Uma abordagem exata para o problema k-in-a-tree utilizando programação linear inteira

Lucas Saldanha Ferreira
Programa de Pós-graduação de Modelagem Matemática e Computacional - CEFET-MG
Av. Amazonas, 7675 - Nova Gameleira, Belo Horizonte - MG, 30510-000
lucas.subli@gmail.com

Elizabeth Fialho Wanner
Programa de Pós-graduação de Modelagem Matemática e Computacional - CEFET-MG
The Aston Lab for Intelligent Collectives Engineering, Aston University
B4 7ET, Birmingham, Reino Unido
e.wanner@aston.ac.uk

Vinícius Fernandes dos Santos
Programa de Pós-graduação de Modelagem Matemática e Computacional - CEFET-MG
Departamento de Ciência da Computação - UFMG
Av. Antônio Carlos, 6627, ICEx - Belo Horizonte - MG, 31270-901
viniciussantos@dcc.ufmg.br

RESUMO

Chudnovsky e Seymour, em 2006, publicaram um algoritmo que resolve, em tempo polinomial, o problema denominado THREE-IN-A-TREE: dados um grafo simples e três vértices arbitrários pertencentes a este grafo, verificar se existe uma subárvore induzida que contenha os três vértices dados. Neste trabalho iremos lidar com uma generalização natural deste problema, denominada k-IN-A-TREE. O problema, portanto, consiste em testar, dado um grafo, se existe uma subárvore induzida que contenha k vértices determinados. Quando o número de vértices fixos k é parte da entrada, o problema é conhecidamente NP-Completo. Quando, ao invés de um k como parte da entrada, temos um k fixo, a complexidade está aberta para $k \geq 4$ para o caso geral. Para casos restritos existem algoritmos eficiente, como nos grafos livres de garras, grafos de cintura maior igual a k e grafos cordais. Neste trabalho apresentamos formulações de Programação Linear Inteira para este problema e verificamos experimentalmente que é possível resolver de forma exata instâncias de tamanho moderado a grande.

PALAVRAS CHAVE. k-in-a-tree, three-in-a-tree, subgrafo induzido, MTZ.

Otimização Combinatória, Teoria e Algoritmos em Grafos

ABSTRACT

Chudnovsky and Seymour proposed the three-in-a-tree algorithm which solves the following problem in polynomial time: given a three fixed vertices in a finite simple Graph, check if an induced tree containing these vertices exists. In this paper we deal with a generalization of this problem, referred as k-IN-A-TREE problem. The k-IN-A-TREE problem, therefore, is to test whether a graph contains an induced tree spanning k given vertices. When k is part of the input, the problem is known to be NP-complete. For fixed k, the complexity of this problem is open for $k \geq 4$, although there are better time algorithms for restricted cases, such as claw-free graphs, graphs with

https://proceedings.science/p/85374?lang=pt-br

girth at least k, and chordal graphs. In this paper we will propose an optimization approach to solve the problem.

KEYWORDS. k-in-a-tree, three-in-a-tree, induced subgraphs, MTZ.

Combinatory Optmization, Graph theory and Algorithms

https://proceedings.science/p/85374?lang=pt-br

1. Introdução

Em 2006 Chudnovsky e Seymour publicaram [Chudnovsky e Seymour, 2010] o algoritmo three-in-a-tree que resolve, em tempo polinomial, o seguinte problema: Dado um grafo $G = (V, E)$ e três vértices $u, v, w \in V$ como entrada, encontrar um grafo T tal que T seja ao mesmo tempo um subgrafo induzido de G, uma árvore e contenha u, v e w. Caso não exista T que conforme com as condições, reportar a impossibilidade.

O trabalho se tornou muito útil na comunidade como uma ferramenta para testes de existência de certas classes de subgrafos induzidos, especialmente no reconhecimento de classes de grafos definidas pela proibição de certos subgrafos induzidos [Vuskovic, 2013]. Um outro exemplo da utilidade do algoritmo foi a utilização deste para o reconhecimento de grafos livres de buracos ímpares [Chang e Lu, 2015].

Neste trabalho será tratada a generalização natural do problema citado, denominada k-IN-A-TREE. É fácil de perceber que para $k = 1$ e $k = 2$ o problema é trivial, para $k = 3$ o problema foi resolvido pelo algoritmo *three-in-a-tree* [Chudnovsky e Seymour, 2010] e para k fixo tal que $k \geq 4$ a ordem de complexidade do problema ainda está em aberto. Quando, ao invés de um k fixo, temos este elemento como parte da entrada o problema se torna conhecidamente NP-difícil [Derhy e Picouleau, 2009]. Para o caso de k fixo são conhecidos algoritmos eficientes para grafos com características específicas, como grafos livres de garras [Fiala et al., 2012], grafos de cintura maior igual a k [Liu e Trotignon, 2010] e grafos cordais [dos Santos et al.]. Por fim, mesmo com a complexidade aberta para $k \geq 4$, problemas de natureza similar são conhecidamente NP-difíceis, como testar a existência de um ciclo induzido contendo k vértices [Bienstock, 1991] e encontrar um caminho induzido contento k vértices (para $k \geq 3$) [Haas e Hoffmann, 2006].

As principais contribuições deste trabalho incluem:

- Apresentar a primeira formulação linear inteira para o problema k-IN-A-TREE;
- Verificação experimental da capacidade do modelo de resolver instâncias de tamanho relativamente grande em pouco tempo;
- Estudo preliminar sobre os parâmetros ou características das instâncias que podem influenciar os tempos de execução quando aplicado o modelo.

2. Notações e Terminologia

Considere que, ao longo deste trabalho, todos os grafos devem ser considerado finitos, conexos, não direcionados e simples, a não ser que o contrário seja explicitamente comunicado. Dizemos que H é um subgrafo induzido de um grafo G se ambas forem atendidas:

1. $V(H) \subseteq V(G)$
2. $E(H) = \{uv \in E(G) \mid u, v \in V(H)\}$.

Apresentamos, agora, a definição formal do problema a ser tratado neste trabalho.

K-IN-A-TREE
INSTÂNCIA: Um grafo $G = (V, E)$ e um conjunto $K \subseteq V(G)$.
PERGUNTA: Existe subgrafo induzido T de G, com $K \subseteq V(T)$ tal que T seja uma árvore?

Podemos assumir que G é conexo pois, caso contrário, podemos trivialmente reduzir G a um grafo conexo: 1) caso G não seja conexo e existam $u, v \in K$ que façam parte de componentes conexas diferentes é óbvio que não existe solução para o problema; 2) caso G não seja conexo e K esteja completamente contido em uma única componente conexa, podemos eliminar todas as outras componentes conexas da entrada.

https://proceedings.science/p/85374?lang=pt-br

3. Modelagem

A abordagem proposta neste trabalho para o K-IN-A-TREE é a utilização de otimização linear inteira, assegurando aciclicidade através de restrições e maximizando a função objetivo $|V(T)| - |E(T)|$. O princípio básico da formulação que será apresentada parte da definição de árvore como um grafo acíclico e conexo. Além disso toda árvore T satisfaz $|V(T)| - |E(T)| = 1$.

Com a garantia de aciclicidade pelas restrições, é trivial deduzir que o menor valor possível para a função objetivo $|V(T)| - |E(T)|$ será 1. Caso este valor seja alcançado, podemos afirmar que T é uma árvore. Será assegurado que T é um subgrafo induzido de G, acíclico e que $K \subseteq T$ através de restrições. Desta forma, o problema original possuirá solução se e somente se a função objetivo alcançar o valor 1.

Para garantir a aciclicidade do grafo, a formulação MTZ [Miller et al., 1960] será usada de forma modificada. Ela foi originalmente desenvolvida para o problema do caixeiro viajante e é amplamente utilizada para a eliminação de ciclos indesejados [Pataki, 2000]. Após a explicação da formulação MTZ e como modificá-la para atender às necessidades do problema em questão, a seção seguinte irá explicar as demais restrições.

3.1. Formulação MTZ

Para explicar a formulação, primeiramente, deve-se formular o problema do caixeiro viajante, que pode ser modelado por grafo direcionado $D = (N, A)$ onde busca-se o *tour* (um ciclo direcionado que contenha todos os elementos) de menor caminho. Denota-se custo das arestas por c_{ij}, e a existência das arestas por $x_{i,j}$, que representam se a aresta ij está na solução ou não. A aresta ij é aquela que conecta o vértice i ao vértice j. As soluções factíveis do problema de programação inteira

$$\begin{aligned} \text{min} \quad & \sum_{i,j} c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{i} x_{ij} = 1 \quad \forall i \\ & \sum_{j} x_{ij} = 1 \quad \forall i \\ & x_{ij} \in \mathbb{B} \end{aligned} \tag{1}$$

vão geralmente conter ciclos direcionados, denominados subtours. Para excluir os subtours da solução pode-se utilizar variáveis adicionais $u_i (i = 1, \ldots, n)$, na forma

$$\begin{aligned} & u_1 = 1 \\ & 2 < u_i < n \qquad\qquad \forall i \neq 1 \\ & u_i - u_j + 1 \leq n(1 - x_{ij}) \quad \forall i \neq 1, \forall j \neq 1 \end{aligned} \tag{2}$$

A última restrição em (2) é conhecida, em inglês, como *arc-constraint* para (i, j), e quando unidas as formulações (1) e (2) temos a formulação de Miller-Tucker-Zemlin, ou MTZ. De fato podemos observar que as restrições apresentadas eliminam subtours pois: 1) as *arc-constraints* para (i, j) forçam $u_j \geq u_i + 1$ quando $x_{i,j} = 1$; 2) Se a solução factível de (1) e (2) contiver mais de um subtour, pelo menos um destes subtours iria conter o nó 1, e ao longo deste subtour os valores de u_i tenderiam a aumentar infinitamente (implicando que o único valor factível para u_i é a posição do nó i no tour, uma forma de ordenação) [Desrochers e Laporte, 1991].

Apesar de extremamente interessante, esta formulação, conforme apresentado anteriormente, permite um e apenas um ciclo como esperado do problema do caixeiro viajante. Portanto a formulação será utilizada de forma modificada, visando inibir todo e qualquer ciclo, conforme necessário em uma árvore. Alterações similares são comuns em formulações no problema da árvore de Steiner: dado um grafo não direcionado com pesos não negativos nas arestas e um subconjunto

https://proceedings.science/p/85374?lang=pt-br

de vértices denominados terminais, o problema consiste em encontrar uma árvore de peso mínimo que contenha todos os terminais. Este problema, em grafos, pode ser visto como uma generalização de dois outros problemas combinatórios famosos: o problema de menor caminho, e a árvore geradora mínima [Chlebík e Chlebíková, 2008]. Se por um lado com dois terminais o problema é simplificado para o problema do menor caminho, quando todos os vértices são terminais o problema se torna o da árvore geradora mínima. A maior parte das varições do problema da árvore de Steiner é NP-difícil [Klein e Ravi, 1995], apesar disso, muitas delas podem ser resolvidas de maneira eficiente na prática, mesmo para problemas reais de larga escala [Bern e Graham, 1989]. Este problema, por ser muito estudado, possui formulações eficientes, muitas vezes baseadas na MTZ, para a remoção de ciclos e de subtours em busca de uma árvore, assim como o problema em questão.

3.1.1. Modificações do MTZ

Na formulação, y_i indica se o vértice i está na solução ou não. As variáveis $x_{i,j}$ indicam se existe uma aresta conectando i e j na solução. É importante ressaltar que a matriz x não é simétrica, apesar de estarmos lidando com grafos não direcionados, e caso exista uma aresta entre i e j ela será representada por x_{ij} **ou** x_{ji} e nunca ambos. Apesar desta notação se assemelhar à de grafos direcionados, este não é o caso: esta peculiaridade se deve à forma de funcionamento da formulação MTZ. Outro ponto peculiar que merce destaque é que devido ao fato de todos os grafos serem simples, conforme colocado na Seção 2, podemos afirmar também que $x_{ii} = 0$ para todo i.

O conjunto de restrições que garante aciclicidade que será usado pode ser obtido através de algumas modificações na formulação MTZ. Seguindo a lógica previamente mencionada desta formulação, escolhe-se um vértice como raiz que não irá receber nenhuma aresta, apenas enviar. Na formulação original este vértice poderia ser escolhido arbitrariamente pois todos os vértices fariam parte da solução final. Nesta variação, o vértice deverá pertencer a K, pois estes são os únicos vértices que necessariamente serão parte da solução. Isso é garantido pela Equação (3).

$$\sum_{i=1}^{n} x_{i1} = 0 \tag{3}$$

Em seguida para cada vértice j uma restrição é necessária para forçar j a ter exatamente uma aresta chegando, caso j esteja na solução, ou nenhuma caso contrário. Esta etapa é mais uma adaptação da formulação MTZ, uma vez que nem todos os vértices necessariamente estarão na solução final. A modelagem está na Equação (4).

$$\sum_{i=1}^{n} x_{ij} = y_j \qquad \forall j \neq 1 \tag{4}$$

Um conjunto adicional de variáveis de decisão u_i será necessário para indexar os vértices. Para a formulação ser efetiva o nó raiz (escolhido conforme mencionado nos parágrafos anteriores) deverá ter o menor valor u de todos, por simplicidade nó raiz será denotado como nó 1 e seu valor u será $u_1 = 1$. Além disso todo valor u_i deve estar entre 2 e $\sum_{i=1}^{n} y_i$. Desta forma é possível, porém não necessário, que cada u seja único. Estas propriedades são representadas pelas Equações (5) e (6).

$$u_1 = 1 \tag{5}$$

$$2 < u_i < n \qquad \forall i \neq 1. \tag{6}$$

Por fim um novo conjunto de restrições será introduzido para impor uma estrutura adicional na variável u_i. Esta estrutura, junto com as anteriores, irá proporcionar a eliminação de ciclos.

https://proceedings.science/p/85374?lang=pt-br

Um vértice i poderá enviar arestas para um vértice j se e apenas se u_j for maior que u_i. Isso define uma ordenação parcial dos vértices contidos na solução através dos valores u_i atribuídos a cada vértice i. A formulação original do MTZ continha uma exceção feita ao primeiro vértice (definido como raiz previamente) fazendo a última restrição se aplicar para todo $i \neq 1$ and $j \neq 1$, gerando um único ciclo contendo todos os vértices. Como isto é interessante apenas no problema do caixeiro viajante e não para k-in-a-tree, a restrição foi alterada para ser válida para todo i, j. Em resumo, a versão modificada, aqui apresentada, irá garantir que apenas vértices de um dado valor u poderão enviar arestas para vértices de um valor u maior, e só poderão receber arestas de vértices com um valor u menor. Como cada vértice pode receber uma e apenas uma aresta nenhum ciclo poderá existir sem violar uma dessas restrições. A Equação (7) irá produzir o efeito de ordenação desejado.

$$u_i - u_j + 1 \leq n(1 - x_{ij}) \qquad \forall i, j \qquad (7)$$

A restrição é mais simples de se entender quando separada em duas situações:

- Se a aresta x_{ij} não está na solução (e portanto tem valor 0) a restrição pode ser simplificada para $u_i - u_j + 1 \leq n$, o que é sempre verdade (conforme Equação (6)).
- Se a aresta x_{ij} está presente a restrição pode ser lida como $u_i + 1 \leq u_j$. Implicando em $u_i < u_j$.

A combinação das restrições (5), (6) e (7) é conhecida, na formulação MTZ, como restrições de eliminação de subtour (modificadas para se adequarem ao problema).

3.2. Subgrafos induzidos e K

As restrições relativas à $K \subseteq V(T)$ e a restrição que garante T ser um subgrafo induzido de G serão explicadas nesta seção.

Conforme explicado anteriormente, y_i indica se o vértice i está na solução ou não - e portanto $\sum_{i=1}^{n} y_i$ é o número de vértices na solução. As variáveis $x_{i,j}$ indicam se existe uma aresta conectando i e j na solução, e $\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}$ é o número de arestas na solução. Note que $\sum_{i=1}^{n} x_{ij}$ indica o número de arestas chegando no vértice j e da mesma forma $\sum_{j=1}^{n} x_{ij}$ indica o número de arestas saindo do vértice i. Por fim z é a matriz de adjacência (simétrica) de entrada que indica se o vértice i se conecta ao vértice j no grafo G de entrada, enquanto k é um vetor de n posições, uma para cada vértice que existe em G. Especificamente, $k_i = 1$, se $i \in K$, e $k_i = 0$, caso contrário.

$$x_{ij} \leq y_i \qquad \forall i, j \qquad (8)$$
$$x_{ij} \leq y_j \qquad \forall i, j \qquad (9)$$
$$x_{ij} \leq z_{ij} \qquad \forall i, j \qquad (10)$$
$$x_{ij} + x_{ji} \leq 1 \qquad \forall i, j \qquad (11)$$
$$x_{ij} + x_{ji} \geq y_i + yj + z_{ij} - 2 \qquad \forall i, j \qquad (12)$$
$$y_i \geq k_i \qquad \forall i \qquad (13)$$
$$n \in \mathbb{N}^* \qquad (14)$$
$$x, y \in \mathbb{B} \qquad (15)$$

As restrições (8) e (9) garantem que uma aresta não será selecionada caso ambos os vértices também não estejam selecionados. A restrição (10) garante que uma aresta só pode fazer parte da solução caso ela exista em G, (11) garante que uma aresta não é escolhida duas vezes

https://proceedings.science/p/85374?lang=pt-br

e (12) garante que caso a aresta exista em G e tanto y_i quanto y_j estejam na solução, a aresta necessariamente também esterá na solução. Estas cinco restrições asseguram que soluções factíveis sempre são subgrafos induzidos de G.

As três próximas restrições - Equações (13), (14) e (15) - garantem que K pertence à solução e os domínios das variáveis de decisão.

Por fim, a função objetivo $|V(G)|-|E(G)|$, dada que a aciclicidade está garantida, poderá atingir um valor mínimo de 1. É fácil perceber que este valor somente será atingido se e somente se o grafo seja composto de uma única componente conexa. A função objetivo está formalmente escrita na Equação (16).

$$\min \sum_{i=1}^{n} y_i - \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} \tag{16}$$

Ao combinar restrições de (4) até (15) com a função objetivo (16) obtemos uma formulação inteira na qual o menor valor possível (de função objetivo) é 1, que ocorrerá se e somente se, para o problema original, existir uma subárvore induzida (grafo conexo e acíclico) que contém todos os vértices K, *i.e.*, (G, K) é uma instância positiva.

O problema final (com a adição de algumas desigualdades válidas) pode ser visto na Equação (17).

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} y_i - \sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} & \\
\text{subject to} \quad & x_{ij} \leq y_i & \forall i, j \\
& x_{ij} \leq y_j & \forall i, j \\
& x_{ij} \leq z_{ij} & \forall i, j \\
& x_{ij} + x_{ji} \leq 1 & \forall i, j \\
& x_{ij} + x_{ji} \geq y_i + y_j + z_{ij} - 2 & \forall i, j \\
& y_i \geq k_i & \forall i \\
& \sum_{i=1}^{n} x_{i1} = 0 & \\
& \sum_{i=1}^{n} x_{ij} = y_j & \forall j \neq 1 \\
& u_1 = 1 & \\
& 2 < u_i < n & \forall i \neq 1 \\
& u_i - u_j + 1 \leq n(1 - x_{ij}) & \forall i, j \\
& (1 - y_i)n \leq u_i & \forall i \neq 1 \\
& (1 - y_i)(n - 2) + 2 \leq u_i & \forall i \neq 1 \\
& n \in \mathbb{N}^* & \\
& x, y \in \mathbb{B} &
\end{aligned} \tag{17}$$

4. Resultados

O problema de otimização detalhado na seção anterior foi implementado na linguagem C++, utilizando a biblioteca de otimização IBM CPLEX V12.7.1, as configurações da biblioteca foram deixadas nos valores *default*. Os códigos foram executados em uma máquina com processador i7-4700k, 32 GB de memória RAM, rodando o sistema operacional Ubuntu versão 16.04.

https://proceedings.science/p/85374?lang=pt-br

O código implementado foi testado em dois tipos de instâncias de grafos: instâncias de benchmark disponíveis publicamente para outros problemas e grafos gerados aleatoriamente. Todos os resultados positivos (existência de uma solução) foram validados. Os resultados negativos (inexistência de solução) não foram verificados. Isso se deve ao fato que a verificação da validade de uma possível solução para o problema é trivial, enquanto detectar a inexistência de uma solução não é, uma vez que não existe um certificado simples para tal.

Quando usado para resolver instâncias públicas de testes para outros tipos de problema, como problemas de coloração [Col], e árvores de Steiner [Koch et al., 2000], o algoritmo resolveu a grande maioria em tempos inferiores a um segundo, tornando os resultados eficientes porém pouco interessantes. Esse fato pode ser atribuído ao fato de instâncias destas classes específicas possuírem construções que dificultam a resolução de seus respectivos problemas, porém não afetam ou até mesmo facilitam o problema em questão.

Desta forma optou-se pela geração de instâncias completamente aleatórias. Para a geração das instâncias define-se o número de vértices, a probabilidade de existência de uma aresta e também o número de vértices obrigatórios. Manipulando-se a probabilidade de existência de arestas é possível conseguir instâncias de densidade específicas. Porém, como a geração é aleatória não se pode garantir um número exato de arestas nas instâncias.

Para os testes foram gerados 5 tipos de instâncias de grafos aleatórios com densidades de arestas diferentes. Para cada tipo foram gerados 30 instâncias diferentes para cada um de 4 valores de n distintos, sendo eles: 50, 500, 5.000 e 50.000. O número de vértices obrigatórios ($|K|$) foi fixado em 5, 10 e 20. No total 1.800 instâncias diferentes foram geradas.

- **Tipo 1**: Densidade de aproximadamente $2n$ - Resultados apresentados na tabela 1;
- **Tipo 2**: Densidade de aproximadamente $5n$ - Resultados apresentados na tabela 2;
- **Tipo 3**: Densidade de aproximadamente $n \log n$ - Resultados apresentados na tabela 3;
- **Tipo 4**: Densidade de aproximadamente $n \sqrt{n}$ - Resultados apresentados na tabela 4;
- **Tipo 5**: Densidade de aproximadamente $\binom{n}{2}/2$ - Resultados apresentados na tabela 5;

Nas tabelas que seguem os resultados para $n = 50.000$ são mostrados como percentual de instâncias resolvidas dentro do limite de tempo de 2 horas. Todos os outros resultados são mostrados em segundos de execução do algoritmo.

n	$\|K\| = 5$	$\|K\| = 10$	$\|K\| = 20$
50	0.01 ($\sigma = 0$)	0.01 ($\sigma = 0$)	0.01 ($\sigma = 0$)
500	0.01 ($\sigma = 0$)	1.32 ($\sigma = 1.01$)	5.32 ($\sigma = 2.35$)
5.000	50.73 ($\sigma = 55.42$)	120.77 ($\sigma = 86.89$)	210.10 ($\sigma = 180.65$)
50.000	83%	77%	77%

Tabela 1: Resultado para instâncias de tipo 1.

n	$\|K\| = 5$	$\|K\| = 10$	$\|K\| = 20$
50	0.01 ($\sigma = 0$)	0.01 ($\sigma = 0$)	0.01 ($\sigma = 0$)
500	0.17 ($\sigma = 2.56$)	3.53 ($\sigma = 2.11$)	5.06 ($\sigma = 2.45$)
5.000	40.87 ($\sigma = 22.76$)	128.36 ($\sigma = 167.98$)	240.23 ($\sigma = 434.76$)
50.000	77%	77%	63%

Tabela 2: Resultado para instâncias de tipo 2.

https://proceedings.science/p/85374?lang=pt-br

n	$\|K\| = 5$	$\|K\| = 10$	$\|K\| = 20$
50	0.01 ($\sigma = 0$)	0.01 ($\sigma = 0$)	0.01 ($\sigma = 0$)
500	0.01 ($\sigma = 0$)	1.32 ($\sigma = 7.87$)	5.32 ($\sigma = 3.54$)
5.000	80.73 ($\sigma = 67.33$)	165.77 ($\sigma = 190.87$)	268.68 ($\sigma = 356.56$)
50.000	50%	57%	43%

Tabela 3: Resultado para instâncias de tipo 3.

n	$\|K\| = 5$	$\|K\| = 10$	$\|K\| = 20$
50	0.01 ($\sigma = 0$)	0.01 ($\sigma = 0$)	0.01 ($\sigma = 0$)
500	0.84 ($\sigma = 1.45$)	2.65 ($\sigma = 6.56$)	2.88 ($\sigma = 6.99$)
5.000	80.50 ($\sigma = 101.41$)	223.43 ($\sigma = 345.89$)	855.00 ($\sigma = 1112.76$)
50.000	43%	53%	37%

Tabela 4: Resultado para instâncias de tipo 4.

n	$\|K\| = 5$	$\|K\| = 10$	$\|K\| = 20$
50	0.01 ($\sigma = 0$)	0.01 ($\sigma = 0$)	0.01 ($\sigma = 0$)
500	0.32 ($\sigma = 3.65$)	1.21 ($\sigma = 2.77$)	1.27 ($\sigma = 1.43$)
5.000	75.23 ($\sigma = 95.43$)	387.63 ($\sigma = 567.54$)	485.01 ($\sigma = 789.08$)
50.000	57%	57%	43%

Tabela 5: Resultado para instâncias de tipo 5.

Os resultados dos experimentos sugerem um crescimento não polinomial do tempo em relação à dimensão de entrada. Além disso, ficou claro que, em grafos com densidades de arestas muito alta ou muito baixa, a facilidade em se encontrar soluções é maior. Este fato pode estar relacionado à maior probabilidade de infactibilidade em grafos desta natureza, ou a um número muito grande de soluções viáveis, respectivamente, o que levaria o algoritmo a não perder tempo em busca de soluções. Em contrapartida, grafos com densidades intermediárias possuem um espaço de busca de soluções candidatas muito maior, o que pode prejudicar o tempo de execução. Esta hipótese é corroborada pelos altos desvios padrão encontrados, em soluções infactíveis os tempos tendem a ser muito menores do que em soluções factíveis, levando a uma grande diferença de tempo de execução entra as amostras que gera um desvio padrão alto.

Por fim, há indícios de que o tamanho de K exerce influência relevante no tempo de execução do algoritmo, independente da densidade do grafo. Este resultado parece natural sabendo existe algoritmo polinomial para o caso $|K| = 3$ e NP-difícil no caso geral. Os dados corroboram esta conclusão simplista. O tamanho de K em função de n também parece ser relevante do ponto de vista teórico uma vez que quando $|K| = n$ o problema pode ser simplificado em verificar se G é uma árvore, e para $|K| = n - c$, para alguma constante c, o problema é igualmente simples de se verificar em tempo polinomial $O(n^c)$. Infelizmente os tempos computacionais para estes estudos ainda são proibitivos em instâncias grandes. Estudos mais aprofundados são necessários para validar estas hipóteses e já se encontram em andamento.

5. Conclusão

A modelagem e resolução do problema foi efetiva e os resultados atuais são promissores. Através da modelagem proposta é viável resolver problemas de dimensão de entrada relativamente grande, por vezes superior à 50.000 vértices, em tempo hábil. Mesmo com estes resultados ainda existe muito espaço para melhorias tanto no modelo quanto na sua implementação. Também existe a necessidade de uma análise mais profunda e estatisticamente mais relevante dos resultados, além da comparação dos resultados em outros tipos de instâncias de grafos.

As principais melhorias e trabalhos futuros podem ser encontradas na implementação e utilização dos métodos de programação inteira: identificar e implementar técnicas de diminuição

do espaço de soluções factíveis e desigualdades válidas relevantes, estudar seu impacto no tempo de execução, além de melhorar o gerenciamento de memória quando trabalhando com grafos de densidade de arestas elevados.

Instâncias com características específicas também podem ser consideradas. Como exemplo, intuitivamente é possível que a posição relativa dos elementos de K dentro do grafo desempenhe um papel importante na eficiência da solução.

Outras direções de investigação já em andamento incluem desigualdades alternativas para garantia de aciclicidade e também novas formulações com garantia de conectividade através da formulação, com a aciclicidade garantida na otimalidade.

Agradecimentos

Gostaríamos de agradecer o professor Dr. Cristiano Arbex Valle, do DCC/UFMG por disponibilizar os códigos de sua biblioteca pessoal, e pelo tempo gasto ajudando a encontrar a melhor forma de codificar e testar o problema.

Agradecemos ao CNPq, CAPES, FAPEMIG e CEFET-MG pelo apoio financeiro.

Referências

MS Windows NT kernel description. `http://mat.gsia.cmu.edu/COLOR/instances.html`. Accessed: 2018-02-20.

Bern, M. W. e Graham, R. L. (1989). The shortest-network problem. *Scientific American*, 260(1): 84–89.

Bienstock, D. (1991). On the complexity of testing for odd holes and induced odd paths. *Discrete Mathematics*, 90(1):85–92.

Chang, H.-C. e Lu, H.-I. (2015). A faster algorithm to recognize even-hole-free graphs. *Journal of Combinatorial Theory, Series B*, 113:141–161.

Chlebík, M. e Chlebíková, J. (2008). The steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science*, 406(3):207–214.

Chudnovsky, M. e Seymour, P. (2010). The three-in-a-tree problem. *Combinatorica*, 30(4):387–417.

Derhy, N. e Picouleau, C. (2009). Finding induced trees. *Discrete Applied Mathematics*, 157(17): 3552–3557.

Desrochers, M. e Laporte, G. (1991). Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. *Operations Research Letters*, 10(1):27–36.

dos Santos, V. F., da Silva, M. V., e Szwarcfiter, J. L. The k-in-a-tree problem for chordal graphs. *Matemática Contemporânea*, 44:1–10.

Fiala, J., Kamiński, M., Lidickỳ, B., e Paulusma, D. (2012). The k-in-a-path problem for claw-free graphs. *Algorithmica*, 62(1-2):499–519.

Haas, R. e Hoffmann, M. (2006). Chordless paths through three vertices. *Theoretical Computer Science*, 351(3):360–371.

Klein, P. e Ravi, R. (1995). A nearly best-possible approximation algorithm for node-weighted steiner trees. *Journal of Algorithms*, 19(1):104–115.

Koch, T., Martin, A., e Voß, S. (2000). SteinLib: An updated library on steiner tree problems in graphs. Technical Report ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin. URL `http://elib.zib.de/steinlib`.

https://proceedings.science/p/85374?lang=pt-br

Liu, W. e Trotignon, N. (2010). The k-in-a-tree problem for graphs of girth at least k. *Discrete Applied Mathematics*, 158(15):1644–1649.

Miller, C. E., Tucker, A. W., e Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329.

Pataki, G. (2000). The bad and the good-and-ugly: formulations for the traveling salesman problem.

Vuskovic, K. (2013). The world of hereditary graph classes viewed through truemper configurations. *Surveys in Combinatorics 2013*, 409:265.

https://proceedings.science/p/85374?lang=pt-br