Relatório 2: Um Enigma das Galáxias

Lucas G. M. Miranda - 10265892
Marcela Tiemi Shinzato - 10276953
Sérgio Ricardo G. B. Filho - 10408386
Tiago Lascala Aude - 8936742

6 de dezembro de 2020

Introdução

A segunda parte do projeto *um enigma das galáxias* focou no aprimoramento do código e do modelo apresentados na primeira parte. Isso foi feito com o objetivo de viabilizar a resolução de instâncias maiores do problema do caixeiro viajante (TSP). Abaixo, apresentamos as mudanças feitas, juntamente com testes computacionais, que visam avaliar o quanto essas mudanças melhoraram aspectos diversos do programa.

Alterações

Nessa seção são descritas as alterações feitas nos componentes entregues na parte 1.

Alteração no modelo

A principal alteração foi feita no modelo utilizado para representar o problema. O modelo anterior era o seguinte:

$$\min \quad \sum_{i,j \in A} (\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \cdot Y_{ij})$$

sujeito a:

$$Y_{ij} = \{0, 1\},\ i, j \in A$$

$$\sum_{j=1}^{n} Y_{ji} = 1,\ i = 1, \ldots, n$$

$$\sum_{j=1}^{n} Y_{ij} = 1,\ i = 1, \ldots, n$$

$$\sum_{i,j \in S} Y_{ij} \leq |S| - 1,\ S \subseteq N - \{1\}, |S| \geq 2$$

Suas particularidades estão apresentadas no relatório da parte anterior, por isso não vamos focar em explicá-lo novamente. Um aspecto interessante de se ressaltar, porém, é o fato desse modelo representar a versão assimétrica do TSP. Essa versão considera que pode existir uma aresta (caminho) de uma cidade i para uma cidade j, e outra aresta, com custo diferente, de j para i. Esse não é o caso do problema que estamos resolvendo: a distância de i para j é a mesma que a de j para i. Em outras palavras, estamos resolvendo instâncias do *TSP simétrico*. O modelo foi, então, alterado para refletir melhor esse fato.

O novo modelo é definido em um grafo não-dirigido $G = (V, E)$ com um custo c_e para cada aresta $e \in E$. Ele está representado abaixo:

$$\min \quad \sum_{e \in E} (\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \cdot x_e)$$

sujeito a:

$$\sum_{e \in \delta(v)} x_e = 2, \; \forall \, v \in V \tag{1}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \; \forall \, S \subset V \; : 3 \leq |S| \leq |V| - 1 \tag{2}$$

$$x_e \in \{0, 1\}, \; \forall \, e \in E \tag{3}$$

Lembre-se que, ao resolver o problema, devemos obter uma rota ótima (a menor possível) que passa por todos os vértices. Vamos, em seguida, elucidar cada componente do modelo:

Função objetivo

$$\sum_{e \in E} (\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \cdot x_e)$$

Nela, existe a variável x_e, que é definida da seguinte maneira:

$$\forall \, e \in E, \; x_e = \begin{cases} 1, \text{ se a aresta } e \text{ faz parte da rota,} \\ 0, \text{ caso contrário} \end{cases}$$

Existe também o fator $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$, que é o custo da aresta e que liga os vértices i e j. Analisando esse custo, observa-se que ele é definido como a distância euclidiana entre i, que tem coordenadas (x_i, y_i), e j, que tem coordenadas (x_j, y_j).

Levando em conta tudo isso, conclui-se que a função objetivo retorna o custo da rota considerada, e o modelo está tentando minimizar esse custo.

Conjunto de restrições (1)

$$\sum_{e \in \delta(v)} x_e = 2, \; \forall \, v \in V$$

Embaixo do símbolo de somatório, observa-se a existência do conjunto $\delta(v)$. Trata-se do conjunto de arestas que tem uma de suas duas extremidades conectadas ao vértice v.

Tendo isso em vista, esse conjunto de restrições garante que as soluções factíveis representem grafos compostos por vértices que contêm apenas duas arestas conectadas em cada um.

Conjunto de restrições (2)

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \; \forall \, S \subset V \; : 3 \leq |S| \leq |V| - 1$$

Nessa conjunto de restrições, temos duas notações de conjuntos diferentes: S e $E(S)$. S é o conjunto de vértices que *podem* fazer parte de um subciclo (o número de vértices nesse conjunto é indicado por $3 \leq |S| \leq |V| - 1$). Já $E(S)$ é o composto pelas arestas que possuem as duas extremidades conectadas a vértices de S.

Ele garante a eliminação de subciclos ilegais. Como estamos modelando o TSP simétrico, não é preciso se preocupar com subciclos ilegais composto por dois vértices, pois existe apenas uma aresta possível entre dois vértices, o que torna impossível a formação de um subciclo.

Conjunto de restrições (3)

$$x_e \in \{0, 1\}, \; \forall \, e \in E$$

Esse conjunto de restrições apenas define os valores que x_e pode assumir.

O porquê da alteração

O modelo foi alterado porque a representação do problema como o TSP simétrico diminui o número de variáveis, o que faz com que os solvers levem menos tempo para processar uma solução ótima [1].

Alteração no solver utilizado

Na parte 1, o código entregue foi implentado na linguagem *Python* e utilizava um solver de programação por restrições (*CP-SAT*). Na correção dessa parte, nos foi dito que deveríamos usar um solver de programação linear inteira, por isso, para implementar a parte 2, usamos o *SCIP*, através do *Google OR-Tools*, sendo o código ainda implementado em *Python*.

Adicionando restrições ao problema

Uma estratégia que *não* foi alterada na segunda parte foi a maneira como resolvemos o problema e adicionamos suas restrições sucessivamente. Como essa é uma parte integral do projeto, cabe relembrá-la. O código implementado, em essência, executa a seguinte sequência de passos:

1. Implementa o modelo descrito sem as restrições de eliminação de subciclos ilegais;
2. Resolve-o;

3. Se a solução encontrada não possuir subciclos ilegais, então ela é ótima, e corresponde à resposta final;

4. Se a solução possuir subciclos ilegais, adiciona ao modelo as restrições de eliminação desses subciclos;

5. Resolve o problema novamente usando o modelo com as novas restrições e volta ao passo 2.

A detecção de subciclos ilegais no passo 2 é feita através da execução do algoritmo de busca em largura, implementado pela função `acha_subciclos`.

As restrições de eliminação de subciclos ilegais foram adicionadas dessa maneira para evitar gerar todas elas de uma vez só, visto que isso é custoso para casos em que o número de vértices é muito elevado.

Resolução do toy problem

A seguir, vamos detalhar a resolução do *toy problem* usando o novo modelo. Esse problema é composto por cinco vértices, com as seguintes coordenadas:

- (-310, 121)
- (-80, 170)
- (101, 391)
- (217, 97)
- (107, 40)

A primeira iteração resolve o problema sem nenhuma restrição de eliminação de subciclos ilegais:

$$\min \quad \sum_{e \in E} (\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \cdot x_e)$$

sujeito a:

$$\sum_{e \in \delta(v)} x_e = 2, \; \forall \, v \in V$$

$$x_e \in \{0, 1\}, \; \forall \, e \in E$$

A seguinte solução é retornada:

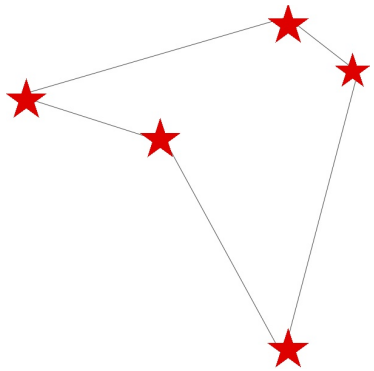

Figura 1: Primeira iteração do *toy problem*

Como essa solução não possui nenhum subciclo ilegal, trata-se da solução ótima [1].

Adições

Nessa seção são descritas as adições feitas aos componentes entregues na parte 1. Todas elas foram feitas com um único objetivo: chegar em soluções melhores e em menor tempo.

Heurística dos vizinhos mais próximos

Como foi explicado durante as aulas, fornecer uma solução inicial para o solver pode diminuir o tempo para que a solução ótima seja encontrada. *Heurísticas construtivas* são capazes de obter uma solução inicial. Dentre esse tipo de heurística, existe a do *vizinho mais próximo*. Escolhemos implementar ela pois é um procedimento relativamente simples [2]. Bastou aplicar a seguinte sequência de passos:

1. Começar no primeiro vértice;
2. Achar o vértice não-visitado mais próximo e visitá-lo;
3. Se ainda restar vértices não-visitados, voltar ao passo 2;
4. Retornar ao primeiro vértice.

Essa heurística foi implentada na função `nearest_neighboors`.

Heurística 2-opt

A solução inicial obtida pela heurística do vizinho mais próximo representa um bom começo, mas existem maneiras de melhorar essa solução, ou seja, obter outra com um custo menor. Uma dessas maneiras é aplicando a heurística *2-opt* [2]. Escolhemos ela e a implementamos na função `two_opt`, que utiliza `troca_two_opt`. A rota retornada por ela foi, então, convertida em solução inicial para o solver.

Algoritmo para conectar subciclos

A depender do número de vértices na instância de um problema, o programa implementado não consegue calcular uma solução ótima se definirmos um limite de tempo para o solver. Nesse caso o solver retorna uma solução não factível, ou seja, com subciclos ilegais. Como temos a intenção de retornar uma solução factível, implementamos um algoritmo para reconectar a rota retornada, de modo a desfazer os subciclos. Esse algoritmo está definido na função `elimina_subciclos`.

Para ilustrar o funcionamento da heurística, vamos executar o algoritmo no grafo abaixo:

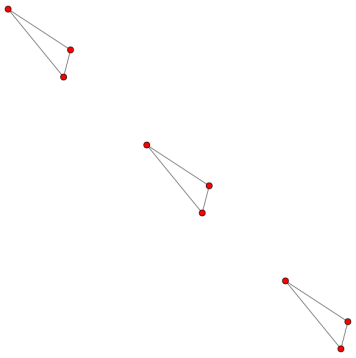

Figura 2: Rota com subciclos

A função recebe como parâmetro a matriz de adjacência que representa o grafo resultante da solução ótima e a usa para preservar parte da rota retornada pelo solver. O processamento começa a partir do primeiro nó (note que a rota sendo construída é representada pela cor verde):

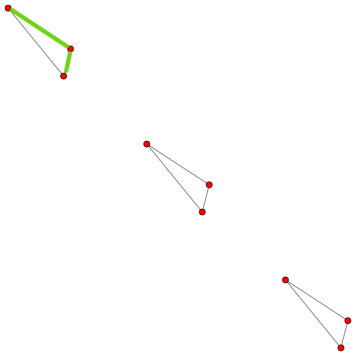

Figura 3: Parte da rota original é preservada

Quando a rota original forma um subciclo, a função conecta o último vértice processado ao seu vizinho mais próximo cuja conexão não formaria um subciclo:

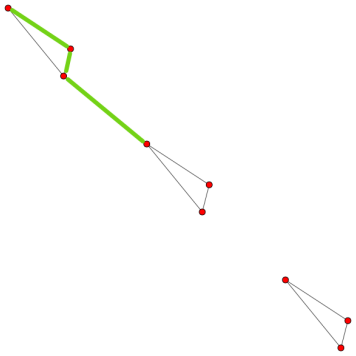

Figura 4: A formação de subciclos é impedida

Repetimos o processo para o vizinho mais próximo, e assim sucessivamente, até todos os vértices estarem ligados:

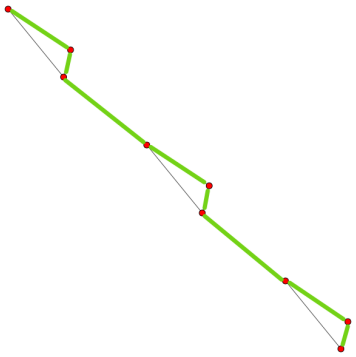

Figura 5: Todos os vértices ligados

Por fim, conectamos o primeiro vértice com o último:

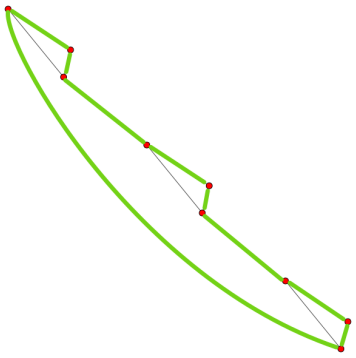

Figura 6: Rota final

Note que eliminamos os subciclos tentando minimizar o aumento do custo da rota. Fazemos isso ao tentarmos preservar ao máximo a rota original, e, quando isso é impossível, conectamos o vértice ao seu vizinho mais próximo. Além disso, tentamos melhorar a rota ainda mais com a heurística 2-opt.

Testes computacionais e discussão dos resultados

Para avaliar a eficiência do programa implementado, realizamos vários testes computacionais. Os testes foram feitos em uma máquina com processador Intel Core i7-9750H e 16GB de memória RAM. Para fazê-los, utilizamos as instâncias *Western Sahara*, *Djibouti*, *Qatar* e *Uruguay* da biblioteca disponível em Waterloo (2020) [3].

O primeiro teste foi realizado nas quatro instâncias, com um tempo limite de 30 minutos. Nele, não foi utilizado nenhum tipo de heurística para fornecer uma solução inicial ao solver, nem para melhorar a solução factível retornada pelo algoritmo de conectar subciclos (`elimina_subciclos`). Cabe notar, porém, que essa função só foi utilizada na instância *Uruguay*, pois em todas as outras, o solver foi capaz de retornar a solução ótima dentro do tempo estipulado.

Os resultados contendo informações do tempo de execução, número de nós explorados, custo e GAP estão dispostos na Tabela 1. A solução ótima para as 3 primeiras instâncias estão representadas em forma de grafos na Figura 7 e a melhor solução para a instância *Uruguay* nessas condições corresponde ao grafo da esquerda na Figura 8.

Resultado Sem Uso de Heurísticas						
Instâncias	**Estrelas**	**Tempo (min)**	**Nós Explorados**[1]	**Melhor Solução**	**GAP**[2] **(limitantes)**	**GAP**[3] **(relativo)**
W. Sahara	29	0.00098	1	27603	0%	0%
Djibouti	38	0.0014	1	6656	0%	0%
Qatar	194	0.98	3	9352	0%	0%
Uruguay	734	30	244	90553	14.81%	14.46%

Tabela 1: Resultados sem uso de heurísticas para as instâncias W. Sahara, Djibouti, Qatar e Uruguay.
1: nós explorados no último problema resolvido pelo solver
2: calculado a partir da diferença entre limitantes primal e dual
3: calculado a partir da diferença entre a nossa melhor solução e a melhor solução reportada em Waterloo (2020)[3]

Notamos que o aumento de 1 ordem de grandeza no número estrelas foi suficiente para causar um aumento de 5 ordens de grandeza no tempo de execução do algoritmo(sem resolvê-lo por completo). Assim, a partir dessa primeira tabela de resultados, podemos já identificar a natureza exponencial do problema que estamos resolvendo.

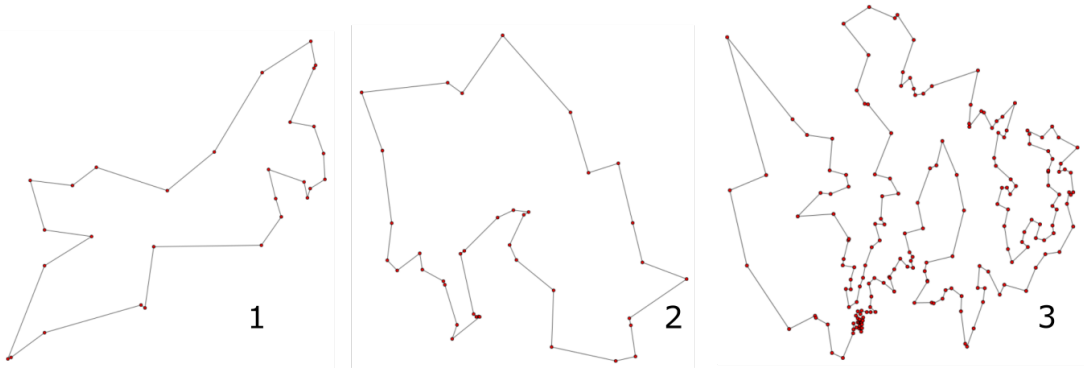

Figura 7: Solução ótima encontrada para as instâncias: Western Sahara (1), Djibouti (2) e Qatar (3)

Como já foi mencionado, o solver é incapaz de retornar uma solução ótima em 30 minutos para instância *Uruguay* (por causa do elevado número de vértices/estrelas). Tendo isso em vista, fizemos os próximos testes com o objetivo de avaliar o quanto as heurísticas contribuem para o cálculo de uma solução factível próxima da ótima. Testamos cinco métodos, que estão listados abaixo, juntamente com os seus resultados:

1. Não fornece solução inicial usando heurísticas, mas utiliza 2-OPT na solução final após conectar os subciclos.

2. Utiliza heurística dos vizinhos mais próximos para fornecer uma solução inicial, mas não utiliza heurística na solução final após conectar os subciclos.

3. Utiliza heurística dos vizinhos mais próximos para fornecer uma solução inicial e utiliza 2-OPT na solução final após conectar os subciclos.

4. Utiliza heurística 2-OPT para fornecer uma solução inicial, mas não utiliza heurística na solução final após conectar os subciclos.

5. Utiliza heurística 2-OPT tanto para fornecer uma solução inicial quanto para melhorar a solução final após conectar os subciclos.

Resultado com Heurísticas - Instância Uruguay					
Método	**Tempo (min)**	**Nós Explorados[1]**	**Melhor Solução**	**GAP[2] (limitantes)**	**GAP[3] (relativo)**
1	30	244	82400	4.472%	4.154%
2	30	1	95826	22.00%	21.12%
3	30	1	82779	5.388%	4.633%
4	30	1	83841	6.304%	5.975%
5	30	1	82296	4.519%	4.022%

Tabela 2: Resultados com uso de heurísticas para a instância Uruguay de acordo com os métodos de 1 a 5
1: nós explorados no último problema resolvido pelo solver
2: calculado a partir da diferença entre limitantes primal e dual
3: calculado a partir da diferença entre a nossa melhor solução e a melhor solução reportada em Waterloo (2020)[3]

O detalhe mais chamativo dessa tabela é o número de nós explorados no método 1, que é muito maior do que em todos os outros métodos. Isso acontece porque, sem o uso de heurísticas para fornecer uma solução inicial, o programa passa todo o tempo estipulado resolvendo o problema inicial (sem nenhuma restrição de eliminação de subciclos ilegais). Todo esse tempo, é, então, gasto explorando nós.

De maneira geral percebemos que o uso de heurísticas diminui consideravelmente os GAPs, sendo o método 5, o qual utiliza heurística 2-OPT no início e no final, o que forneceu o menor custo. Apesar disso, mais especificamente para o método 2, podemos notar que a utilização apenas da heurística dos vizinhos mais próximos, resultou num custo maior, com GAP (relativo) de 21.12%, do que resolver sem heurística apenas unindo os subciclos no final, como visto na Tabela 1, resultando em GAP (relativo) de 14.46%.

Abaixo, temos as representações em grafos das melhores soluções encontradas por cada método.

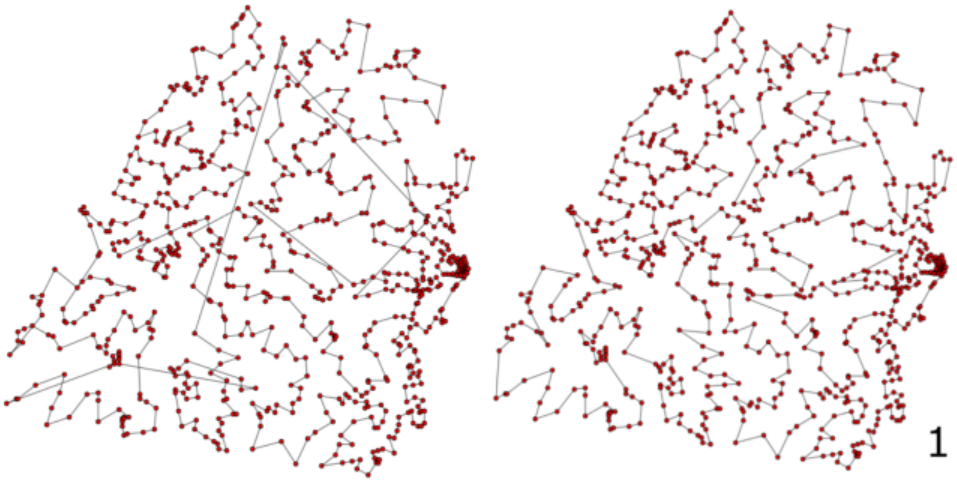

Figura 8: Melhor solução encontrada para a instância *Uruguay* sem utilização de heurísticas na solução inicial. Fez-se a união dos subciclos(esquerda) seguida pela aplicação da heurística 2-OPT(direita)

Comparando a Figura 9, a Figura 10 e os resultados da Tabela 2, podemos perceber uma visível melhora entre usar ou não a heurística 2-OPT após unir os subciclos. Note que na Figura 9 os grafos possuem conexões entre estrelas muito distantes enquanto que, na Figura 10, essa diferença é minimizada.

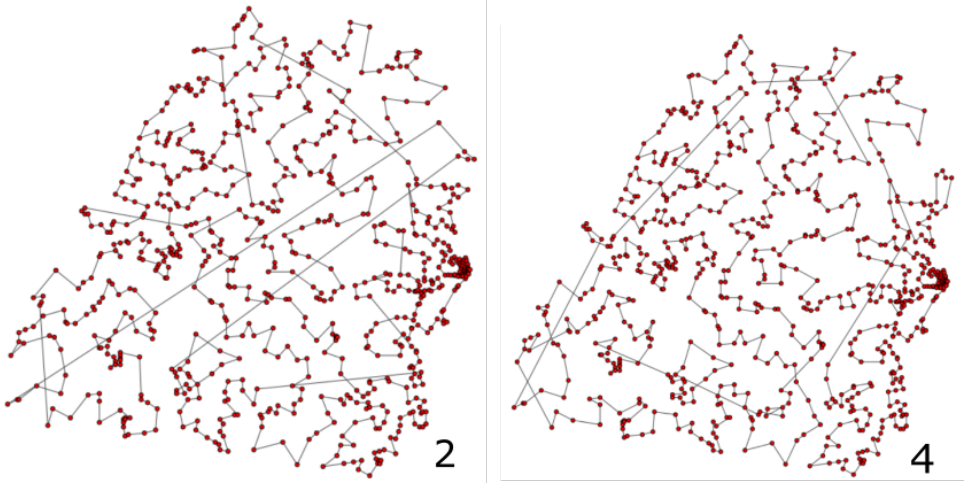

Figura 9: Melhor solução encontrada para a instância *Uruguay* pelos métodos (2) e (4)

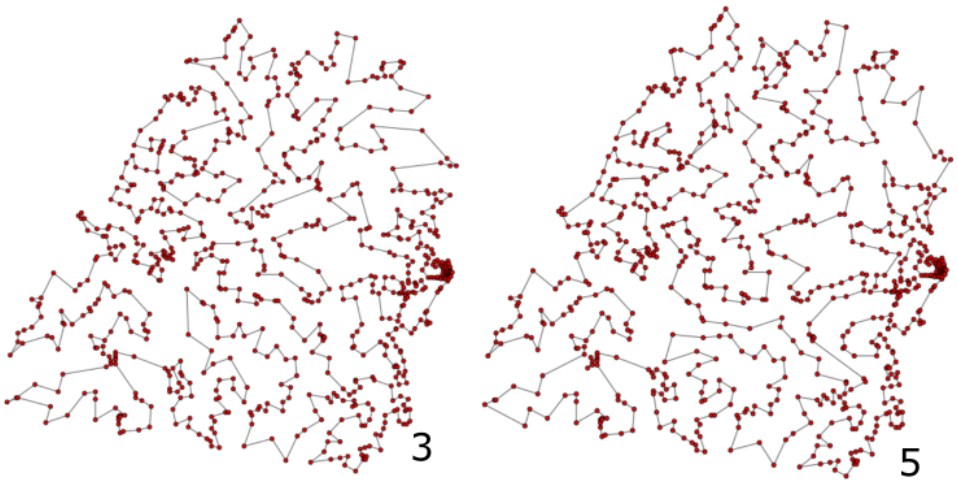

Figura 10: Melhor solução encontrada para a instância *Uruguay* pelos métodos (3) e (5)

Finalmente, tentamos também resolver as instâncias restantes de Waterloo (2020) [3] aplicando o mesmo método 5 utilizado para a instância *Uruguay*. No entanto, dada a limitação de 30 minutos para o tempo máximo de execução do solver, as soluções encontradas foram iguais à dada como solução inicial usando a heurística 2-OPT. Além disso, dadas as características do problema e a capacidade da máquina utilizada, a quantidade de memória RAM foi insuficiente para conseguir resolver além da 11° instância em diante.

A seguir disponibilizamos uma tabela resumindo o GAP relativo à melhor solução conhecida para as 10 primeiras instâncias que conseguimos resolver. Considerando um GAP de 100% para todas as 17 instâncias não resolvidas a média de GAPs obtida foi de 65.6852%.

Instância	**Estrelas**	**Custo Encontrado**	**Custo Conhecido**	**GAP (relativo)**
W. Sahara	29	27603	27603	0%
Djibouti	38	6656	6656	0%
Qatar	194	9352	9352	0%
Uruguay	734	82296	79114	4.022%
Zimbabwe[1]	929	101263	95345	6.207%
Luxembourg[1]	980	12241	11340	7.945%
Rwanda[1]	1621	28214	26051	8.303%
Oman[1]	1979	94184	86891	8.393%
Nicaragua[1,2]	3496	103351	96132	7.509%
Canada[1,2]	4663	1691882	1290319	31.12%

Tabela 3: Resultados de GAPs para as 10 primeiras instâncias de Waterloo (2020)[3]
1: Solução final foi a mesma fornecida como solução inicial utilizando a heurística 2-OPT
2: Tempo para encontrar a solução inicial com a heurística 2-OPT ultrapassou 30 minutos

Conclusão

Pelo que foi analisado, percebemos que as heurísticas implementadas ajudaram consideravelmente a atingir soluções factíveis com custos reduzidos durante a resolução de instâncias com um elevado número de vértices. Isso leva a crer que tal custo poderia ser reprimido ainda mais com algoritmos de eliminação de subciclos e heurísticas melhores. Outra opção seria alterar os parâmetros do solver, que foi uma estratégia recomendada pelas docentes, mas que optamos por não seguir.

Referências bibliográficas

[1] DE GIOVANNI, L.; DI SUMMA, M. Methods and Models for Combinatorial Optimization - Exact methods for the Traveling Salesman Problem. Disponível em: `https://www.math.unipd.it/~luigi/courses/metmodoc1819/m08.01.TSPexact.en.pdf`. Acesso em: 30/11/2020.

[2] NILSSON, C. Heuristics for the Traveling Salesman Problem. Disponível em: `http://160592857366.free.fr/joe/ebooks/ShareData/Heuristics%20for%20the%20Traveling%20Salesman%20Problem%20By%20Christian%20Nillson.pdf`. Acesso em: 03/12/2020.

[3] WATERLOO, U. (2020). University of Waterloo, National Traveling Salesman Problems. Disponível em: `http://www.math.uwaterloo.ca/tsp/world/countries.html`. Acesso em: 30/11/2020.