

Atividade 2

Modelagem de Problema de Programação Linear Inteira

Grupo 1

Ítalo Fernandes Gonçalves - RA: 234990

Rebecca Moreira Messias - RA: 186416

Vinicius da Silva - RA: 206734

Abril de 2022

MC859/MO824

1 Objetivo

Nessa atividade será modelado um problema com Programação Linear Inteira [4] utilizando *lazy constraints* e solucionado com o software Gurobi [1]. O problema que será tratado nessa atividade é do k-TSP, uma variação do problema do TSP [5] (problema do caixeiro viajante), e ele será explicado melhor na próxima sessão.

As próximas sessões consistem na descrição do problema, na formulação matemática, geração das instâncias, experimentos e análise dos resultados.

2 Descrição do problema

O problema a ser trabalhado nessa atividade é do ~~k travelling salesman problem~~ (k-TSP), uma variação do *travelling salesman problem* (TSP). Tratando-se somente do TSP, o objetivo consiste em encontrar um ciclo Hamiltoniano [6] de custo mínimo. Para isso, é considerado um grafo não-orientado completo $G(V, E)$, onde V é o conjunto dos vértices e E é o conjunto das arestas. Em cada aresta $e \in E$ há um custo $c_e : E \rightarrow \mathbb{R}$. Já o objetivo do k-TSP é encontrar dois ciclos Hamiltonianos com custo total mínimo, tal que pelo menos k arestas do grafo sejam visitadas por ambos os ciclos. Para isso, é considerado um grafo não-orientado completo $G(V, E)$, onde V é o conjunto dos vértices e E é o conjunto das arestas em que cada aresta $e \in E$ há dois custos:

$$c_e^1, c_e^2 : E \rightarrow \mathbb{R}^+$$

O parâmetro k define a similaridade entre os ciclos, de modo que $k = |V|$ implica que uma solução factível corresponda a dois ciclos Hamiltonianos contendo as mesmas $|V|$ arestas, enquanto $k = 0$ resulta em uma solução factível ser qualquer par de ciclos Hamiltonianos, visto que a restrição exige ter pelo menos k arestas em comum.

3 Desenvolvimento

3.1 Formulação Matemática

Definição das variáveis)

Tendo em vista o problema apresentado, sua formulação pode ser escrita como:

$$\text{minimizar} \quad \sum_{e \in E} x_e^1 \cdot c_e^1 + \sum_{e \in E} x_e^2 \cdot c_e^2 \quad (1)$$

$$\text{sujeito a} \quad \sum_{e \in \delta(i)} x_e^1 = 2 \quad \forall i \in V, \quad (2)$$

$$\sum_{e \in \delta(i)} x_e^2 = 2 \quad \forall i \in V, \quad (3)$$

$$\sum_{e \in E} D_e \geq k \quad (4)$$

$$\sum_{e \in E(S)} x_e^1 \leq |S| - 1 \quad \forall S \subset V, \quad (5)$$

$$\sum_{e \in E(S)} x_e^2 \leq |S| - 1 \quad \forall S \subset V, \quad (6)$$

$$x_e^1 \geq D_e \quad \forall e \in E, \quad (7)$$

$$x_e^2 \geq D_e \quad \forall e \in E, \quad (8)$$

$$x_e^1 = \{0, 1\} \quad \forall e \in E, \quad (9)$$

$$x_e^2 = \{0, 1\} \quad \forall e \in E, \quad (10)$$

$$D_e = \{0, 1\} \quad \forall e \in E, \quad (11)$$

A equação (1) expressa a função objetivo do problema, estando sujeito as restrições de (2) - (11). As restrições em (2) e (3) garantem que para cada vértice seja conectado por 2 arestas, em cada conjunto de aresta, para que seja formado o ciclo hamiltoniano desejável. A restrição (4) garante que pelo menos k arestas sejam utilizadas ~~por os ciclos~~. A restrição em (5) e (6) garante que exista somente um único ciclo para cada TSP. As restrições (7) e (8) garantem que se uma aresta foi duplicada então ela deve ser usada. As restrições (9) a (11) garantem que as variáveis criadas (x_e^1 , x_e^2 e D_e) sejam binárias.

3.2 Geração das Instâncias

As instâncias foram geradas de acordo com o arquivo de texto que pode ser encontrado neste [link](#), composto de 250 linhas. Cada linha é formada por 4 números separados por espaço,

sendo os 2 primeiros a primeira coordenada, x^1 e y^1 , e os 2 últimos a segunda coordenada, x^2 e y^2 . Para o cálculo do custo de cada aresta, para cada TSP, foram realizados os cálculos das equações (12) e (13), para todo par de coordenadas de cada TSP. Cada uma dessas equações, é a distância euclidiana entre cada ponto, respectivamente do primeiro e segundo TSP.

$$c_e^1 = \sqrt{(x_i^1 - x_j^1)^2 + (y_i^1 - y_j^1)^2} \quad \forall i \in V, \forall j \in V, i \neq j \quad (12)$$

$$c_e^2 = \sqrt{(x_i^2 - x_j^2)^2 + (y_i^2 - y_j^2)^2} \quad \forall i \in V, \forall j \in V, i \neq j \quad (13)$$

Foram geradas 12 instâncias a partir da combinação de 4 conjuntos de vértices ($|V| = \{100, 150, 200, 250\}$) e três valores de similaridade ($k = \{0, \frac{|V|}{2}, |V|\}$) para testar a efetividade do modelo construído. Cada instância foi gerada com as $|V|$ primeiras linhas do arquivo.

4 Resultados e discussões

Para a realização do experimento, as instâncias foram geradas em uma máquina com sistema operacional Ubuntu 21.10, processador Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz e 8Gb de memória RAM. A linguagem de programação Python [2] foi escolhida por sua facilidade de desenvolvimento e praticidade fornecida pela biblioteca *gurobipy*.

A implementação das *lazy constraints* foi realizada por meio do uso de *Callbacks*. A cada nova solução factível obtida, inferências são feitas com o objetivo de adicionar restrições que reduzem o espaço de busca. No caso do k-TSP, é adicionada a restrição para eliminar *subtours*.

Para desenvolver o código do k-TSP, foi feita uma adaptação de um dos códigos de exemplos fornecidos pelo Gurobi, com a solução do TSP com *lazy constraints* [3]. A adaptação consistiu na formulação para dois TSPs, inclusive suas restrições, e criada restrições para ambos os TSPs, na estratégia de *lazy constraint* específica para compartilharem um número mínimo de arestas pelo menos K ? -. A função *Callback* foi duplicada e aplicada em cada TSP, ao encontrar soluções factíveis inteiros.

A Tabela 1 mostra os resultados para cada instância. As três primeiras colunas indicam a identificação da instância e a configuração dessa instância, indicando os valores de $|V|$ e k . As próximas colunas exibem o tempo de execução em segundos, o custo da solução e por fim o

Instância	$ V $	k	Tempo de execução (s)	Custo da solução	GAP de otimalidade (%)
1	100	0	5,08	1.630	0
2	100	50	59,44	2.102	0
3	100	100	3,73	3.463	0
4	150	0	43,32	1.966	0
5	150	75	551,9	2.748	0
6	150	150	15,74	4.780	0
7	200	0	88,51	2.308	0
8	200	100	1.800,01	3.905	13,2394
9	200	200	95,86	6.003	0
10	250	0	295,71	2.597	0
11	250	125	1.800,01	-	-
12	250	250	305,22	6.999	0

Tabela 1: Valores obtidos em cada instância.

GAP de otimalidade informado pelo Gurobi em suas execuções. A execução de cada instância foi limitado em 30 minutos (1.800 segundos).

O código desenvolvido e as saídas de cada execução podem ser encontradas nesse [link](#).

4.1 Análise

Pode-se notar que o algoritmo tem melhor performance quando $k = 0$ e $k = |V|$, apresentando os menores tempos de execução quando comparados a $k = \frac{|V|}{2}$. O Gurobi encontrou o ótimo para todas instâncias, exceto para casos em que o espaço de busca é estritamente grande como se pode notar nas instâncias 8 e 11, que refletem o fato anterior mencionado. Nesses casos, usando a estratégia de *lazy constraints*, encontrar soluções factíveis é probabilisticamente mais complexo e consequentemente a quantidade de adição de novas restrições é quase nula, fato que ocorreu na instância 11, em que nenhuma solução foi encontrada dentro do tempo limite de 30 minutos. Já quando lidamos com um k muito alto ou muito baixo, isto é, em seus valores extremos, o espaço de busca é consideravelmente menor já que k está diretamente atrelado ao fato de encontrar uma solução factível ou não, isso implica um maior numero de restrições adicionadas e por sua vez, um espaço de busca significativamente reduzido.

contraditório, pois quando $K=0$ o espaço de soluções é máximo (qualquer per de ciclos Hamiltonianos).

Referências

- [1] Gurobi documentation. <https://www.gurobi.com/documentation/9.5/>.
- [2] Python documentation. <https://docs.python.org/3/>. Accessed: 2022-04-09.
- [3] tsp.py. https://www.gurobi.com/documentation/9.0/examples/tsp_py.html. Accessed: 2022-04-09.
- [4] Fábio Luiz Usberti and Celso Cavellucci. Integer programming.
- [5] Wikipedia contributors. Travelling salesman problem — wikipédia, a enciclopédia livre. https://en.wikipedia.org/w/index.php?title=Travelling_salesman_problem&oldid=1080774476, 2022. [Online; accessed 9-April-2022].
- [6] Wikipédia. Caminho hamiltoniano — wikipédia, a enciclopédia livre, 2021. [Online; accessed 7-janeiro-2021].

MO824 | Atividade 2

Nome Adolfo Schneider

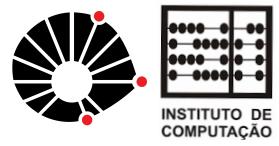
RA 234991

Nome Pedro Mota

RA 185853

Nome Fabio Stori

RA 196631



1. Introdução

O objetivo desta atividade consiste na modelagem em programação linear e otimização do *k-similar Travelling Salesman Problem* (kSTSP) utilizando *lazy constraints* no software Gurobi. O problema modelado é descrito a seguir:

Seja um grafo não-orientado completo $G(V, E)$, onde V é o conjunto dos vértices e E é o conjunto das arestas. Em cada aresta $e \in E$ há dois custos $c_e^1, c_e^2 : E \rightarrow R^+$. O objetivo do problema consiste em encontrar dois ciclos Hamiltonianos com custo total mínimo, tal que pelo menos k arestas do grafo sejam visitadas por ambos os ciclos. O parâmetro k define a similaridade entre os ciclos, de modo que $k = |V|$ implica que uma solução factível corresponda a dois ciclos Hamiltonianos disjuntos nas arestas, enquanto $k = 0$ resulta em uma solução factível ser qualquer par de ciclos Hamiltonianos

2. Modelo

Para modelar o problema apresentado, usou-se três variáveis de decisão:

$x_{i,i}^1$: variável binária, 1 se a aresta i faz parte do ciclo hamiltoniano 1, e 0 caso contrário

$x_{i,i}^2$: variável binária, 1 se a aresta i faz parte do ciclo hamiltoniano 2, e 0 caso contrário

$y_{i,i}$: variável binária, 1 se a aresta i está em ambos os ciclos, 0 caso contrário

A grandeza a ser minimizada neste problema é a soma do custo total dos dois ciclos. A *Equação 1* expressa o custo total dos ciclos de forma matemática. Onde $c^1(i)$ e $c^2(i)$ indicam o custo da aresta para o ciclo 1 e 2, respectivamente.

$$\sum_{i \in E} (x_{i,i}^1 \cdot c^1(i) + x_{i,i}^2 \cdot c^2(i)) \quad (1)$$

As *Equações 2* e *3* expressam a restrição de que cada vértice deve ter duas arestas incidentes (uma "de entrada" e outra "de saída").

$$\sum_{i \in \delta(i)} x_{i,i}^1 = 2, \forall i \in V \quad (2)$$

$$\sum_{i \in \delta(i)} x_i^2 = 2, \forall i \in V \quad (3)$$

As restrições das Equações 4 e 5 eliminam subciclos ilegais ao impedir que em qualquer subconjunto de vértices existam mais do que $|S|-1$ arestas, onde S é um subconjunto próprio de vértices.

$$\sum_{i \in E(S)} x_i^1 \leq |S| - 1, \forall S \subset V \quad (4)$$

$$\sum_{i \in E(S)} x_i^2 \leq |S| - 1, \forall S \subset V \quad (5)$$

As Equações 6, 7, 8 e 9 expressam as restrições de similaridades entre os ciclos. As Equações 6, 7 e 8 expressam a definição de que, se $y_i = 1$, a aresta i pertence aos dois ciclos e que caso i pertença a apenas 1 ciclo então $y_i = 0$. Já a Equação 9 define a restrição de que pelo menos k arestas estão presentes nos dois ciclos (x_i^1 e x_i^2).

ou nenhum ciclo

$$x_i^1 \geq y_i$$

(6)

$$x_i^2 \geq y_i$$

(7)

$$x_i^1 + x_i^2 - 1 \geq y_i$$

(8)

$$\sum_{i \in E} y_i \geq k$$

(9)

domínio das restrições?

essa restrição não é válida para o problema. Consegue perceber porque?

Domínio das variáveis?

3. Resultados

Implementamos o modelo construído para o problema com a biblioteca gurobipy, implementação do software Gurobi para a linguagem de programação Python. Na execução dos experimentos computacionais, foi utilizado um computador com 32GB de memória RAM, processador Intel® Core™ i5-8265U CPU @ 1.60GHz × 8 e sistema operacional Ubuntu 20.04.

O código implementado utiliza-se das coordenadas fornecidas e executa o modelo um total de 12 vezes, variando entre as possíveis combinações de N e K , onde

$$N = \{100, 150, 200, 250\} \quad (8)$$

$$K = \{0, N/2, N\} \quad (9)$$

conforme definido no enunciado da atividade. Os resultados para cada instância do modelo são apresentados na Tabela 1, e Gráficos 1 e 2.

N	K	Custo Ótimo	Gap de Otimalidade	Tempo de Execução (segundos)
100	0	1630	0.0000%	10.71
100	50	2102	0.0000%	115.75
100	100	3463	0.0000%	9.02
150	0	1966	0.0000%	52.61
150	75	2748	0.0000%	693.45
150	150	4780	0.0000%	42.61
200	0	2308	0.0000%	153.94
200	100	3390*	-	1800**
200	200	6003	0.0000%	49.27
250	0	2597	0.0000%	540.69
250	125	3912*	-	1800**
250	250	6999	0.0000%	426.44

Tabela 1: Tabela com os dados de execução do modelo para cada conjunto de N e K iterados.

*Último valor exibido antes da interrupção pelo limite de tempo

**Execução não chegou ao fim e alcançou o tempo limite de 30 minutos de execução

primal ou dual?

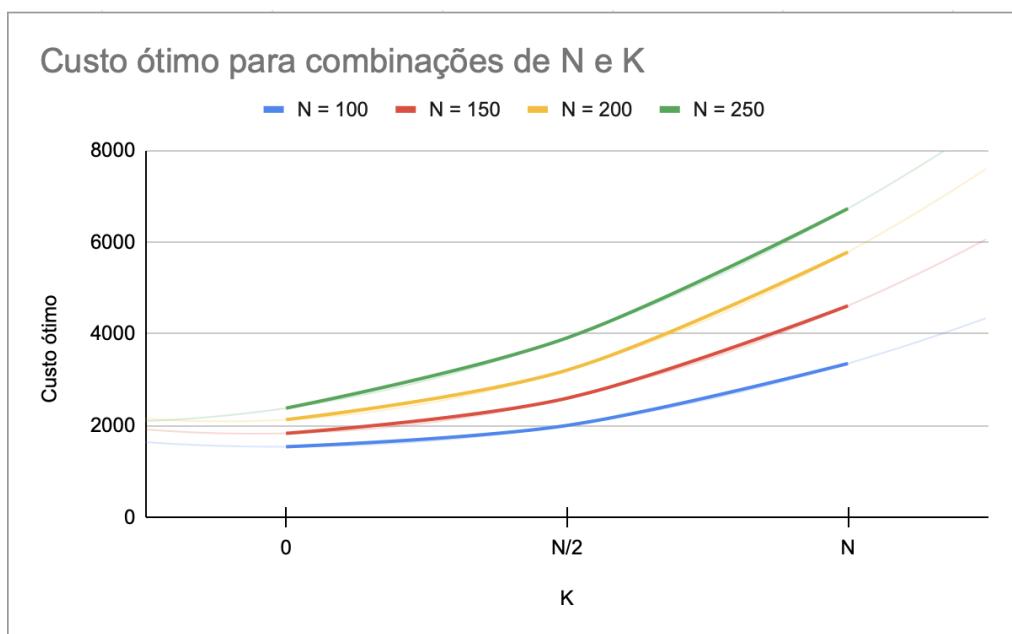
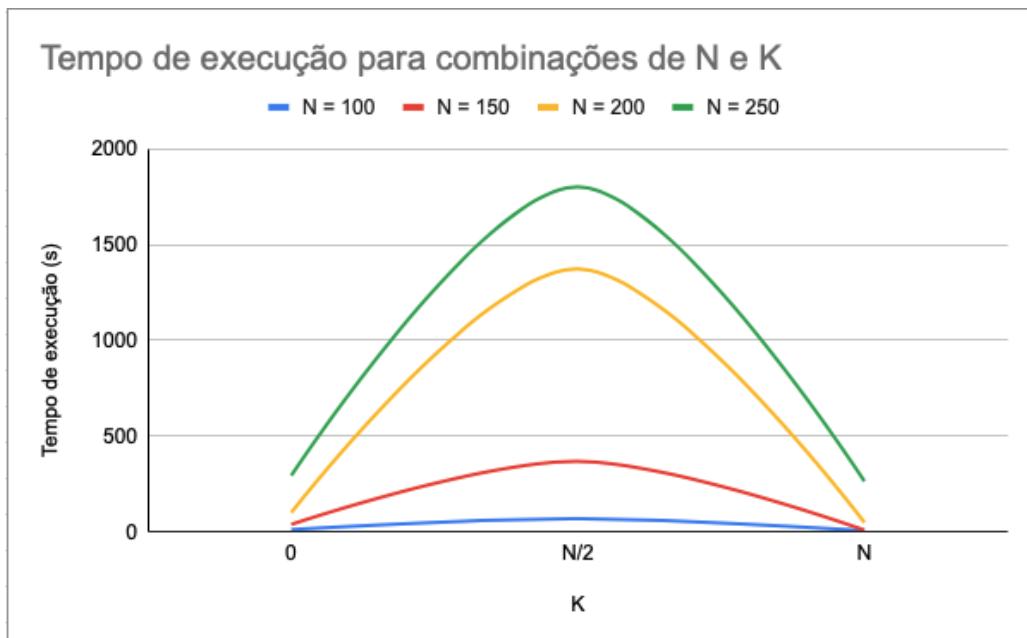


Gráfico 1: Gráfico mostrando o custo ótimo do modelo para cada conjunto de N e K.



muito
bons
gráficos

Gráfico 1: Gráfico mostrando o tempo de execução do modelo para cada conjunto de N e K .

3. Análise

ítalo

A **Tabela 1** e **Gráfico 1** mostram um padrão de crescimento do custo ótimo para as instâncias do modelo. Como era esperado, quanto maior o valor de N , maior o valor do custo, já que há mais vértices para serem visitados. Além disso, o custo ótimo é proporcional ao grau de similaridade K entre os ciclos. Para todos os valores de N , o custo mínimo é menor quando $K = 0$, e maior quando $K = N$. Isso acontece porque quando $K = 0$ é possível escolher a melhor combinação de arestas que formam os dois ciclos com custo total mínimo, sem a necessidade ter uma ou mais arestas em comum nos ciclos, o que poderia aumentar o custo da solução para satisfazer essa restrição. Já quando $K = N$ os dois ciclos precisam conter as mesmas arestas, diminuindo o número de soluções factíveis e de custos menores para satisfazer essa restrição.

A **Tabela 1** e **Gráfico 2** também mostram um padrão de crescimento do tempo de execução para as instâncias do modelo. O **Gráfico 2** mostra que para todos os valores de K , um aumento em N gera um aumento no tempo de execução. Além disso, observando o tempo de execução para as 3 variações de K para um mesmo N , é possível perceber um menor tempo de execução quando $K = N$ em todos os casos. Isso acontece porque quando $K = N$, os dois ciclos devem ser exatamente iguais, simplificando o problema a encontrar apenas um ciclo hamiltoniano.

4. Conclusão

→ $N \geq 2$ verdade esse fenômeno pode ser explicado pelo problema combinatório de escolher quais K dentre $\binom{N}{2}$ arestas ($\binom{N}{K}$) serão escolhidas pela restrição de similaridade.

A partir dos resultados e análise deste relatório, é possível concluir que: o custo mínimo é proporcional ao valor de N . E que para um mesmo valor de N , quanto maior o grau de similaridade K entre os ciclos, maior será o custo mínimo. Além disso, o tempo de execução também é proporcional ao valor de N . E que para um mesmo N , o tempo de execução é mínimo quando $K = N$, e máximo para $K = N/2$.

Atividade 2 - Programação Linear

Felipe Domingues RA 171036

Athyrsen M. Ribeiro RA 203963

Adivair Santana Ramos RA 193325

1. Introdução

Descrição do problema

O kSTSP pode ser descrito da seguinte forma. Seja um grafo não-orientado completo $G(V, E)$, onde V é o conjunto dos vértices e E é o conjunto das arestas. Em cada aresta $e \in E$ há dois custos $c_e^1, c_e^2 : E \rightarrow \mathbb{R}^+$. O objetivo do problema consiste em encontrar dois ciclos Hamiltonianos com custo total mínimo, tal que pelo menos k arestas do grafo sejam visitadas por ambos os ciclos. O parâmetro k define a similaridade entre os ciclos, de modo que $k = |V|$ implica que uma solução factível corresponda a dois ciclos Hamiltonianos disjuntos nas arestas, enquanto $k = 0$ resulta em uma solução factível ser qualquer par de ciclos Hamiltonianos. Faça a formulação em programação linear inteira do STSP.

Você deve gerar 12 instâncias, combinando quatro conjuntos de vértices ($|V| = 100, 150, 200, 250$) e três valores de similaridade ($k = 0, \frac{|V|}{2}, |V|$). Será fornecido um arquivo texto com coordenadas inteiras (x_i^1, y_i^1) e (x_i^2, y_i^2) para $i = 1, \dots, 250$.

Os dados no arquivo estarão organizados conforme abaixo:

<coordenada x_1^1 ><coordenada y_1^1 ><coordenada x_1^2 ><coordenada y_1^2 >

<coordenada x_2^1 ><coordenada y_2^1 ><coordenada x_2^2 ><coordenada y_2^2 >

A partir dessas coordenadas você deverá gerar os custos c_e^1 e c_e^2 de uma aresta $e = (i, j)$ calculando o teto da distância Euclidiana dos pares de coordenadas, conforme as equações abaixo:

$$c_e^1 = \left\lceil \sqrt{(x_i^1 - x_j^1)^2 + (y_i^1 - y_j^1)^2} \right\rceil$$

$$c_e^2 = \left\lceil \sqrt{(x_i^2 - x_j^2)^2 + (y_i^2 - y_j^2)^2} \right\rceil$$

ISSO NÃO
FAZ PARTE
DA DESCRIÇÃO
DO PROBLEMA

enunciado
atualizado

atenção ao
que for
copiar do
enunciado

2. Modelo Matemático

Um modelo de programação linear inteira para o kTSP é apresentado a seguir:

$$(P) \quad \text{minimize} \sum_{e \in E} c_e^1 \cdot x_e^1 + \sum_{e \in E} c_e^2 \cdot x_e^2 \\ \text{sujeito a} \quad \sum_{e \in \delta(i)} x_e^1 = 2 \quad \forall i \in V \quad (2.1)$$

$$\sum_{e \in \delta(i)} x_e^2 = 2 \quad \forall i \in V \quad (2.2)$$

$$\sum_{e \in E(S)} x_e^1 \leq |S| - 1 \quad \forall S \subset V \quad (2.3)$$

$$\sum_{e \in E(S)} x_e^2 \leq |S| - 1 \quad \forall S \subset V \quad (2.4)$$

$$\sum_{e \in E} D_e \geq k \quad (2.5)$$

$$x_e^1 + x_e^2 \geq 2D_e \quad \forall e \in E \quad (2.6)$$

$$x_e^1, x_e^2, D_e \in \{0, 1\} \quad \forall e \in E \quad (2.7)$$

Onde:

- x_e^1 é uma variável de decisão binária associada à presença ($x_e^1 = 1$) ou não ($x_e^1 = 0$) da aresta e na solução do primeiro ciclo Hamiltoniano;
- x_e^2 é uma variável de decisão binária associada à presença ($x_e^2 = 1$) ou não ($x_e^2 = 0$) da aresta e na solução do segundo ciclo Hamiltoniano;
- c_e^1 refere-se ao custo da aresta e no primeiro ciclo Hamiltoniano
- c_e^2 refere-se ao custo da aresta e no segundo ciclo Hamiltoniano
- $\delta(i)$ é o conjunto de arestas que incidem no vértice i ;
- $S \in V$ é um subconjunto próprio de vértices;
- $E(S)$ é o conjunto das arestas cujos dois vértices terminais estão em S .
- D_e é uma variável de decisão binária associada a presença ($D_e = 1$) ou não ($D_e = 0$) da aresta e na solução de ambos os ciclos Hamiltoniano;
- k é uma variável inteira que define a similaridade entre os dois ciclos, ou seja, k é o número de arestas que são visitadas por ambos os ciclos Hamiltonianos.

Já definidas
na página
anterior

2.1. Variáveis de Decisão

As variáveis x_e indicam se a aresta e do grafo faz parte ou não no primeiro ciclo Hamiltoniano da solução, e as variáveis y_e indicam se a aresta e faz parte do segundo ciclo Hamiltoniano, ambas são variáveis binárias.

2.2. Restrições

O modelo desenvolvido possui os seguintes grupos de restrições:

- (2.1): garante que todo vértice do grafo será visitado pelo primeiro ciclo Hamiltoniano da solução;
- (2.2): garante que todo vértice do grafo será visitado pelo segundo ciclo Hamiltoniano da solução;
- (2.3): visa a eliminação dos subciclos ilegais do grafo 1 da solução, impedindo que em qualquer subconjunto próprio de vértices existam mais do que $|S| - 1$ arestas, o que seria suficiente para formar um ciclo.
- (2.4): visa a eliminação dos subciclos ilegais do grafo 2 da solução, impedindo que em qualquer subconjunto próprio de vértices existam mais do que $|S| - 1$ arestas, o que seria suficiente para formar um ciclo.
- (2.5) Garante que o número de arestas visitadas pelos dois ciclos Hamiltonianos seja maior ou igual a k
- (2.6) Garante que uma aresta só será contada como visitada pelos dois ciclos Hamiltonianos, se ela fizer parte da solução de ambos os ciclos Hamiltonianos.
- (2.7) Garante que x_e^1 , x_e^2 e D_e sejam variáveis binárias, ~~recebendo o valor 1 apenas se fizerem parte da solução final.~~

3. Materiais e Métodos

3.1. Geração das instâncias

As instâncias foram geradas considerando as primeiras $N \in (100, 150, 200, 250)$ linhas do arquivo **mo824_atividade2_coords** provido junto ao enunciado. Para cada conjunto de N pontos, foram calculadas as distâncias c_e^1 e c_e^2 , assim como 3 valores de $k \in (0, N/2, N)$ sendo geradas, assim, 12 instâncias salvas em arquivos no formato **instance_<N>_<k>.pkl**. Cada instância foi lida interativamente e processada conforme o modelo e restrições.

3.2. Código e Dependências

O código para resolução do problema foi escrito na linguagem Python 3[1], executando na plataforma Jupyter Notebook[2] com a biblioteca gurobipy para utilizar o solver Gurobi[3]. Os códigos estão disponibilizados no formato `.ipynb` junto a esse documento, e também no formato PDF.

3.3. Máquina

Para execução do problema foi utilizado um computador com 16GB de memória RAM, processador Intel Core i5-8400 CPU 2.80GHz e sistema operacional Windows 11 de 64 bits.

4. Resultados

Cópia?

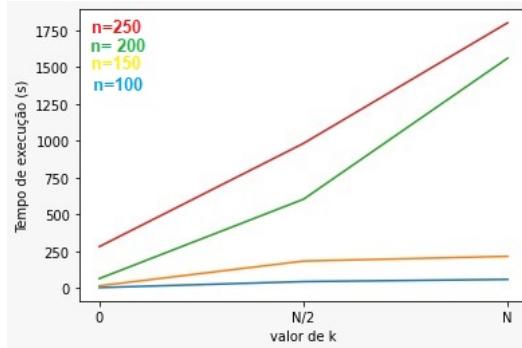
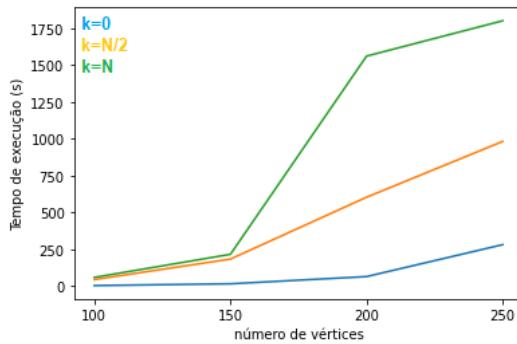
Os testes foram realizados para cinco valores diferentes para a quantidade de vértices ($|V|$), com os pesos das arestas no intervalo $[0, 1]$. Os resultados obtidos são mostrados na tabela a seguir.

Tabela 1: Experimentos com as instâncias geradas para o kTSP

Instância Nº de vértices	k	Custo	kTSP gap de optimalidade	Tempo (s)
100	0	1630	0.41%	4.26s
100	50	1758	0.23%	44.80s
100	100	2102	0.06%	59.74s
150	0	1966	0.04%	16.32s
150	75	2231	0.15%	183.99s
150	150	2748	0.31%	216.27s
200	0	2308	0.05%	65.18s
200	100	2642	0.07%	603.55s
200	200	3403	0.03%	1560.56s
250	0	2597	0.08%	281.85s
250	125	3033	0.04%	982.19s
250	250	3952	0.25%	1800.42s

Os valores descritos na tabela, assim como os caminhos obtidos, estão disponíveis junto ao documento no arquivo `result_p2.csv`. Os logs do gurobi também foram disponibilizados no arquivo `gurobi_logs.txt` como parte dos resultados.

os resultados não condizem com os valores esperados. Possivelmente algum problema de implementação

Figura 1: Tempos de execução para cada tamanho de instância com relação a k Figura 2: Tempos de execução para cada valor de k com relação a cada tamanho de instância

5. Análise dos Resultados

Nota-se ao observar a Figura 1 que o tempo de execução é proporcional ao valor de k . Isso é esperado, já que as soluções com valores maiores de k são mais restritivas, e portanto exigem maior poder computacional para encontrar uma solução válida. Com base nos valores apresentados no gráfico, o crescimento aparenta ser linear com relação a k , mas seriam necessárias mais execuções com outros valores de k para verificar se a tendência é linear, polinomial ou exponencial.

Com relação a Figura 2, deve-se levar em consideração que, na curva $k = N$, quando $N = 250$ o tempo limite de execução foi atingido, então o valor máximo de 1800s não representa, de fato, o tempo para encontrar a solução ótima. Com esse gráfico se observa um crescimento mais acentuado do tempo de execução em função no número de vértices, mas seriam necessários valores adicionais para descrever se

o crescimento é polinomial ou exponencial.

Fazendo análise dos logs é possível notar, especialmente para instâncias grandes como a de 250 vértices e k=250, que o solver leva um grande tempo até encontrar a primeira solução válida. Para trabalhos futuros, pode ser interessante prover ao solver uma solução inicial trivial - por exemplo, percorrendo os vértices na ordem em ambos os ciclos - fazendo com que o solver explore o espaço de busca com uma vantagem inicial.

Referências

- [1] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.
- [2] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, “Jupyter notebooks – a publishing format for reproducible computational workflows,” in *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (F. Loizides and B. Schmidt, eds.), pp. 87 – 90, IOS Press, 2016.
- [3] L. Gurobi Optimization, “Gurobi optimizer reference manual,” 2022.

Atividade 2

Problema do Caixeiro Viajante K-similar

Grupo 4

Luiz Fernando Bueno Rosa,
Renan Fernando Franco da Silva

Abril de 2022

MC859/MO824

Excelente relatório!

1 Introdução

Bônus + 1,0 pela parte experimental

De forma a resolver o Problema do Caixeiro Viajante k -similar ($kSTSP$), vamos utilizar uma modelagem em programação linear inteira e resolvê-la com o Solver Gurobi. Faremos uma análise do tempo de execução, do custo da solução e do GAP de otimalidade encontrados pelo solver para diversas instâncias do problema.

O $kSTSP$ consiste de encontrar, para cada caixeiro $i \in \{1, 2\}$, um ciclo Hamiltoniano $G_i = (V, E_i)$ mínimo dado um grafo $G = (V, E)$ e uma função de pesos $f^i : E \hookrightarrow \mathbb{R}^+$ de entrada em que no mínimo k arestas estão inclusas em ambos G_1 e G_2 .

1.1 Variáveis de Decisão

- $x_e^1 := 1$ caso a aresta $e \in E$ esteja inclusa em G_1 , isto é, $e \in E_1$, e 0 caso contrário.
- $x_e^2 := 1$ caso a aresta $e \in E$ esteja inclusa em G_2 , isto é, $e \in E_2$, e 0 caso contrário.
- $d_e := 1$ caso a aresta $e \in E$ esteja inclusa em G_i para todo $i \in \{0, 1\}$, e 0 caso contrário.

2 Modelo PLI

$$\text{minimizar} \quad \sum_{e \in E} c_e^1 x_e^1 + c_e^2 x_e^2 \quad (1)$$

$$\text{sujeito a} \quad \sum_{e \in \delta(v)} x_e^i = 2 \quad \forall v \in V, \quad \forall i \in \{1, 2\} \quad (2)$$

$$\sum_{e \in E(S)} x_e^i \leq |S| - 1 \quad \forall S \subset V, \quad \forall i \in \{1, 2\} \quad (3)$$

$$\sum_{e \in E} d_e \geq k \quad , \quad (4)$$

$$d_e \leq x_e^i \quad \forall e \in E, \quad \forall i \in \{1, 2\} \quad (5)$$

$$x_e^i, d_e \in \{0, 1\} \quad \forall e \in E \quad (6)$$

$$\forall i \in \{1, 2\} \quad (7)$$

Descrição do modelo?

3 Eliminação de Euler Subtours Dinamicamente

~~Veja que~~ ~~E~~ existe uma quantidade exponencial de restrições (3). Devido a isso, utilizamos a flag Lazy Constraints do Gurobi, para inserir as restrições violadas desse tipo dinamicamente, por meio de callbacks.

Note que em uma solução inteira, as restrições (2) garantem que o grafo G'_i , induzido pela ~~S~~ arestas $x_e^i > 0$ de cada caixeiro i , forme um conjunto de ciclos. Contudo, é necessário que cada G'_i seja composto por um único ciclo para uma solução ser viável.

Boa

explicação

Para checar a propriedade acima podemos procurar a menor componente conexa C_i^{\min} de cada G'_i . Se $|C_i^{\min}| = |V|$ para todo i , então a solução é viável. Já se $|C_i^{\min}| < |V|$, note que se somarmos as restrições de grau 2 para cada vértice $v \in C_i^{\min}$ obtemos $\sum_{e \in E(C_i^{\min})} x_e^i = |S|$, ou seja, temos que a restrição de subtour $\sum_{e \in E(C_i^{\min})} x_e^i \leq |S|-1$ está violada, e podemos acrescentá-la na formulação.

Por fim, note que o argumento acima de somar as restrições de grau também é aplicável em soluções fracionárias, ou seja, podemos utilizá-lo para realizar a separação de restrições violadas de subtour em soluções fracionárias.

4 Instâncias

O conjunto de instâncias foi gerado conforme as instruções, ou seja, todas as combinações tomado o número de vértices $|V| \in [100, 150, 200, 250]$ e a similaridade $k \in [0, \frac{|V|}{2}, |V|]$. O custo de cada aresta foi feito com as coordenadas dadas, o que nos deu um conjunto de 12 instâncias.

5 Resultados Computacionais

Os testes computacionais foram feitos em um Intel Core i5-8265U de 1.60GHz, com 8 GB de memória RAM e utilizando 8 Threads. O Solver PLI utilizado foi o Gurobi na versão 9.51. Além disso, a linguagem utilizada foi a C++17 com o Compilador GCC 9.4.0 no Sistema Operacional Ubuntu 20.04.01.

Como o Gurobi disponibiliza uma gama de parâmetros que podem ser alterados, nós testamos 6 configurações diferentes de tais parâmetros. Em todas as versões nós desabilitamos o PreSolve e colocamos um tempo limite de 30 minutos. Por padrão, utilizamos o método dual simplex para resolver a relaxação linear na raiz e nos nós da árvore de *branch-and-bound*, além de realizar a separação das inequações de subtour ~~euleriano~~ em toda solução inteira e em toda solução ótima da relaxação.

Os parâmetros testados foram Cuts, Heuristics e Method/NodeMethod. O parâmetro Cuts controla a agressividade da geração de planos de corte, onde o valor -1 representa o gerenciamento automático pelo Gurobi, o valor 0 desabilita a geração, e o valor 3 torna a geração o mais agressivo possível. Já o parâmetro Heuristics controla a porcentagem de tempo que o Gurobi vai gastar procurando soluções viáveis, sendo por padrão essa porcentagem igual a 0.05 (5% do tempo total). Por fim, os parâmetros Method e NodeMethod, controlam o método utilizado para resolver a relaxação na raiz e em cada nó da árvore de *branch-and-bound*, respectivamente, sendo que por padrão utilizamos o dual simplex.

Nas Tabelas 1 e 2, temos os resultados das seis versões para cada uma das 12 instâncias. A coluna **Obj** diz o custo da melhor solução inteira encontrada. Já a coluna **Bound** diz qual é o melhor *lower-bound* encontrado. Além disso temos a coluna **Gap** com o gap de otimalidade, e a coluna **Time** com tempo de execução total.

Como podemos ver, o melhor resultado foi alcançado com Cuts = 3, Heuristics = 0.5 e utilizando o Primal Simplex, onde resolvemos 10 das 12 instâncias de forma ótima, onde as duas instâncias restantes tiveram um gap de otimalidade de 0.26% e 4.4%. Consideramos tal resultado consistente, pois o *lower-bound* inicial em geral se encontra longe do valor da solução ótima, logo planos de cortes são primordiais para acelerar a convergência para a otimalidade. Além disso, também tínhamos notado que é razoavelmente difícil encontrarmos soluções inteiras, logo é sensato gastarmos boa parte do tempo com isso. O que não faz muito sentido é o melhor resultado ter sido alcançado com o método Primal Simplex, pois pelos nossos cálculos o problema primal tem mais restrições que variáveis, logo deveria ser melhor otimizarmos pelo dual.

Uma observação é que desabilitar os cortes tende a piorar o tempo de execução, contudo, curiosamente a melhor solução na instância N250K125 foi encontrada por tal versão. Além disso, a nossa ideia de separar as restrições de subtour ~~euleriano~~ em soluções fracionárias parece ajudar a melhorar a qualidade das soluções, visto que os resultados foram piores em tempo/gap quando passamos a separar apenas em soluções inteiras.

Em relação aos custos, podemos ver que conforme aumenta a similaridade, maiores os mesmos se tornam, o que é consiste, visto que quanto maior o k mais arestas custosas para alguns dos tours precisam ser usadas.

excellent

Por fim, algo interessante é que a dificuldade dos nossos algoritmos se encontram para $k = \frac{|V|}{2}$, o que faz bastante sentido, visto que assim há um misto de dependência e liberdade, que deve prejudicar bastante a qualidade da relaxação linear. E isso não acontece para os outros valores de k , visto que para $k = 0$, o problema consiste em resolver dois TSP individuais, e para $k = |V|$, o problema consiste em resolver um TSP onde cada aresta e tem custo $c_e^1 + c_e^2$.

Cuts = 3				Cuts = 3, Heuristics = 0.5				Cuts = 3, Heuristics = 0.5, Primal Simplex				
Obj	Bound	Gap (%)	Time (s)	Obj	Bound	Gap (%)	Time (s)	Obj	Bound	Gap (%)	Time (s)	
N100K0	1630	1630	0.00	5.4	1630	1630	0.00	3.9	1630	1630	0.00	5.5
N100K50	2102	2102	0.00	62.4	2102	2102	0.00	47.0	2102	2102	0.00	87.1
N100K100	3463	3463	0.00	6.7	3463	3463	0.00	5.5	3463	3463	0.00	22.8
N150K0	1966	1966	0.00	15.9	1966	1966	0.00	19.7	1966	1966	0.00	23.0
N150K75	2748	2748	0.00	215.5	2748	2748	0.00	146.8	2748	2748	0.00	138.0
N150K150	4780	4780	0.00	19.4	4780	4780	0.00	17.4	4780	4780	0.00	46.9
N200K0	2308	2308	0.00	82.7	2308	2308	0.00	45.0	2308	2308	0.00	56.8
N200K100	3458	3390	1.97	1801.2	3407	3398	0.26	1800.4	3405	3396	0.26	1800.1
N200K200	6003	6003	0.00	64.8	6003	6003	0.00	62.9	6003	6003	0.00	58.7
N250K0	2597	2597	0.00	376.1	2597	2597	0.00	170.5	2597	2597	0.00	152.4
N250K125	7525	3930	47.77	1800.5	4181	3935	5.88	1800.0	4118	3937	4.40	1800.0
N250K250	6999	6999	0.00	186.7	6999	6999	0.00	107.9	6999	6999	0.00	153.8

Table 1: Comparação entre as versões com o parâmetro Cuts = 3

Cuts = -1				Cuts = -1, Euler only in MIPSOL				Cuts = 0				
Obj	Bound	Gap (%)	Time (s)	Obj	Bound	Gap (%)	Time (s)	Obj	Bound	Gap (%)	Time (s)	
N100K0	1630	1630	0.00	2.1	1630	1630	0.00	3.5	1630	1630	0.00	5.8
N100K50	2102	2102	0.00	46.9	2102	2102	0.00	79.9	2102	2102	0.00	70.5
N100K100	3463	3463	0.00	2.4	3463	3463	0.00	3.2	3463	3463	0.00	8.2
N150K0	1966	1966	0.00	6.6	1966	1966	0.00	30.3	1966	1966	0.00	17.8
N150K75	2748	2748	0.00	123.9	2748	2748	0.00	498.3	2748	2748	0.00	356.0
N150K150	4780	4780	0.00	15.1	4780	4780	0.00	14.1	4780	4780	0.00	55.1
N200K0	2308	2308	0.00	43.5	2308	2308	0.00	102.3	2308	2308	0.00	33.7
N200K100	3535	3391	4.07	1800.5	7096	3389	52.24	1804.0	3420	3390	0.88	1800.2
N200K200	6003	6003	0.00	178.1	6003	6003	0.00	50.7	6003	6003	0.00	133.9
N250K0	2597	2597	0.00	305.5	2597	2597	0.00	393.9	2597	2590	0.27	1800.5
N250K125	7525	3928	47.80	1800.1	7525	3926	47.83	1800.0	4067	3936	3.22	1800.2
N250K250	6999	6999	0.00	131.9	6999	6999	0.00	100.0	6999	6999	0.00	605.1

Table 2: Comparação entre as versões com o parâmetro Cuts ≠ 3

MO824A/MC859A – Tópicos em Otimização Combinatória 1S2022
Atividade 2 - Relatório

Gabriela Arcifa De Resende, Gabriel Gomes De Siqueira, João Victor Aquino Batista

15 de Abril de 2022

O problema solucionado não foi o TSP

Introdução

O seguinte problema de programação linear foi modelado e solucionado utilizando o software Gurobi:

O problema do caixeiro viajante (travelling salesman problem – TSP) pode ser descrito da seguinte forma. Seja um grafo não-orientado completo $G(V, E)$, onde V é o conjunto dos vértices e E é o conjunto das arestas. Em cada aresta $e \in E$ há um custo $c_e : E \rightarrow \mathbb{R}^+$. O objetivo do problema consiste em encontrar um ciclo Hamiltoniano (que visita todos os vértices do grafo) de custo mínimo. Um modelo de programação linear inteira para o TSP é fornecido a seguir:

$$\text{MIN } \sum_{e \in E} c_e x_e \quad (1)$$

s. t.

$$\sum_{e \in \delta(i)} x_e = 2 \quad \forall i \in V \quad (2)$$

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

$$x_e \in \{0, 1\} \quad \forall e \in E \quad (4)$$

Onde cada parâmetro representa:

- não é parâmetro* →
- x_e é uma variável de decisão binária associada à presença ($x_e = 1$) ou não ($x_e = 0$) da aresta e na solução;
 - $\delta(i)$ é o conjunto de arestas que incidem no vértice i ;
 - $S \subset V$ é um subconjunto próprio de vértices;
 - $E(S)$ é o conjunto das arestas cujos dois vértices terminais estão em S .

Modelo matemático

A função objetivo (1) minimiza o custo da solução, composto pela soma dos custos de todas as arestas presentes na solução. O conjunto de restrições (2) diz que cada vértice deve possuir duas arestas incidentes na solução. Finalmente, o conjunto de restrições (3) visa a eliminação de subciclos ilegais, ou seja, a solução deve possuir um único ciclo que visita todos os vértices. O modo como as restrições (3) eliminam subciclos ilegais está em impedir que em qualquer subconjunto próprio de vértices existam mais do que $|S| - 1$ arestas, o que seria suficiente para formar um ciclo. Cabe notar que a quantidade de restrições (3) é exponencial em função da quantidade de vértices do grafo. Sendo assim, torna-se impraticável enumerar todas as restrições (3) para instâncias grandes. Uma forma de contornar essa questão está em utilizar o conceito de lazy constraints, onde as restrições são consideradas apenas sob demanda. Dessa forma, primeiro é passado ao resolvelor somente o conjunto de restrições (2). O resolvelor trata o modelo incompleto e ao encontrar uma solução inteira, essa solução

é passada ao usuário por meio de uma função de callback. O usuário verifica se a solução é composta por um ou mais ciclos. Se possuir somente um ciclo, a solução é válida e o algoritmo declara otimalidade. Caso contrário, o usuário insere uma ou mais restrições (3) violadas pela solução e o resolvedor re-otimiza considerando as novas restrições.

→ Descrição do problema sendo resolvido?

Variáveis de decisão:

- x_{1_e} é uma variável de decisão binária associada à presença ($x_{1_e} = 1$) ou não ($x_{1_e} = 0$) da aresta no primeiro ciclo;
- x_{2_e} é uma variável de decisão binária associada à presença ($x_{2_e} = 1$) ou não ($x_{2_e} = 0$) da aresta no segundo ciclo;

Parâmetros:

- $\delta(i)$ é o conjunto de arestas que incidem no vértice i ;
- $S \subset V$ é um subconjunto próprio de vértices;
- $E(S)$ é o conjunto das arestas cujos dois vértices terminais estão em S .
- c_e custo de uma aresta $e = (i, j)$

Objetivo:

$$MIN \sum_{e \in E} c_1 x_{1_e} + \sum_{e \in E} c_2 x_{2_e}$$

Restrições

$$\sum_{e \in \delta(i)} x_{1_e} = 2 \quad \forall i \in V$$

$$\sum_{e \in \delta(i)} x_{2_e} = 2 \quad \forall i \in V$$

$$\sum_{e \in E(S)} x_{1_e} \leq |S| - 1 \quad \forall S \subset V$$

$$\sum_{e \in E(S)} x_{2_e} \leq |S| - 1 \quad \forall S \subset V$$

$$\sum_{e \in E, x_{1_e} = x_{2_e}} x_{1_e} \leq k$$

Domínios?

Isso não é PLI

Esta última restrição se trata da similaridade de ambos os ciclos, ou seja, a quantidade k de arestas $e \in E$ que precisam ser iguais nos dois ciclos.

Seja um grafo não-orientado completo $G(V, E)$, onde V é o conjunto dos vértices e E é o conjunto das arestas. Em cada aresta $e \in E$ há dois custos c_e^1, c_e^2 .

A partir das coordenadas gerou-se os custos c_e^1 e c_e^2 de uma aresta $e = (i, j)$ calculando o teto da distância Euclidiana dos pares de coordenadas, conforme as equações abaixo:

$$c_e^1 = \left[\sqrt{(x_i^1 - x_j^1)^2 + (y_i^1 - y_j^1)^2} \right]$$

$$c_e^2 = \left[\sqrt{(x_i^2 - x_j^2)^2 + (y_i^2 - y_j^2)^2} \right]$$

Domínios:

$$|V| = \{100, 150, 200, 250\}$$

$$k = \{0, |V|/2, |V|\}$$

$$c_e^1, c_e^2 : E \rightarrow \mathbb{R}^+$$

Configuração dos experimentos?
Ambiente computacional?

Experimentos

Isso faz parte da descrição do problema

Buscou-se encontrar dois ciclos Hamiltonianos com custo total mínimo, tal que pelo menos k arestas do grafo fossem visitadas por ambos os ciclos. O parâmetro k define a similaridade entre os ciclos, de modo que $k = |V|$ implica que uma solução factível corresponda a dois ciclos Hamiltonianos contendo as mesmas $|V|$ arestas, enquanto $k = 0$ resulta em uma solução factível ser qualquer par de ciclos Hamiltonianos.

Gerou-se 12 instâncias, combinando quatro conjuntos de vértices ($|V| = \{100, 150, 200, 250\}$) e três valores de similaridade ($k = \{0, |V|/2, |V|\}$). *ítálico*

O problema foi resolvido utilizando o solver Gurobi na linguagem Python 3.

Segue a tabela com as instâncias e os resultados obtidos:

Instância	Parâmetros	Tempo de execução	Custo da solução
1	$n = 250, k = 0$	9.84s	2244
2	$n = 250, k = 125$	11.93s	2291
3	$n = 250, k = 250$	10.24s	2441
4	$n = 200, k = 0$	1.96s	2030
5	$n = 200, k = 100$	4.41s	2325
6	$n = 200, k = 200$	5.43s	2453
7	$n = 150, k = 0$	1.06s	1751
8	$n = 150, k = 75$	3.58s	1754
9	$n = 150, k = 150$	3.94s	2206
10	$n = 100, k = 0$	0.38s	1430
11	$n = 100, k = 50$	0.95s	1704
12	$n = 100, k = 100$	0.83s	1706

Os tempos computacionais e os custos sugerem algum problema de implementação

Análise

A partir do resultado obtido, podemos observar que ao aumentarmos o valor de k , o custo e o tempo de execução tendem a aumentar também (na maioria das vezes). Tal fato ocorre pois ao exigirmos um grau de similaridade maior, maior será o processamento necessário para encontrar uma situação que satisfaça o problema, já que seu grau de especificidade é maior, como visto entre as instâncias 4, 5 e 6. Também pode ocorrer a situação em que tais ciclos podem ser encontrados antes e em menos tempo, mesmo com o aumento de k , como podemos ver entre as instâncias 11 e 12. Como o experimento não possui um comportamento constante, pode-se dizer que um tempo de solução e custo mais fidedigno pode ser obtido através de uma análise de uma bateria de testes com os mesmos parâmetros.

→ não necessariamente

Atividade 2

Grupo 7

Lucas Costa De Oliveira RA: 182410, Matheus Cesar Nunes RA: 203373

15 de Abril de 2022

MO824A/MC859A – Tópicos em Otimização Combinatória

Descrição do Problema ?

Modelo Matemático

itônico

Como objetivo da Atividade 2 temos como finalidade resolver o problema de kSTSP para diferentes valores de k . Para a formulação desse problema utilizamos como base o problema TSP pois são problemas semelhantes.

Para resolver esse problema fizemos uma formulação de PLI em que criamos as seguintes variáveis binárias:

x_e^1 = indica se a aresta e foi utilizada no ~~caminho~~ ^{ciclo} 1

x_e^2 = indica se a aresta e foi utilizada no ~~caminho~~ ^{ciclo} 2

D_e = indica que a aresta e foi utilizada nos 2 ~~caminhos~~ ^{ciclos}

E utilizamos as seguintes constantes:

K = número de arestas presentes nos 2 caminhos

C_e^1 = custo da aresta e no caminho 1

C_e^2 = custo da aresta e no caminho 2

V = conjunto de vértices do grafo

E = conjunto de arestas do grafo

$S \subset V$ = é um subconjunto próprio de vértices

$\delta(i)$ = conjunto de arestas que incide no vértice $i \in V$

$E(S)$ = conjunto de arestas cujo os 2 vértices terminais estão em S

Como objetivo queremos minimizar a função dada por:

$$F(x_e^1, x_e^2) = \sum_e x_e^1 C_e^1 + \sum_e x_e^2 C_e^2$$

Para resolver esse problema de PLI estabelecemos as seguintes restrições às nossas variáveis:

$$(1) \sum_{e \in \delta(i)} x_e^1 = 2 \quad \forall i \in V$$

$$(2) \sum_{e \in \delta(i)} x_e^2 = 2 \quad \forall i \in V$$

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

$$(4) \sum_{e \in E(S)} x_e^2 \leq |S| - 1 \quad \forall S \subset V$$

$$(5) x_e^1 + x_e^2 \geq 2D_e \quad \forall e \in E$$

$$(6) \sum_{e \in E} D_e \geq K$$

Domínio das variáveis?

Que podem ser traduzidas como:

- (1) A soma da quantidade de arestas incidentes no vértice i deve ser igual a 2 para garantir que o vértice é visitado apenas 1 vez no caminho 1, uma aresta entra e uma aresta sai do vértice.
- (2) A soma da quantidade de arestas incidentes no vértice i deve ser igual a 2 para garantir que o vértice é visitado apenas 1 vez no caminho 2, uma aresta entra e uma aresta sai do vértice.
- (3) A soma da quantidade de arestas com os 2 vértices terminais em S , ou seja, no ciclo 1, deve ser igual ou menor ao número de vértices presentes no ciclo 1 menos 1 para garantir que eliminaremos subciclos ilegais ao limitar o número de arestas presentes num subconjunto de vértices. para AS CV
idem
Se De tentão e
- (4) A soma da quantidade de arestas com os 2 vértices terminais em S , ou seja, no ciclo 2, deve ser igual ou menor ao número de vértices presentes no ciclo 2 menos 1 para garantir que eliminaremos subciclos ilegais ao limitar o número de arestas presentes num subconjunto de vértices.
- (5) A soma das variáveis binárias representando se as arestas estão no ciclo deve ser maior ou igual a 2 vezes a variável que representa se a aresta se encontra em ambos os ciclos, ou seja, se a aresta é compartilhada ela deve estar em ambos os ciclos.
- (6) A soma das variáveis binárias que representam se uma aresta está nos dois ciclos deve ser maior ou igual a K , para garantir que pelo menos K arestas sejam compartilhadas.

Resultados

É importante ressaltar que na implementação para a resolução do problema kSTSP nos baseamos no código já apresentado que resolve o problema TSP. As principais modificações feitas foram a duplicação de caminhos, pois queremos encontrar um caminho para cada viajante, e a criação da restrição de similaridade K entre os caminhos. Além disso, as restrições dadas por (3) e (4) não foram implementadas diretamente no código pois seu número seria exponencial se comparado ao número de vértices. Para implementar essas restrições utilizamos o conceito de *lazy constraints*, ou seja, criamos restrições sob demanda. No código temos uma função callback que é responsável por verificar se a solução encontrada é válida, caso não seja observamos qual restrição, dentre as restrições (3) e (4), que impediria que o programa encontre essa solução inválida e executamos novamente, desse modo inserimos apenas as restrições necessárias dentre todas presentes em (3) e (4).

A solução foi desenvolvida na linguagem de programação Java, e como argumento recebe o diretório para o arquivo de coordenadas dos vértices para os dois viajantes, conforme disponibilizado no enunciado da atividade. A respeito da configuração da máquina que executou as instâncias, foi executado pelo processador Intel® Core™ i7-7600U CPU @ 2.80GHz × 4, o qual o solver - *Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (linux64)* utilizou da seguinte forma: “*Thread count: 2 physical cores, 4 logical processors, using up to 4 threads*”.

Como as instâncias foram geradas?

Instância #	Vértices ($ v $) e Similaridade (k)	Custo da Solução	Chamadas de callback	Gap de Optimalidade	Tempo de Execução (segundos)
1	100 e 0	1630	2984	0.0000%	4.18
2	100 e 50	2102	25214	0.0000%	71.10
3	100 e 100	3463	6361	0.0000%	55.93
4	150 e 0	1966	9143	0.0000%	29.80
5	150 e 75	2748	174798	0.0000%	1305.95
6	150 e 150	4780	4124	0.0000%	28.56
7	200 e 0	2308	34453	0.0000%	154.72
8	200 e 100	4769	110349	28.9579%	1800.01
9	200 e 200	6003	26009	0.0000%	178.36
10	250 e 0	2597	34100	0.0000%	193.98
11	250 e 125	—	166519	— %	1800.05
12	250 e 250	6999	103155	0.0000%	1472.06

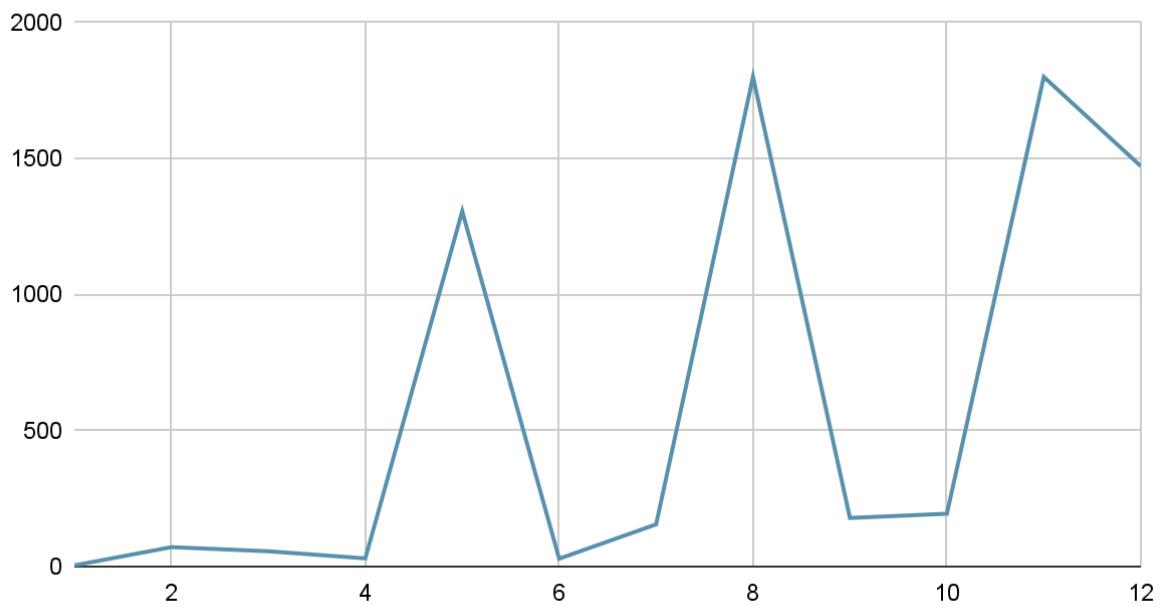
Análise

Muito boa análise!

Como pode ser observado na tabela de resultados acima obtivemos um padrão em que os tempos de execução mais demorados foram os para qual a similaridade é dada por $|v|/2$. Esse tipo de comportamento era esperado pois ao conhecer o problema TSP e entender as modificações necessárias de restrição para o kSTSP podemos presumir que uma solução para $k = 0$ é apenas achar 2 ciclos hamiltonianos, ou seja, resolver 2 instâncias de TSP, pois não temos restrições a mais. Para o caso de $k = |v|$ podemos presumir que devemos achar 1 ciclo hamiltoniano com o menor custo para os 2 viajantes, ou seja, resolvemos 1 instância de TSP com 2 valores de custo em cada aresta. ~~essa modificação não deveria ser tão custosa.~~ Porém para o caso de $k = |v|/2$ (ou quaisquer outros valores em que $k \neq 0$ e $k \neq |v|$) é que temos a real dificuldade dessa modificação pois impõe uma restrição muito forte em nosso problema.

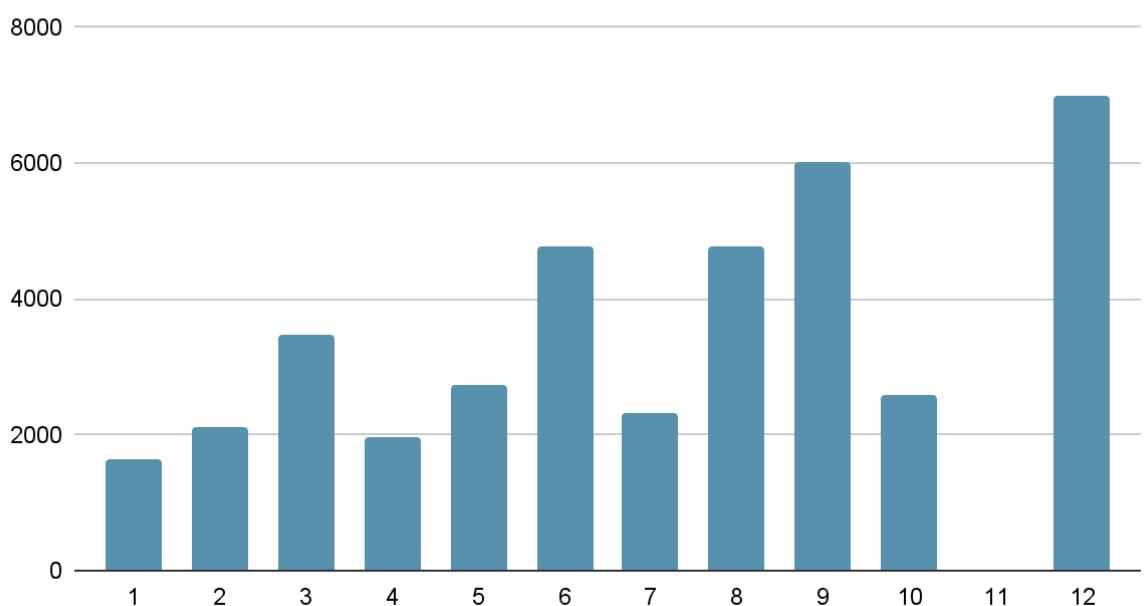
Estabelecemos um tempo limite de execução de 30 minutos em nosso código e apenas 2 instâncias ultrapassaram esse limite, a 8 e a 11, ambas com $k = |v|/2$. Com o gráfico abaixo representando o tempo de execução fica mais claro observar um padrão em como picos são formados pois as instâncias foram geradas na ordem $k = 0, k = |v|/2$ e $k = |v|$. Normalmente o tempo de execução de $k = 0$ e $k = |v|$ deveria ser parecidos por se tratarem de problemas igualmente fáceis de serem resolvidos, mas tivemos 2 resultados diferentes com as instâncias 3 e 12 em que o tempo de execução para $k = |v|$ foi muito maior. Uma hipótese para esse tempo de execução maior pode ser o fato de que nessa iterações foi preciso um número muito maior de callbacks para adicionar as *lazy constraints*, como mais restrições tiveram que ser adicionadas, mais tentativas de solução do TSP foram executadas, levando a um maior tempo de execução.

Tempo de Execução (segundos)



Olhando para os custos obtidos das soluções também obtivemos um comportamento esperado, pois em $k = 0$ são os 2 ciclos ótimos de TSP para cada viajante, resultando no menor valor. Para $k = |V|$ é o ciclo ótimo somando os custos dos 2 viajantes, com as arestas estarem atreladas não é possível escolher a melhor aresta para cada caminho, resultando no maior valor. Para $k = |V|/2$ temos um meio termo da solução, em que metade das arestas são as melhores em par e metade são as melhores específicas para cada viajante, por isso temos um custo entre os 2 anteriores. Olhando pelo gráfico é possível ver o crescimento do custo ao longo do aumento de $|V|$ e mudança de k .

Custo da Solução



MO824 - Tópicos em Otimização Combinatória

TSP k-semelhantes

Equipe 8

Lucas de Oliveira Silva, Luiz Gustavo S. Aguiar

RAs: 220715, 240499

15 de Abril de 2022

Excelente
relatório!

Descrição do Problema

+0,5 ponto para parte experimental

O kSTSP pode ser descrito da seguinte forma: seja um grafo não-orientado completo $G(V, E)$, onde V é o conjunto dos vértices e E é o conjunto das arestas, onde para cada aresta $e \in E$ há dois custos $c_e^1, c_e^2 : E \Rightarrow \mathbb{R}^+$, o objetivo do problema consiste em encontrar dois ciclos Hamiltonianos com custo total mínimo, de forma que pelo menos k arestas do grafo sejam visitadas por ambos os ciclos. O parâmetro k define a similaridade entre os ciclos, de modo que $k = |V|$ implica em uma solução onde os dois ciclos contém exatamente as mesmas $|V|$ arestas, enquanto $k = 0$ implica em uma solução onde os ciclos podem ser completamente distintos. O custo das arestas será sempre um inteiro positivo, pois será o teto da distância euclidiana.

Formulação

A seguir iremos descrever todas as componentes da formulação do programa linear.

Variáveis

- x_e^1 : Variável binária que diz se a aresta e foi utilizada no ciclo 1.
- x_e^2 : Variável binária que diz se a aresta e foi utilizada no ciclo 2.
- D_e : Variável binária que diz se a aresta e foi utilizada em ambos os ciclos.

Constantes

- c_e^1 : Custo da aresta e quando utilizada no ciclo 1, e $c_e^1 \in \mathbb{Z}^+$.
- c_e^2 : Custo da aresta e quando utilizada no ciclo 2, e $c_e^2 \in \mathbb{Z}^+$.
- k : Quantidade mínima de arestas compartilhadas por ambos os ciclos. $0 \leq k \leq |V|$ e $k \in \mathbb{Z}$.
- $\delta(v)$: conjunto de arestas incidentes no vértice v .

Função objetivo

$$MIN \quad \sum_{e \in E} x_e^1 \cdot c_e^1 + x_e^2 \cdot c_e^2$$

Na função objetivo queremos minimizar o custo total dos ciclos, que é dado pela soma dos custos das arestas utilizadas em cada um dos ciclos.

Restrições

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

$$\sum_{e \in E} D_e \geq k \quad (2)$$

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

$$x_e^1 \geq D_e, x_e^2 \geq D_e, \forall e \in E \quad (4)$$

$$x_e^1, x_e^2, D_e \in \{0, 1\}, \forall e \in E \quad (5)$$

As restrições são as seguintes:

1. O número de arestas utilizadas pelos caixeiros em cada vértice deve ser igual a dois caso seja menor não teremos um ciclo e caso seja maior teremos mais de um ciclo.
2. O número de arestas compartilhadas deve ser pelo menos k arestas.
3. Não haverá subciclos ilegais relativos aos ciclo, isto é, os ciclos passam por todos os vértices utilizando somente a quantidade de arestas necessárias para formá-los.
4. Se uma aresta é duplicada então ela obrigatoriamente é utilizada no ciclo caso contrário ela estaria na rota mas seu custo não seria levado em consideração.
5. x_e^1, x_e^2, D_e são binárias.

Resultados

Forma de geração das instâncias?

Implementamos o modelo descrito com o solver Gurobi na versão 9.5.1 utilizando a linguagem C++ compilador g++ 11.2.0 com as opções:

```
model.set(GRB_IntParam_MIPFocus, GRB_MIPFOCUS_FEASIBILITY);
```

```
model.set(GRB_IntParam_Cuts, GRB_CUTS.Aggressive);
```

```
model.set(GRB_IntParam_Presolve, GRB_PRESOLVE.Aggressive);
```

numa máquina com o processador Intel Core i5-7300HQ com 2.50 GHz, 8GB de memória RAM e sistema operacional Pop!_OS 21.10 e kernel 5.16.11.

Análises

Exelente análise!

Quando $k = 0$ todos os custos são menores. Isso pode ser justificado devido ao fato dos ciclos não terem obrigatoriedade de compartilhar arestas, sendo cada um deles o melhor ciclo possível. Quanto maior o valor de k , maior o custo, pois os ciclos são obrigados a compartilhar arestas que podem tornar mais custosos.

O tempo de execução sempre é mais rápido quando $k = 0$, pois não há necessidade de verificar a restrição de similaridade. Quando $k = |V|$ o problema se reduz ao TSP normal onde o custo agora é a soma das duas funções de custo originais.

Os piores tempos foram com a similaridade igual a metade do número de nós. Isso pode ser justificado pelo comportamento binomial apresentado pelo parâmetro k , ou seja, $\binom{|V|}{k}$ é máximo quando $k = |V|/2$. Portanto há muito mais opções sobre quais arestas escolher duplicar. Esse fato aumenta a profundidade da árvore de busca e torna a busca por soluções mais demorada. Se tivéssemos explorado as propriedades combinatórias da estrutura do problema ao invés de usar PLI puro poderíamos ter melhorado o resultado, pois sabemos que o valor ótimo para $k = |V|$ é um limitante trivial.

Vértices	Similaridade	0	50	100	0	100	0	150	0	200	0	200	0	250	0	250	Execução
OPT	1630,0	2102,0	3463,0	1966,0	2748,0	4780,0	2308,0	119,3	1800,0	152,6	191,0	1800,0	-	6003,0	2597,0	-	6999,0
tempo total (s)	5,4	64,2	8,1	14,2	241,0	29,9	119,3	1800,0	152,6	191,0	147,0	191,0	-	688,3	-	1625,0	1 ^a
tempo até 1º sol (s)	5,0	60,0	7,0	13,0	244,5	27,0	108,0	1141,0	147,0	191,0	-	3920,7	6999,0	-	-	-	-
LB	1630,0	2102,0	3463,0	1966,0	2748,0	4780,0	2308,0	3390,0	6003,0	2597,0	3458,0	6003,0	3458,0	6003,0	2597,0	-	6999,0
UB	1630,0	2102,0	3463,0	1966,0	2748,0	4780,0	2308,0	3390,0	6003,0	2597,0	3458,0	6003,0	3458,0	6003,0	2597,0	-	6999,0
OPT	1630,0	2102,0	3463,0	1966,0	2748,0	4780,0	2308,0	3390,0	6003,0	2597,0	3458,0	6003,0	3458,0	6003,0	2597,0	-	6999,0
tempo total (s)	5,7	63,7	8,1	14,4	241,0	30,4	116,0	1800,0	152,3	194,4	1800,0	-	685,1	-	-	-	2 ^a
tempo até 1º sol (s)	5,0	60,0	7,0	13,0	244,3	27,0	106,0	1136,0	147,0	194,0	-	3920,7	6999,0	-	-	-	1622,0
LB	1630,0	2102,0	3463,0	1966,0	2748,0	4780,0	2308,0	3390,0	6003,0	2597,0	3458,0	6003,0	3458,0	6003,0	2597,0	-	6999,0
UB	1630,0	2102,0	3463,0	1966,0	2748,0	4780,0	2308,0	3390,0	6003,0	2597,0	3458,0	6003,0	3458,0	6003,0	2597,0	-	6999,0
OPT	1630,0	2102,0	3463,0	1966,0	2748,0	4780,0	2308,0	3390,0	6003,0	2597,0	3458,0	6003,0	3458,0	6003,0	2597,0	-	6999,0
tempo total (s)	5,4	64,0	8,2	14,0	241,0	29,9	115,5	1800,0	152,2	191,0	1800,0	-	6999,0	-	-	-	6999,0
tempo até 1º sol (s)	5,0	60,0	7,0	13,0	244,4	27,0	105,0	1133,0	147,0	190,0	-	3920,7	6999,0	-	-	-	1690,0
LB	1630,0	2102,0	3463,0	1966,0	2748,0	4780,0	2308,0	3390,0	6003,0	2597,0	3458,0	6003,0	3458,0	6003,0	2597,0	-	6999,0
UB	1630,0	2102,0	3463,0	1966,0	2748,0	4780,0	2308,0	3390,0	6003,0	2597,0	3458,0	6003,0	3458,0	6003,0	2597,0	-	6999,0

Não ganha bônus por fazer sozinho

Atividade 2

Sandro Henrique Uliana Catabriga - RA 219597

April 15, 2022

Descrição do problema?

1 Formulação do Problema

Dado um grafo não-orientado completo $G(V, E)$, onde V é o conjunto dos vértices e E é o conjunto das arestas, sendo que cada aresta $e \in E$ tem dois custos $c_e^1, c_e^2 : E \rightarrow \mathbb{R}^+$, temos:

- x_e^1 : variável de decisão binária que indica se uma aresta e foi utilizada *exclusivamente* no ciclo 1;
- x_e^2 : variável de decisão binária que indica se uma aresta e foi utilizada *exclusivamente* no ciclo 2;
- x_e^{12} : variável de decisão binária que indica se uma aresta e foi utilizada em ambos os ciclos 1 e 2;
- $\delta(i)$: conjunto de arestas que incidem no vértice i ;
- $S \subset V$: subconjunto próprio de vértices;
- $E(S)$: conjunto das arestas cujos dois vértices terminais estão em S .

$$\begin{aligned}
 (\text{STSP}) \quad & \text{minimize} \sum_{e \in E} c_e^1(x_e^1 + x_e^{12}) + c_e^2(x_e^2 + x_e^{12}) \\
 \text{sujeito a} \quad & \sum_{e \in \delta(i)} (x_e^1 + x_e^{12}) = 2 \quad \forall i \in V \quad (1) \\
 & \sum_{e \in \delta(i)} (x_e^2 + x_e^{12}) = 2 \quad \forall i \in V \quad (2) \\
 & \sum_{e \in E(S)} (x_e^1 + x_e^{12}) \leq |S| - 1 \quad \forall S \subset V \quad (3) \\
 & \sum_{e \in E(S)} (x_e^2 + x_e^{12}) \leq |S| - 1 \quad \forall S \subset V \quad (4) \\
 & \sum_{e \in E} x_e^{12} \geq k \quad (5) \\
 & x_e^1, x_e^2, x_e^{12} \in \{0, 1\} \quad \forall e \in E \quad (6)
 \end{aligned}$$

Gastei!

Na função objetivo queremos minimizar o custo total dos ciclos, que é dado pela soma dos custos das arestas utilizadas no ciclo 1, identificadas por $x_e^1 = 1$ ou $x_e^{12} = 1$, mais os custos das arestas utilizadas no ciclo 2, identificadas por $x_e^2 = 1$ ou $x_e^{12} = 1$. Para qualquer aresta e utilizada no ciclo 1, $x_e^1 \neq x_e^{12}$, pois se forem iguais o custo da solução aumentará. O mesmo vale para as arestas utilizadas no ciclo 2.

A restrição (1) diz que cada vértice deve possuir duas arestas incidentes do ciclo 1, já considerando as arestas que fazem parte ~~do~~ ⁵ dos dois ciclos. A restrição (2) diz que cada vértice deve possuir duas arestas incidentes do ciclo 2, também considerando as arestas que fazem parte dos dois ciclos. A restrição (3) garante que não haverão subciclos ilegais relativos ao ~~ciclo~~ 1, isto é, o ciclo 1 passa por todos os vértices utilizando somente a quantidade de arestas necessárias para formar o ciclo. A restrição (4) garante a mesma coisa para o ciclo 2. A restrição (5) faz com que o número de arestas utilizadas nos dois ciclos (similaridade) seja pelo menos k . Por fim, a restrição (6) é relativa ao domínio ~~inteiro~~ binário das variáveis.

De forma semelhante ao TSP, no STSP as restrições (3) e (4) são exponenciais no número de vértices. Para resolver isso, utilizamos *lazy constraints*.

2 Experimentos e Análise

Para os experimentos foi utilizada uma máquina com chip Apple M1 e 8GB de memória RAM. Foi utilizada a API em C++ do software Gurobi, disponibilizando os 8 núcleos físicos da máquina para execução. O tempo de execução das instâncias foi limitado a 1800 segundos.

As instâncias foram geradas a partir do arquivo de coordenadas fornecido pelo professor, e possuem o formato a seguir, onde n é o número de vértices e k o fator de similaridade.

$< n >$

$< k >$

$< \text{coordenada } x_1^1 > \quad < \text{coordenada } y_1^1 > \quad < \text{coordenada } x_1^2 > \quad < \text{coordenada } y_1^2 >$

$< \text{coordenada } x_2^1 > \quad < \text{coordenada } y_2^1 > \quad < \text{coordenada } x_2^2 > \quad < \text{coordenada } y_2^2 >$

\dots

$< \text{coordenada } x_n^1 > \quad < \text{coordenada } y_n^1 > \quad < \text{coordenada } x_n^2 > \quad < \text{coordenada } y_n^2 >$

A Tabela 1 apresenta um resumo das instâncias geradas. A Tabela 2 apresenta algumas informações e resultados das execuções. Com exceção da instância 12, todas terminaram dentro do limite de tempo estabelecido. É possível observar que o custo das instâncias cresce de acordo com o valor de k . Quando $k = 0$ temos os dois ciclos com o menor custo possível, e se k aumenta é possível que o programa linear seja “forçado” a incluir uma aresta mais pesada para que a solução respeite a restrição (5).

Também é possível notar que quando $k = 0$ o tempo de execução das instâncias é menor. Isso pode ser devido ao fato de que, quando $k = 0$, podemos “descartar” a restrição (5). Com exceção das instâncias 4, 5 e 6, para todas as outras o tempo de execução cresceu quando k aumentava.

$$K = \frac{|V|}{2} \quad \text{e} \quad K = M ?$$

Discussão?

Instância	Nº vértices	K
1	100	0
2	100	50
3	100	100
4	150	0
5	150	75
6	150	150
7	200	0
8	200	100
9	200	200
10	250	0
11	250	125
12	250	250

Tabela 1: Tabela de instâncias.

Instância	Custo	GAP (%)	Tempo (s)
1	1630	0	2.1
2	1758	0	10.2
3	2102	0	12
4	1966	0	10.3
5	2231	0	157.5
6	2748	0	55
7	2308	0	47
8	2642	0	443.8
9	3403	0	1566.9
10	2597	0	193
11	3033	0	1264.8
12	4029	2.9	1800

Tabela 2: Tabela de resultados.

4) Tem alguma coisa errada
na implementação

Atividade 2 - MO824

Ieremies Romero, Gian Franco Condori Luna, Gabriel Branco

April 16, 2022

Descrição do problema?

Modelo

Usamos o modelo já conhecido para o TSP convencional como base

$$\begin{aligned} \min & \sum c_e x_e \\ & \sum_{e \in \delta(v)} x_e = 2 \quad v \in V \\ & \sum_{e \in \delta(S)} x_e \leq |S| - 1 \quad \forall S \subset V \end{aligned}$$

No nosso caso, temos que resolver dois TSP's mas que as soluções possuam k arestas em comum.

No nosso modelo x_e^1 indica que usamos a aresta e para o tuor 1 e respectivo para o tuor 2 e D_e indica se a aresta está duplicada.

Nossa função objetivo pode ser a soma dos custos dos dois tuors, ou seja

$$\min \sum_{e \in E} \sum_{i \in \{1,2\}} c_e^i x_e^i.$$

Repetimos as restrições do TSP para cada um dos tuors.

$$\sum_{e \in \delta(v)} x_e^i = 2 \quad v \in V \quad \forall i \in \{1, 2\}$$

$$\sum_{e \in \delta(S)} x_e^i \leq |S| - 1 \quad \forall S \subset V \quad \forall i \in \{1, 2\}$$

É importante notar que a segunda equação dá origem a quantidade exponencial de restrições de eliminação de subtútor. No nosso código, podemos

circundar esse problema adicionando as restrições conforme se faz necessário. Assim, quando o modelo termina com um certo conjunto de restrições, podemos conferir, por meio de uma busca ~~em~~^{em} profundidade, se é uma solução viável considerando a restrição de subtúor. Caso não seja, adicionamos as restrições de subtúor que evitam essa solução. Fazemos isso até encontrarmos uma solução viável.

Por fim, adicionamos as restrições que exigem a quantidade de arestas compartilhadas.

$$x_e^i \geq D_e \quad \forall e \in E \quad \forall i \in \{1, 2\}$$

$$\sum_{e \in E} D_e \geq k$$

Assim, nosso modelo final é

$$\begin{aligned} \min \sum_{e \in E} \sum_{i \in \{1, 2\}} c_e^i x_e^i \\ \sum_{e \in \delta(v)} x_e^i = 2 \quad \forall v \in V \quad \forall i \in \{1, 2\} \\ \sum_{e \in \delta(S)} x_e^i \leq |S| - 1 \quad \forall S \subset V \quad \forall i \in \{1, 2\} \\ x_e^i \geq D_e \quad \forall e \in E \quad \forall i \in \{1, 2\} \end{aligned}$$

$$\sum_{e \in E} D_e \geq k$$

Variáveis de decisão?

Parâmetros?

Domínios?

Explicação do modelo incompleta

Geração de instâncias

Para testar nosso modelo, utilizamos o arquivo de coordenadas disponibilizado pelo professor para calcular nossos custos de arestas. Assim, para instâncias de 100 cidades, utilizamos as 100 primeiras linhas do arquivo.

Durante os testes, modificamos a quantidade de cidades (100, 150, 200 e 250) e o valor de k (zero, metade da quantidade de cidades e a quantidade de cidades).

Resultados

Realizamos os testes em um computador equipado de um processador i5 de oitava geração, com 4 cores e 8 threads a 1.6ghz (max boost 3.2) e 8gb de ram, sem swap, com sistema operacional Linux 64bits.

Table 1: Métricas do modelo para as instâncias citadas no formato custo, gap e tempo de execução.

	$k = 0$	$k = \frac{v}{2}$	$k = v$
$v = 100$	(1630, 0%, 12.46)	(2102, 0%, 51.23)	(3463, 0%, 18.29)
$v = 150$	(1966, 0%, 79.01)	(2748, 0%, 565.42)	(4780, 0%, 36.89)
$v = 200$	(2308, 0%, 208.72)	(17763, 80.9%, 2048)	(6003, 0%, 112.67)
$v = 250$	(2916, 11.1%, 1113)	(-, -, -)	(6999, 0%, 596.74)

É importante ressaltar que a instância com $v = 250$ e $k = 0$ foi finalizada pelo sistema operacional por falta de memória ram. O resultado reportado aqui é da última atualização fornecida pelo Gurobi. Além disso, a instância de $v = 250$ e $k = 125$ o modelo não foi capaz de encontrar uma solução viável em menos de 30 minutos.

Análise

Apresentou os resultados, mas faltou análise e discussão

Essa sentença apresenta erros conceituais importantes!

Observando as métricas obtidas, vemos os piores tempos de execução são encontrados quando $k = \frac{v}{2}$, com as instâncias maiores demorando consideravelmente a terminar, muitas vezes sem achar uma solução ótima. Além disso, apesar não diretamente mostrado na tabela, observamos que o tempo gasto adicionando restrições de forma "lazy" é consideravelmente pequeno em comparação com o tempo total.

O tempo de execução de um problema TSP é um fator polinomial de ordem $O(n!)$. Nosso modelo usa um tempo polinomial para encontrar a solução ótima.

Fontes

https://colab.research.google.com/github/Gurobi/modeling-examples/blob/master/traveling_salesman/tsp_gcl.ipynb

Excelente relatório!

Submetido para disciplina MO824

Atividade 2 - Programação Linear com Lazy Constraints

Felipe Pereira RA 263808

José Antonio Mauad Leis RA 219061

Lucas Guesser Targino da Silva RA 203534

1. Enunciado do Problema

Sejam:

1. $G = \langle V, E \rangle$: um grafo não-orientado completo:
- V : conjunto de vértices;
 - E : conjunto das arestas;
2. $c^1 c^2 : E \rightarrow \mathbb{R}_+$ duas funções custo nos vértices;
- dada uma aresta e , escrevemos $c^1(e) = c_e^1$ e $c^2(e) = c_e^2$;
3. k : parâmetro de similaridade de ciclos;

Objetivo: encontrar dois ciclos Hamiltonianos com custo total mínimo, tal que pelo menos k arestas do grafo sejam visitadas por ambos os ciclos.

2. Modelo Matemático

2.1. Variáveis de Decisão

- X_e^1 : presença da aresta e no primeiro ciclo;
- X_e^2 : presença da aresta e no segundo ciclo;
- D_e : presença de duplicação da aresta e ;

Todas as variáveis de “presença” são decisões binárias com a seguinte interpretação de valores:

0 : ausente

1 : presente

Excelente!

2.2. Problema de Otimização

Minimizar:

$$\sum_{e \in E} c_e^1 X_e^1 + \sum_{e \in E} c_e^2 X_e^2 \quad (2.1)$$

Sujeito a:

$$\sum_{e \in \delta(v)} X_e^1 = 2 \quad \forall v \in V \quad (2.2)$$

$$\sum_{e \in \delta(v)} X_e^1 = 2 \quad \forall v \in V \quad (2.3)$$

$$\sum_{e \in E(S)} X_e^1 \leq |S| - 1 \quad \forall S \subseteq V, S \neq V, S \neq \emptyset \quad (2.4)$$

$$\sum_{e \in E(S)} X_e^2 \leq |S| - 1 \quad \forall S \subseteq V, S \neq V, S \neq \emptyset \quad (2.5)$$

$$X_e^1 + X_e^2 \leq 2 D_e \quad \forall e \in E \quad (2.6)$$

$$\sum_{e \in E} D_e \geq k \quad (2.7)$$

$$X_e^1, X_e^2, D_e \in \{0, 1\} \quad \forall e \in E \quad (2.8)$$

2.3. Explicação das Restrições

- A função objetivo (2.1) é soma do custo de todas as arestas selecionadas.
- As restrições (2.2) e (2.3) garantem que a quantidade de arestas incidentes em todos os vértices seja 2, nos ciclos 1 e 2, respectivamente. Essa condição faz com que todos os vértices tenham que ser visitados (duas arestas pois uma é a de “entrada” e a outra a de “saída”).
- As restrições (2.4) e (2.5) garantem que não existam subciclos nos ciclos. Nessas restrições, S é um subconjunto próprio e não-vazio dos vértices do problema. A expressão $E(S)$ é o conjunto das arestas cujos vértices (ambos) estão em S .
- A restrição (2.6) garante que, se uma aresta foi escolhida para ser duplicada, então essa aresta aparecerá nos dois ciclos.
- A restrição (2.8) garante que todas as variáveis são decisões binárias, ou seja, assumem apenas um de dois possíveis valores: 0 e 1.

2.4. Tamanho das Restrições

- Restrições (2.2) e (2.3): uma para cada vértice. Total: $2 \cdot |V|$;
- Restrições (2.4) e (2.5): uma para $S \in \mathcal{P}(V), S \neq V, S \neq \emptyset$. Total: $2 \cdot (2^{|V|} - 2)$;
- Restrições (2.6): uma para cada aresta. Total: $|E| = \frac{|V|^2 - |V|}{2}$ (já que o grafo é completo);
- Restrições 2.7: apenas uma. Total: 1;

Assim, o número total de restrições é:

$$T_r = 2 \cdot |V| + 2 \cdot (2^{|V|} - 2) + \frac{|V|^2 - |V|}{2} + 1 \quad (2.9)$$

Note que $T_r \in \mathcal{O}(2^{|V|})$, isto é, há um número exponencial de restrições.

Na implementação computacional do problema, não é possível adicionar tantas restrições por falta de recursos computacionais (memória e processamento). Há entretanto uma forma de contornar o problema através do que se chama de *lazy evaluation*. A ideia é não adicionar tais restrições no início. Conforme soluções factíveis são encontradas, verifica-se se há subciclos nelas e, caso sim, adiciona-se apenas as restrições necessárias para eliminar tais subciclos.

Dependendo do caso em mãos, essa abordagem pode reduzir drasticamente o número de restrições e consequentemente acelerar a busca.

3. Experimento Computacional

3.1. Configuração da Máquina

O problema foi executado num ideapad S145 81S90005BR: Lenovo IdeaPad S145 Notebook Intel Core i5-8265U (6MB Cache, 1.6GHz, 8 cores), 8GB DDR4-SDRAM, 460 GB SSD, Intel UHD Graphics 620.

O sistema operacional foi o Fedora 35 executando o Python 3.7.12 e Gurobi Optimizer v9.5.1rc2[1].

Como linguagem de programação, utilizamos Python[2] pela facilidade de uso e disponibilidade de ferramentas.

3.2. Dados do Problema

Os dados do problema foram fornecidos em um arquivo contendo 4 colunas e 250 linhas. A interpretação dos dados é a seguinte: cada linha representa um vértice e cada par de coluna as coordenadas desse vértice. A razão para um vértice ter duas posições diferentes é simplesmente para que as distâncias entre eles tenham valores diferentes no primeiro e no segundo ciclo.

Excelente!

Faltou arredondamento dos custos das arestas.

Quantidade de Vértices	Similaridade	Soluções Testadas	Soluções Fáctíveis	Custo da Solução	Gap de Optimalidade	Tempo de Execução (s)	Límite de Tempo (s)
100	0	12	7	1536.97	4.08	3.55	1800
100	50	12	10	2001.67	0	60.71	1800
100	100	12	3	3353.19	0	52.29	1800
150	0	12	3	1829.19	0	30.88	1800
150	75	12	5	2595.37	0	678.63	1800
150	150	12	1	4612.43	0	15.51	1800
200	0	12	1	2127.86	0	64.82	1800
200	100	12	1	14310.67	0.77	1800	1800
200	200	12	4	5785.59	0	242	1800
250	0	12	10	2379.07	7.43	512.57	1800
250	125	12	0	N/A	N/A	1800	1800
250	250	12	0	N/A	N/A	1800	1800

Figura 1: Resultados da Execução

O modelo na verdade precisa apenas de pesos. Construímos a primeira função de custo como a distância euclidiana entre os pontos das colunas 1 e 2. Da mesma forma, utilizamos distância euclidiana entre os pontos das colunas 3 e 4 para construir a segunda função de custo.

Note que, com essa interpretação, parece que temos dois conjuntos de vértices, um definido pelas colunas 1 e 2, e outro definido pelas colunas 3 e 4. Conforme explicitado no primeiro parágrafo da seção, esse não é o caso. Cada linha é um vértice e os valores fornecidos servem apenas para calcular a distância euclidiana e usá-la como peso para as arestas. Dessa forma, a aresta que liga os vértices representados pelas linhas 12 e 84, por exemplo, possui dois pesos diferentes, um para ser utilizado no primeiro ciclo e outro para ser utilizado no segundo.

3.3. Geração das Instâncias

Para gerar instâncias de um dado tamanho N , utilizamos as primeiras N linhas dos dados fornecidos.

4. Resultados

4.1. Resultados Obtidos

O programa foi construído para processar todas as instâncias e parâmetros pedidos no exercício. Mas devido a restrições computacionais, não conseguimos executar os dois últimos (Vértices-250 & Similaridade-0.5, Vértices-250 & Similaridade-1). Abaixo podemos ver a Tabela 1 dos resultados obtidos.

Podemos notar que, mesmo com o k similar, o aumento no número de Vértices também gera aumento do custo da solução, o que já era esperado pela teoria, conforme demonstrado na Tabela 1:

Com isso, foi possível compilar o custo da solução, de acordo com os parâmetros de Vértices e Similaridade, conforme a Figura 2

Assim como a relação do aumento de complexidade com o Tempo de Execução, conforme a Figura 3

# Vértices	Custo Mínimo	
	K	Custo
100	0	1536,97
150	0	1829,19
200	0	2127,86
250	0	2379,07
100	0.5	2001,67
150	0.5	2595,37
200	0.5	14310,67
250	0.5	N/A
100	1	3353,19
150	1	4612,43
200	1	5785,59
250	1	N/A

Tabela 1: Custo da soluções por Vértice/Similaridade

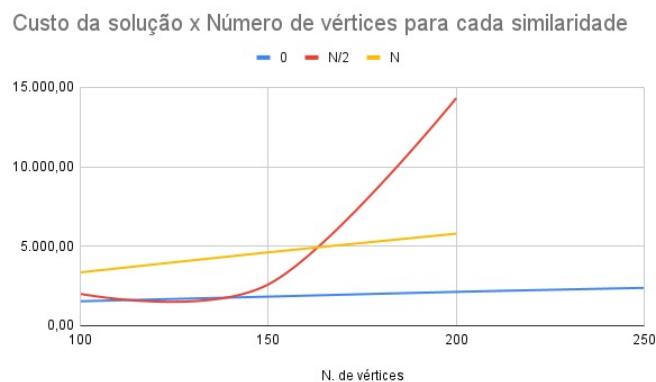


Figura 2: Resultados da Execução

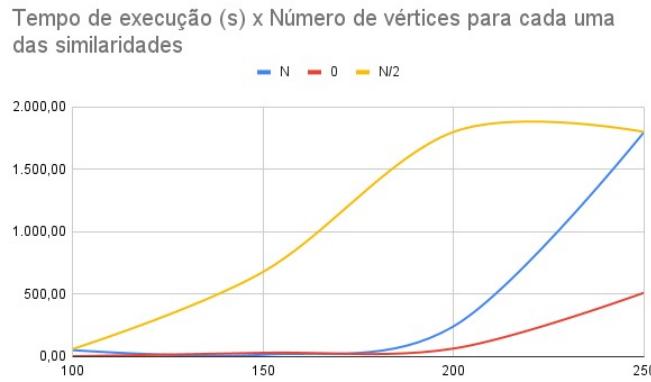


Figura 3: Resultados da Execução

5. Análise e Observações Finais

Baixar discussão

Podemos notar, ao avaliar o resultado das observações, que, para cada conjunto de vértices (em nosso caso: 100, 150, 200 e 250) temos um custo crescente com a adição de complexidade, ou seja, tanto com a adição de vértices, quanto com o aumento da similaridade (onde k pode ser igual a 0, 0,5, 1).

O aumento do custo da solução parece guardar uma relação linear com o aumento de vértices, embora fosse possível especular que a relação fosse exponencial a priori, dado o aumento da complexidade da solução. O mesmo é válido para o aumento da similaridade, representada pela variável k . O aumento do parâmetro de similaridade parece guardar uma relação linear de aumento com o custo da solução obtida.

O mesmo não pode ser dito em relação ao tempo computacional gasto para calcular as soluções. O tempo cresce exponencialmente de acordo com o aumento da complexidade da soluções, que é dada pela quantidade de vértices e pela similaridade entre os dois ciclos.

Este tipo de comportamento fez com que não conseguíssemos calcular as soluções para $k=0,5$ e $k=1$ para $V=250$. É importante notar que o enunciado do exercício nos colocou uma restrição de tempo de execução da busca por soluções em 30 minutos, ou seja, dentro deste tempo, o algoritmo não foi capaz de encontrar soluções viáveis para o problema. Testes com tempos maiores são recomendados para que se encontrem os resultados destas combinações.

Referências

- [1] L. Gurobi Optimization, “Gurobi optimizer reference manual,” 2022.
- [2] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.