DCC215 Trabalho de Implementação

Vinicius Silva Gomes – 2021421869

Data: 24/09/2023

Heurística para o Problema do Caixeiro Viajante

Para este TI, foi implementada uma heurística construtiva para o Problema do Caixeiro Viajante (TSP) métrico. As instâncias foram disponibilizadas previamente e usam o formato de representação TSP-LIB para grafos métricos (euclidianos, para esse caso em específico) e completos, no qual os nós são pontos no plano e as arestas do grafo tem como peso a distância euclidiana entre os pontos. Apesar disso, uma das instâncias possui uma função de distância pseudo-euclidiana. A única diferença dessa instância em relação as outras é que a distância é calculada a partir de uma função diferente, todo o restante permanece o mesmo e a heurística será capaz de performar da mesma maneira.

A heurística escolhida para ser implementada foi o algoritmo *Twice Around the Tree*. Se trata de um algoritmo 2-aproximado para o problema, ou seja, o custo máximo para o circuito no pior caso será até 2 vezes maior que o custo ótimo para a mesma instância. Essa heurística se baseia na premissa da árvore geradora mínima (MST) ser uma bom início para a formação do circuito que percorre o grafo todo, ou seja, na ideia de que a MST deve possuir algumas arestas em comum com o circuito ótimo da instância.

Além disso, instâncias do TSP métrico são caracterizadas por satisfazerem a propriedade da desigualdade triangular: dados três pontos distintos P, Q e R no plano, temos que $dist(PR) < dist(PQ) + dist(QR)$. Ou seja, para três pontos que componham os vértices de um triângulo, a soma de dois lados sempre será maior que o o lado restante, sendo isso válido para todas as escolhas de lados possível. Essa propriedade será usada de maneira inteligente na composição do circuito, de modo que, onde for possível, sequências de duas arestas serão substituídas pela aresta que conecta o primeiro e o último nó, otimizando o custo do trecho.

Sendo assim, o que o algoritmo faz é, partindo da árvore geradora mínima do grafo, realizar uma travessia pré-ordem (*Depth-First Search* ou DFS) pela MST, anotando todos os nós que são visitados a cada passo. Ao final, todos os nós que foram acessados mais de uma vez podem ser excluídos da travessia, visto que, como os nós são pontos no plano, a desigualdade triangular vale e, com isso, podemos reduzir o custo de arestas que passam por nós já visitados, utilizando a aresta que conecta o nó anterior e o nó posterior ao nó que já havia sido visitado na travessia. Repetindo esse processo até atingir o nó inicial, será obtido um circuito que passa por todos os nós do grafo e, como dito anteriormente, tem o custo total de no máximo 2 vezes o valor do custo ótimo para a mesma instância, no pior caso.

A linguagem utilizada para implementar o algoritmo foi a linguagem Rust. Essa escolha foi feita com base em preferências individuais e em aspectos específicos da linguagem e de sua comunidade, como o fato dela ser uma linguagem compilada e de alto desempenho e apresentar uma biblioteca de grafos eficiente, razoavelmente bem mantida e que implementa uma vasta quantidade de algoritmos úteis (sobretudo os necessários para a implementação do *Twice Around the Tree*).

Executando o Código

Para executar o algoritmo, é necessário ter a linguagem Rust, juntamente com seu ecossistema, instalados na máquina. Para tanto, é recomendado que a instalação seja feita através da documentação oficial da linguagem[1].

Uma vez que o ambiente esteja configurado, o comando

```
cargo run -- -p "./instances/instance-name.tsp"
```

pode ser utilizado para executar o algoritmo para uma instância específica, disponível na pasta de instâncias, e então verificar qual foi o circuito obtido, o custo desse circuito e qual foi o tempo necessário para computá-lo.

Adicionalmente, um script foi desenvolvido para automatizar a execução do algoritmo para todas as instâncias disponíveis, de uma só vez. Executando

```
./run.sh
```

[1] https://www.rust-lang.org/tools/install

esse script será executado e os resultados obtidos para cada instância serão apresentados no terminal.

Resultados Obtidos

As instâncias disponibilizadas foram testadas contra o algoritmo implementado, sendo possível observar na tabela 1 o custo do circuito obtido e o tempo necessário para computar esse circuito/custo. Além disso, as instâncias no formato TSP-LIB também armazenam o custo ótimo para aquela instância, sendo possível analisar, posteriormente, o desempenho da heurística em relação ao valor ótimo.

Instância	**Custo**	**Ótimo**	**Razão**	**Tempo (ms)**
att48	15061.0	10628	1.42	0.9489
berlin52	9661.6787	7542	1.28	0.1071
kroA100	27763.3630	21282	1.30	0.4695
kroA150	36859.3564	26524	1.39	1.2635
kroA200	40664.5581	29368	1.38	2.9227
kroB100	26379.4449	22141	1.19	0.4464
kroB150	36076.4092	26130	1.38	1.1478
kroB200	38921.8958	29437	1.32	2.6402
kroC100	28466.3004	20749	1.37	0.4373
kroD100	27118.6766	21294	1.27	0.6505
kroE100	30575.2869	22068	1.39	0.5310
lin105	20433.4629	14379	1.42	0.4887
pr107	56305.7092	44303	1.27	0.5329
pr124	78977.9718	59030	1.34	1.1477
pr136	136550.1895	96772	1.41	0.9120
pr144	85331.5163	58537	1.46	1.4273
pr152	94227.9336	73682	1.28	1.2320
pr76	141087.9817	108159	1.30	0.3638
rat195	3413.1851	2323	1.47	2.8302
rat99	1731.6943	1211	1.43	0.4458
st70	881.2385	675	1.31	0.3036

Tabela 1: Tempo e custo obtidos para cada instância.

Como pode ser observado, apesar do algoritmo ser 2-aproximado, para as instâncias disponibilizadas, a pior razão obtida em relação ao ótimo foi de 1.46 em apenas uma das instâncias. Dessa forma, é possível concluir que a heurística é relativamente eficiente, apresentando resultados próximos do custo ótimo e que, dependendo da aplicação e da necessidade, ela pode ser completamente suficiente.

Além disso, a heurística possui uma complexidade computacional baixa, visto que as duas principais operações realizadas são o algoritmo de para determinar a árvore geradora mínima, com complexidade temporal $\mathcal{O}(E \log V)$, e a DFS, cuja complexidade temporal é $\mathcal{O}(V + E)$. Sendo assim, a complexidade da heurística é limitada pela complexidade de se determinar a árvore geradora mínima em um grafo e, portanto, é $\mathcal{O}(E \log V)$. Isso reflete diretamente na velocidade em que o algoritmo executa e na baixíssima quantidade de tempo despendida para computar o circuito para cada instância.

Por fim, a instância `att48` é a que possui a distância como função pseudo-euclidiana. Como é possível ver, aplicando a função da forma correta, a heurística implementada foi capaz de performar da maneira esperada, resultando num custo 1.42 vezes maior que o custo ótimo, o que respeita os limites provados da heurística.