DCC215 Trabalho de Implementação

Vinicius Silva Gomes – 2021421869

Data: 29/10/2023

Heurística baseada em GRASP para o TSP

O trabalho consiste na implementação de uma heurística baseada em *Greedy Randomized Adaptive Search Procedure* (GRASP) para o Problema do Caixeiro Viajante (*Traveling Salesman Problem* ou TSP) métrico. Esse trabalho é uma continuação direta dos trabalhos anteriores, no qual era proposto a implementação de uma heurística construtiva e uma heurística baseada em *Variable Neughborhood Descent* (VND) para o mesmo problema. Assim, as instâncias utilizadas no processo de experimentação permanecem as mesmas e, portanto, as discussões e propriedades extraídas sobre elas nas outras partes do trabalho continuam válidas.

A ideia por trás de heurísticas baseadas em GRASP é explorar diversos ótimos locais através de alguma estratégia de busca local (uma metaheurística também pode ser aplicada, como VND ou busca tabu). Para tanto, um componente aleatório é inserido no processo de construção da solução inicial na qual a busca local será aplicada. Essa componente permitirá que a busca local seja aplicada em diferentes soluções iniciais, que podem levar a diferentes ótimos locais. A aleatoriedade da solução obtida é ditada por um hiper-parâmetro α de modo que, quanto mais perto de 0 ele estiver, mais gulosa é a construção da solução e quanto mais próximo de 1 ele estiver, mais aleatória é a construção da solução.

Sendo assim, o procedimento inicial realizado pela metaheurística GRASP é construir iterativamente uma solução inicial para o problema. A cada passo de decisão (e.g., adicionar um vértice e, consequentemente, uma aresta à solução do TSP) uma lista restrita de candidatos é construída, de modo que a lista contenha apenas elementos cujo valor da função gulosa seja menor ou igual a seguinte equação:

$$c^{min} + \alpha \cdot (c^{max} - c^{min})$$

onde c^{min} e c^{max} representam o menor e o maior valor da função gulosa observado nessa iteração, respectivamente. Após a construção da lista, um item da lista é sorteado aleatoriamente e adicionado a solução. O processo, então, é repetido até que uma solução valida seja obtida.

Com a solução inicial construída, uma busca local será aplicada até que um ótimo local possa ser observado. Quando isso acontecer, o valor desse ótimo é comparado com o melhor valor observado até agora e atualizado, caso ele seja melhor. Todo esse processo de construção, busca local e atualização é repetido por N iterações, para que diversos ótimos locais possam ser explorados.

Na implementação realizada nesse TI, a função de vizinhança usada na busca local foi um 2-OPT e os valores de α e N foram:

$$\alpha = 0.4$$
$$N = 80$$

Além disso, a função gulosa a cada iteração é dada pelo custo de inserção da aresta que liga o último nó inserido na solução com os que ainda faltam ser inseridos (*Nearest Neighbor* ou heurística do Vizinho mais Próximo). Esse custo é calculado e ele que será usado na definição do valor limite para os nós que podem entrar na lista restrita de candidatos.

A linguagem de programação utilizada para implementar a heurística, assim como nas outras partes, foi a linguagem Rust. Os motivos para a escolha permanecem os mesmos citados anteriormente, acrescidos do fato que a implementação da função de vizinhança 2-OPT feita no trabalho anterior também foi em Rust. Como o GRASP possui uma fase de busca local, após a geração da solução aleatória, a implementação do 2-OPT do trabalho anterior pode ser aproveitada nessa fase da metaheurística.

Executando o código

Para executar a heurística, é necessário ter a linguagem Rust e 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

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

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

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

Adicionalmente, um script foi desenvolvido para automatizar a execução do algoritmo para todas as instâncias disponíveis. Ao executar

```
./run.sh
```

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

Resultados obtidos

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

Instância	**Distância**	**Custo**	**Ótimo**	**Razão**	**Tempo (ms)**
att48	pseudo-euclidiana	10765	10628	1.013	17.2841
berlin52	euclidiana	7811.0066	7542	1.036	21.5110
kroA100	euclidiana	21842.9618	21282	1.026	102.7864
kroA150	euclidiana	27213.8709	26524	1.026	397.6539
kroA200	euclidiana	30975.4466	29368	1.055	805.9582
kroB100	euclidiana	22655.1335	22141	1.023	101.6723
kroB150	euclidiana	27336.7200	26130	1.046	384.2217
kroB200	euclidiana	31402.7261	29437	1.067	717.2561
kroC100	euclidiana	21031.5889	20749	1.014	127.0124
kroD100	euclidiana	21918.2638	21294	1.029	128.5472
kroE100	euclidiana	22656.7244	22068	1.027	130.2019
lin105	euclidiana	14586.5979	14379	1.014	155.7917
pr107	euclidiana	45119.2315	44303	1.018	128.0129
pr124	euclidiana	59396.8465	59030	1.006	186.5917
pr136	euclidiana	100589.4755	96772	1.039	575.6388
pr144	euclidiana	59018.2815	58537	1.008	731.0117
pr152	euclidiana	75095.6381	73682	1.019	860.3110
pr76	euclidiana	108790.8464	108159	1.006	109.6303
rat195	euclidiana	2509.3912	2323	1.080	1675.7656
rat99	euclidiana	1283.9402	1211	1.060	230.2634
st70	euclidiana	688.4658	675	1.020	97.42008

Tabela 1: Tempo e custo obtidos para cada instância.

Comparando com os resultados obtidos para as implementações das outras partes do trabalho, o GRASP teve um desempenho excepcional, especificamente se comparado com o VND, de modo que o GRASP retorna resultados melhores que os do VND em um tempo muito mais curto. De fato, a pior razão $\frac{C_h}{C_o}$ observada foi de 1.080, que representa um valor muito próximo do ótimo para aquela instância.

Além disso, graças aos hiper-parâmetros que podem ser ajustados e a estratégia de busca local que pode ser feita de diversas formas, o processo de experimentação pode avaliar uma série de configurações diferentes e, então, determinar qual seria a melhor para aquele problema em específico ou para o conjunto de instâncias de interesse (configuração que melhora os resultados obtidos e minimiza o tempo de execução). Essas características fazem com que abordagens baseadas em GRASP e, além disso, combinações de GRASP com outra metaheurísticas, sejam fortes candidatas a serem usadas em aplicações no mundo real para resolver esse ou outros problemas semelhantes.