca local ao Problema do Caixeiro Viajante

Lucas S. Beneti

[1] Departamento Acadêmico de Informática
Universidade Tecnológica Federal do Paraná – Curitiba, PR – Brazil

lucasbeneti@alunos.utfpr.edu.br

Resumo. *O presente trabalho tem como finalidade comparar dois algoritmos famosos para solução visando uma solução ótima de um dos problemas mais conhecidos da computação, o problema do Caixeiro Viajante. Os dois algoritmos utilizados foram o Busca Local em Feixe (Local Beam Search) e o Algoritmo Genético (Genetic Algorithm). Foi elencado também os motivos de se utilizar desse tipo de algoritmo para buscar uma solução para esse tipo de problema e qual seria o mais indicado para o presente objeto de estudo.*

1. Introdução

O problema do Caixeiro Viajante (PCV) é considerado um problema NP-difícil de otimização, onde para sua solução, podem ser utilizadas diversas estratégias e algoritmos, como os do tipo Busca Cega. Os mesmos operaram gerando caminhos diferentes até que se chegue ao valor objetivo, entretanto um resultado poderia ser alcançado após o teste de todos os caminhos possíveis, o que, dependendo das suas restrições computacionais ou de tempo, provavelmente não seria viável. Por isso para o PCV é mais indicado que se utilize de algoritmos de Busca Informada, onde a cada iteração, o melhor resultado é averiguado e se decide se a busca continua ou não caso tenha encontrado um máximo local. Como é o caso dos dois algoritmos levados em conta neste trabalho, a Busca Local em Feixe e o Algoritmo Genético. Ao final do trabalho é possível ver um comparativo entre as duas soluções.

2. O Problema

O PVC é um problema de otimização classificado como NP-hard, que consiste em um uma representação de pontos que em um mapa por exemplo cujo objetivo é que se passe por todos os pontos desse gráfico de modo que nunca se pode passar pelo mesmo ponto mais de uma vez e deve ser o caminho mais otimizado a partir de um ponto inicial.

Para problemas de otimização, é esperado pelos algoritmos algo como uma função que mostre o quão próxima do objetivo a solução está, esse tipo de função é chamado de funco fitness, e no caso do PCV é o que determina se uma viagem entre dois pontos é a melhor opção no momento, é seu custo de viagem, que é o mesmo em ambos os sentidos.

Especificamente para o PVC em questão, foi fornecida uma matriz de distâncias, que correlaciona cada ponto do grafo com cada outro ponto, ou seja, se trata de uma matriz simétrica de ordem 15, com 225 caminhos relacionáveis, o que pode ser vista à seguir na Tabela 01:

Tabela 1. Matriz de distâncias

0	29	82	46	68	52	72	42	51	55	29	74	23	72	46
29	0	55	46	42	43	43	23	23	31	41	51	11	52	21
82	55	0	68	46	55	23	43	41	29	79	21	64	31	51
46	46	68	0	82	15	72	31	62	42	21	51	51	43	64
68	42	46	82	0	74	23	52	21	46	82	58	46	65	23
52	43	55	15	74	0	61	23	55	31	33	37	51	29	59
72	43	23	72	23	61	0	42	23	31	77	37	51	46	33
42	23	43	31	52	23	42	0	33	15	37	33	33	31	37
51	23	41	62	21	55	23	33	0	29	62	46	29	51	11
55	31	29	42	46	31	31	15	29	0	51	21	41	23	37
29	41	79	21	82	33	77	37	62	51	0	65	42	59	61
74	51	21	51	58	37	37	33	46	21	65	0	61	11	55
23	11	64	51	46	51	51	33	29	41	42	61	0	62	23
72	52	31	43	65	29	46	31	51	23	59	11	62	0	59
46	21	51	64	23	59	33	37	11	37	61	55	23	59	0

3. Implementação e uso dos algoritmos

O programa inteiro desenvolvido para a comparação dos algoritmos de Busca Local em Feixe (BLF) e Algoritmo Genético (AG) foi feito em Python 3.7 e com o auxílio de um pacote se intitulado MLRose, justamente utilizado para resolver rapidamente problemas de otimização como o PCV.

Para rodar o programa, é necessário que o usuário forneça um valor para a constante k do BLF, onde k é a largura do feixe da busca, o que basicamente é o número de caminhos que manterá em memória para posteriormente, ao final da execução, escolher o melhor caminho dentre os k-caminhos monitorados.

Já o para o AG, o usuário precisa fornecer alguns outros valores como a probabilidade de mutação, tentativas máximas e um estado randômico, para que o algoritmo possa rodar.

O código pode ser encontrado e baixado por completo através do repositório que criei para o experimento: https://github.com/LucasBeneti/sistemas-inteligentes

4. Resultados e Discussões

Ao final dos testes, sendo o mesmo problema aplicado para ambos os algoritmos em questão, foi constatado que o algoritmo de Busca Local em Feixe obteve um desempenho muito mais satisfatório que o Algoritmo Genético, provavelmente pela sua simplicidade e a própria dimensão do problema. Provavelmente o AG teria se saído muito melhor em um problema mais complexo em termos de tamanho.

Ao rodar o programa e fornecer as devidas entradas requeridas pelo menos, ao seu término, é mostrada em tela, do terminal no caso, os resultados, do BLF com seu ponto inicial de busca, caminho tomado e resultado em termos do custo total para realizar esse caminho e em seguida é possível ver o resultado final do AG, com seu melhor estado atingido, após o número máximo de tentativas fornecidas pelo usuário. Ao fim do

experimento, é possível também ver o tempo de execução de cada algoritmo basicamente. Nota-se que o tempo necessário para o AG rodar, é muito superior ao do BLF.

Um exemplo dos resultados podem ser vistos à seguir:

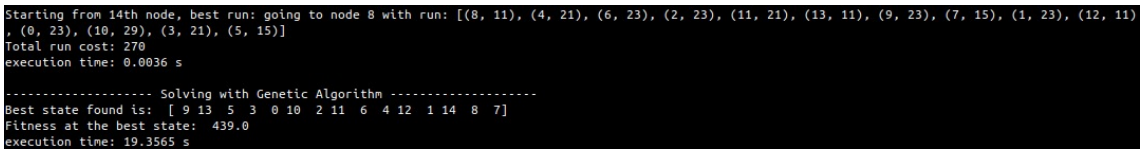

Figura 1. Resultado final do teste entre BLF e AG, respectivamente

5. Conclusão

Tendo os resultados em mãos, pode-se dizer que mesmo para algoritmos que poderiam se sair muito bem em outros problemas, o algoritmo genético deixou muito a desejar no presente experimento. Provavelmente, como comunicado anteriormente, é mais provável que esse resultado deplorável do algoritmo genético se dá pela simplicidade do problema, provavelmente ele deve se sair muito melhor em problemas onde a variância é mais apreciada, o que não é o foco no problema do caixeiro viajante.

Já para a BLF, em comparação ao AG, se saiu muito melhor na busca por uma solução ótima do PCV, provavelmente pelo fato de ser uma averiguação em paralelo dos k melhores caminhos.

A noção que o algoritmo genético se sairia melhor dependendo da complexidade do problema se dá pelo fato de que o próprio AG é considerado uma melhoria do Busca Local em Feixe Estocástico, que seria o BLF com a adição de probabilidade na sua implementação, onde, dependendo do resultado obtido, ocorre resets na posição inicial até que a mais otimizada solução tenha sido encontrada.

O que conlui-se, é que algoritmos de Busca Local se saem muito melhor em problemas que vizam principalmente obter uma solução otimizada, não necessariamente a perfeita, ainda mais visto que a solução perfeita, provavelmente seria subjetiva a determinadas restrições e pontos de vista.

Referências

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition.