Trabalho de Implementação 01

Matéria: Heurísticas e Metaheurísticas **Professor**: Thiago Ferreira de Noronha

Aluno: Guilherme Teres Nunes **Matrícula**: 2016077187

Problema do Caixeiro Viajante

Esse trabalho consiste em apresentar uma heurística construtiva para resolver o problema clássico do Caixeiro Viajante, ou **Traveling Salesman Problem (TSP)**. Esse problema consiste em: dado uma lista de cidades “E” e suas coordenadas (ou distâncias entre elas), calcular qual a menor rota que o caixeiro viajante pode percorrer, começando de uma cidade X e terminando no mesmo local, passando apenas uma vez em cada cidade, para visitar **todas** as cidades da lista “E”.

Esse é um problema NP-Difícil extremamente conhecido e popular no meio da computação, exatamente por sua aplicabilidade prática e pela vasta opção de heurísticas para resolvê-lo.

Heurística: Nearest Neighbor

A heurística construtiva escolhida para essa implementação foi o Nearest Neighbor, um algoritmo guloso que, iniciando em uma cidade X aleatória, sempre vai até a próxima cidade Y mais próxima não visitada, até que todas sejam percorridas. Então ele finalmente volta para a cidade inicial X e termina o trajeto.

A motivação por trás da escolha deste algoritmo em questão foi sua simplicidade e o fato de que geralmente é um caso base, um ponto de partida para pensar em heurísticas melhores. A literatura utilizada para compreender tais questões foi o livro **Local Search in Combinatorial Optimization**, capítulo 8 (Arts and Lenstra J. K., John Wiley and Suns, 1997).

Implementação

A linguagem de programação utilizada foi o C++14. ***Nota***: O código utiliza a biblioteca nativa ***std::experimental::filesystem*** para lidar com os arquivos de entrada. A partir

do C++17, ela não é mais experimental e deve ser usada como "*std::filesystem*" apenas. O código também está disponível no github através desse link:

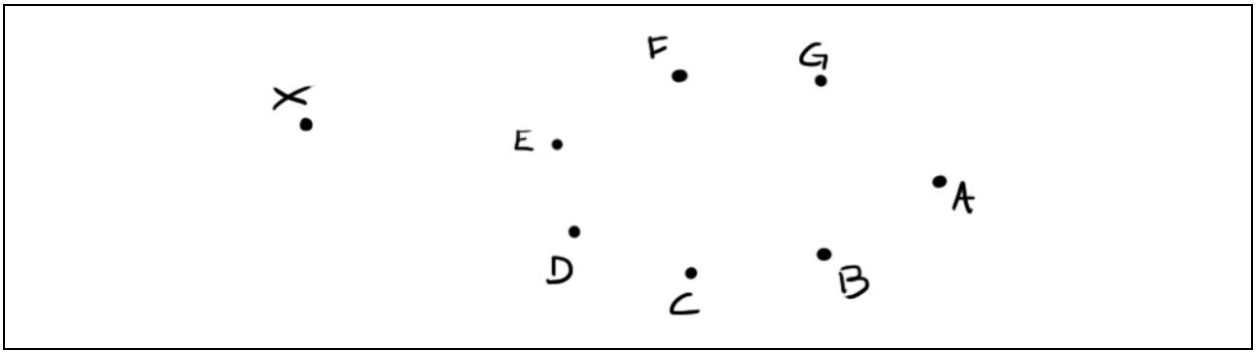
https://github.com/UnidayStudio/UFMG-TSP-Heuristics

O algoritmo funciona da seguinte forma:

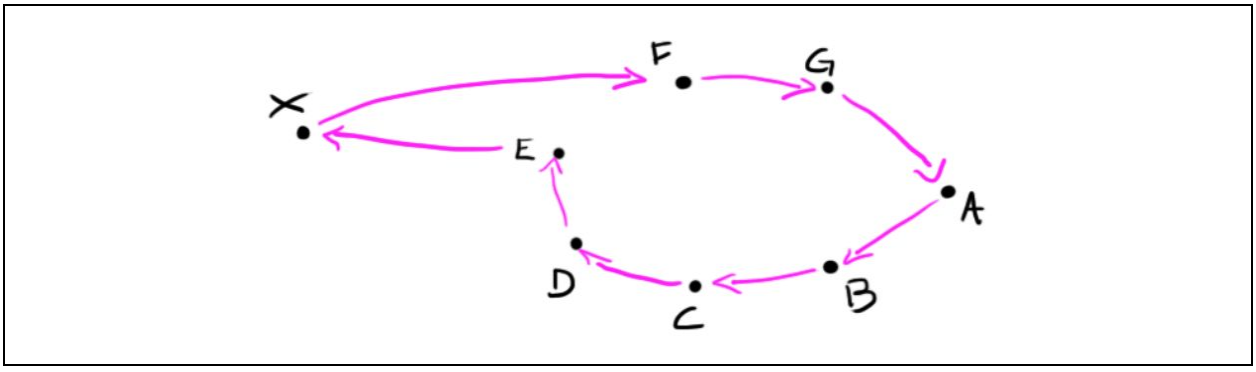
Ele irá percorrer todas as N cidades da entrada, começando da primeira (poderia ser qualquer cidade aleatória, mas para manter os resultados da execução determinísticos, foi fixado na primeira) e para cada uma das cidades, ele irá encontrar a mais próxima não visitada (percorrendo, verificando se já foi visitada e testando as distâncias de todas as N-1 outras cidades).

Esta implementação escolhida possui complexidade $O(n^2)$. (n=quantidade de cidades).

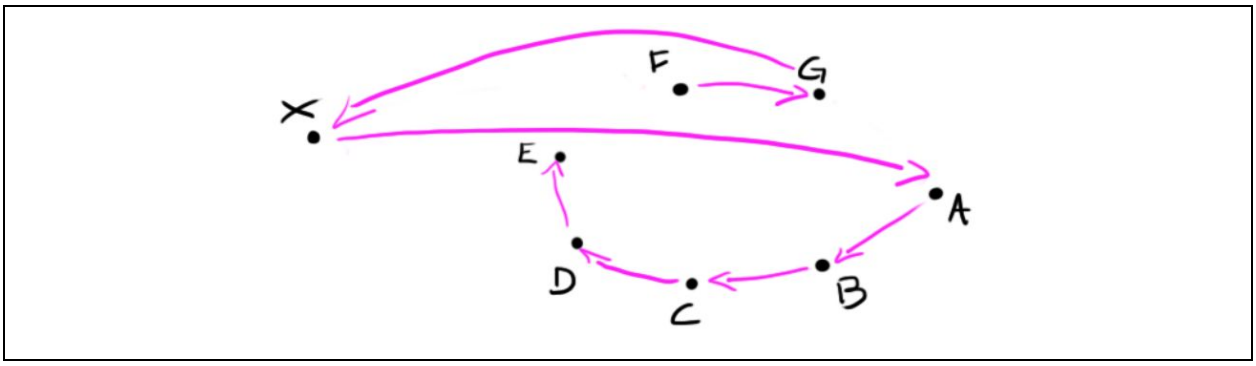
Problemas
Apesar de simples e eficiente em termos de performance comparado com outras heurísticas, essa abordagem possui alguns problemas. Por exemplo, considere o seguinte mapa de cidades:



Considere que o viajante começa na cidade A. Uma solução ótima seria passar pelas cidades, na ordem: A, B, C, D, E, X, F, G, A. Conforme na imagem abaixo:



Entretanto, por ser um algoritmo guloso que sempre considera a cidade mais próxima, ao chegar em E, ele irá identificar F primeiro e seguirá por esse caminho. Após isso ele só irá considerar X novamente ao terminar todas as outras cidades não visitadas, ou seja, depois de passar em G:



Perceba que tal decisão acarretará em um aumento considerável na distância percorrida pelo viajante.

Resultados

Cada uma das entradas e seus respectivos algoritmos de distâncias foram executadas **mil vezes** para obter a média de tempo de execução. Já que o algoritmo usa uma cidade inicial fixa (sempre a primeira), não houve diferenças no resultados da distância percorrida.

Abaixo a tabela com os resultados obtidos:

Valores de Referência		**Trabalho**		
Instância	Solução ótima	Custo	Tempo (s)	gap (%)
att48	10628	12813	0.000050	20.56%
berlin52	7542	8980	0.000053	19.07%
kroA100	21282	27807	0.000234	30.66%
kroA150	26524	33633	0.000521	26.80%

kroA200	29368	35859	0.000818	22.10%
kroB100	22141	29158	0.000204	31.69%
kroB150	26130	34499	0.000455	32.03%
kroB200	29437	36980	0.000806	25.62%
kroC100	20749	26227	0.000205	26.40%
kroD100	21294	26947	0.000201	26.55%
kroE100	22068	27460	0.000203	24.43%
lin105	14379	20356	0.000209	41.57%
pr76	108159	153462	0.000107	41.89%
pr107	44303	46680	0.000207	5.37%
pr124	59030	69297	0.000291	17.39%
pr136	96772	120769	0.000353	24.80%
pr144	58537	61652	0.000378	5.32%
pr152	73682	85699	0.000428	16.31%
rat99	1211	1554	0.000184	28.32%
rat195	2323	2752	0.000698	18.47%
st70	675	830	0.000098	22.96%
			Média	24.21%

Conclusão

A heurística Nearest Neighbor para o TSP, apenas de gulosa, apresentou um tempo de execução consideravelmente baixo e uma média de ***gap*** entre o resultado ótimo de cerca de 24%, com gap mínimo encontrado de 5.37% e máximo próximo de 50%. Ou seja, em média o caixeiro viajante percorrerá um caminho 24% maior do que o ótimo.

Tal heurística pode ser utilizada em ocasiões reais onde a garantia de um resultado ótimo não é essencial, como no caminho a ser percorrido por um agente de inteligência artificial em um Jogo (video game) de mundo aberto.

Referências

- **Local Search in Combinatorial Optimization**. Arts and Lenstra J. K., John Wiley and Suns, 1997. (Capítulo 8)