Introdução

Como sabemos, o problema do caixeiro viajante é um problema de alta complexidade - se tratando de sua performance, aumentando exponencialmente o seu tempo com cada cidade adicionada. Por esta razão, não conhecemos ainda e talvez nunca conheceremos algoritmo que resolva esse problema com um custo de tempo linear.

É por esse motivo que estudamos algoritmos probabilísticos que reduzem o tempo de execução, porém não são 100% assertivos, no caso, nesse trabalho, falaremos do algoritmo Simulated Annealing (SA), que é um algoritmo que usa uma logica de temperatura e resfriamento para a verificação de caminhos, e assim, tentando chegar em um ponto ótimo. O problema, como falado anteriormente é que o algoritmo não é totalmente confiável, então em uma aplicação real, que não tenhamos a resposta pronta para saber se chegamos realmente em um ponto ótimo, o único jeito para descobrir a distância do seu resultado para o ponto ótimo, seria um algoritmo que realmente teste os caminhos possíveis, não fugindo de algoritmos exponenciais, então, o objetivo do SA é achar o melhor ponto possível, em um algoritmo mais eficiente, mesmo não sendo determinístico.

Desenvolvimento

Para o desenvolvimento do algoritmo SA eu basicamente segui os passos descritos nos slides do professor, que consiste em gerar uma solução aleatória inicial, e com essa solução inicial que no caso é um vetor na ordem com que a ida nas cidades será, você irá perturbar esse vetor, assim, achando resultados melhores, que sempre substituirão esse nosso vetor, quanto resultados piores, que irá depender da temperatura na interação atual do programa, se vai ou não fazer essa troca.

Sendo um algoritmo relativamente simples e rápido de ser implementado, porém que demanda diversos testes e experimentos, principalmente a respeito de uma temperatura inicial e final ideal, pois, se a temperatura estiver muito alta, a aleatoriedade ao longo da execução será maior, tendo menos tempo de otimização do caminho, chegando em um resultado não ótimo, e se configurarmos uma temperatura muito baixa, a aleatoriedade vai ser pequena, fazendo com que o resultado final não saia de uma área que tenha um ótimo local longe do ótimo global, chegando em um resultado também não ótimo.

Temos 10 gráficos disponibilizados pelo professor com comportamentos diferentes, para fazermos os testes de resfriamento, testei com o gráfico, 0, 2 e 8, nos testes realizados, o resfriamento que nos deu melhor caminho e melhor média, foi o gráfico 8, os demais gráficos testados, nos deram um resultado não tão ruim “sempre acima de 600”, mas a sua média de custos foi sempre muito acima do gráfico 8, maiores detalhes sobre os resultados, serão discutidos na parte de análise.

Experimentos

Nos meus experimentos, eu usei 3 funções de resfriamento, como já foi falado acima. A melhor temperatura inicial que achei foi 50, temperatura final deixei em 1, se eu deixasse uma temperatura menor que essa na inicial, a aleatoriedade do algoritmo ficava quase nula, mesma coisa com alguma alteração da temperatura final.

Gráficos com a equação 8, com 200 mil interações e 50 cidades.

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

Para as 100 cidades, tive que aumentar a temperatura para 12 mil, e a temperatura final para 24 e aumentei para 500 mil instancias

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

Se formos comparar com o primeiro gráfico, teríamos o seguinte resultado

Gráfico

Descrição gerada automaticamente