Árvore geradora mínima quadrática

Problema NP-Dificil - Projeto e Análise de Algoritmos

Felipe C. Ramos, Paulo L. Medeiros

Agosto - 2019

Conteúdo

1 Introdução

O problema da árvore geradora mínima quadrática ($AGMQ$) é um problema de otimização combinatorial e foi apresentado em [1] por *Assad e Xu* em 1992. Em seu artigo, além da definição do problema, foi apresentada uma prova de que o problema apresentado é *NP-difícil*, limites inferiores para a AGMQ, um algoritmo exato e três algoritmos heurísticos.

Ainda nessa seção, será apresentada a definição do problema e de algumas variantes, seguidos de sua aplicabilidade. Na seção 2 serão comentados trabalhos na literatura sobre a AGMQ. A demonstração de que o problema é NP-Difícil feita em [1] será descrita na seção 3. Na quarta seção será apresentada uma das heurísticas propostas em [1]. Na seção 5 serão apresentados algoritmos exatos desenvolvidos para o problema e na sexta seção os resultados, descrições e discussões dos experimentos executados.

1.1 Definição do problema

O problema da árvore geradora mínima quadrática pode ser descrito como uma extensão do problema da árvore geradora mínima (AGM), já muito conhecida pela academia.

Seja $G = (V, E)$ um grafo conexo não-direcionado, tal que $V = \{v_1, ..., v_n\}$ e $E = \{e_1, ..., e_m\}$, e $\mathscr{T}$ o conjunto com todas as árvores geradoras de G.

Considere agora uma função de custo $l : E \to \mathbb{R}$. O problema da AGM pode ser descrito como:

$$\min_{t \in \mathscr{T}} \; z(t) = \sum_{e \in E(t)} l(e) \tag{1}$$

Defina agora uma outra função $q : E \times E \to \mathbb{R}$, onde $q(e, e) = 0$, para todo $e \in E$. Note que consideramos l como o custo linear sobre G, enquanto q é considerado o custo quadrático. O problema da AGMQ é então descrito por:

$$\min_{t \in \mathscr{T}} \; z(t) = \sum_{e \in E(t)} l(e) + \sum_{(e,e') \; \in \; E(t) \times E(t)} q(e, e') \tag{2}$$

Um caso específico da $AGMQ$ é o problema da árvore geradora mínima quadrática em adjacência de arestas ($AGMQA$), onde $q(e, e') = 0$ se e não é adjacente a e'. Ou seja, só é considerado quadrático de arestas adjacentes.

Já em contraste à formulação mono-objetivo do problema, o problema da $AGMQ$ apresenta uma outra versão, a multi-objetivo. Ao invés de ser considerada apenas uma função objetivo, os dois termos da soma de (2) são vistos como dois objetivos diferentes e é procurado minimizar os dois sob diferentes circunstâncias.

1.2 Aplicabilidade

A estrutura definida permite interações entre arestas que podem ser modeladas. Por exemplo, numa transferência entre diferentes canos, o custo pode depender da relação entre os canos, principalmente quando consideramos o caso AGMQA. A ideia também pode ser aplicada para a relação entre cabos subterrâneos e em superfície.

2 Trabalhos relacionados

Após a introdução da *AGMQ* em 1992, novos algoritmos - exatos e heurísticos - foram apresentados à academia. Junto a estes, novas formas de calcular limites inferiores para o uso em *branch-and-bound*, por exemplo, foram calculados.

As seguintes subseções apresentam um apanhado de alguns trabalhos relacionados ao problema da AGMQ mono-objetivo. Em [7] pode ser encontrado um estudo sobre a versão multi-objetivo.

2.1 Algoritmos heurísticos

Nessa seção serão apresentados algoritmos heurísticos (e meta-heurísticos) desenvolvidos a fim de aproximar soluções ótimas para o problema. [12] foi o primeiro a propor um algoritmo meta-heurístico para abordar o problema, apresentando um algoritmo genético que utiliza sequência de Prüfer para codificar as árvores. Um novo algoritmo genético foi proposto por [10], utilizando então a representação *edge-window-decoder* para as árvores.

Um algoritmo utilizando a meta-heurística de colônia de abelhas acompanhada de uma busca local foi proposto por [11]. [8] apresentou um outro algoritmo meta-heurístico: inicialmente, é realizado um recozimento simulado, e então é aplicada uma busca tabu ou um algoritmo genético híbrido. [2] e [6] apresentam outros algoritmos de busca tabu para o problema da *AGMQ*.

Mais recentementemente, foi proposto um algoritmo transgenético para o problema em [4].

2.2 Algoritmos exatos e limites inferiores

Além dos propostos por [1], outros algoritmos exatos foram propostos por [2] e [9], ambos usando a técnica *branch-and-bound*. Em [9] também são calculados novos limites inferiores a serem usados na técnica citada.

3 Complexidade do problema da AGMQ

Em [1] a demonstração de que o problema da AGMQ é NP-Difícil foi feita a partir da redução de dois problemas da classe NP-Completo: o problema de achar um caminho hamiltoniano e o problema de alocação quadrática. Aqui, iremos descrever apenas a redução para o primeiro problema. Uma leitura completa pode ser encontrada no texto original.

3.1 Redução do problema do caminho hamiltoniano

O problema do caminho hamiltoniano já é conhecido na literatura e consiste em determinar se em um dado grafo $G = (V, E)$ existe um caminho hamiltoniano, isto é, um caminho que passe por todos os vértices exatamente uma vez. Esse problema é NP-Completo.

Lema 1. *Uma árvore com n vértices tem $n - 1$ arestas.*

Lema 2. *Para qualquer grafo $G = (V, E)$, onde $|E| = m$, $|V| = n$ e d_i é o grau do i-ésimo vértice, $1 \leq i \leq n$, temos que*

$$\sum_{i=1}^{n} d_i = 2m$$

Teorema 3. *É possível realizar uma redução em tempo polinomial do problema do caminho hamiltoniano para o problema da AGMQ.*

Demonstração. Será considerado o caso especial do problema para adjacência de arestas, a AGMQA. Dado um grafo não trivial $G = (V, E)$ que é instância do problema do caminho hamiltoniano, onde $|V| = n$ e $|E| = m$, defina

$$q(e, e') = \begin{cases} 1 & \text{se } e \text{ e } e' \text{ são adjacentes} \\ 0 & \text{c.c.} \end{cases}$$

e $l(e) = 0$ para todo $e \in E$.

Considere uma árvore geradora mínima $t \in \mathscr{T}$ solução (não necessariamente ótima) para o problema da AGMQA para G. Considere agora que d_i, $1 \leq i \leq n$, é o grau do vértice v_i contabilizando somente as arestas presentes na solução. Note que um dado vértice v_i contribui com $d_i(d_i - 1)$ pares de arestas adjacentes (que compartilham v_i) na árvore e nenhum par de vértices contribui com o mesmo par de arestas. Dessa forma, podemos calcular o custo de t,

$$z(t) = \sum_{i=1}^{n} d_i(d_i - 1) \tag{1}$$

Pelos Lemas 1 e 2, podemos notar também que, em t,

$$\sum_{i=1}^{n} d_i = 2m = 2(n-1) \tag{2}$$

Agora, assuma, sem perda de generalidade, que, em nossa solução t, $d_1 = d_n = 1$ (no mínimo dois vértices em qualquer árvore possuem grau igual a 1). Então, a partir da equação 2,

$$\sum_{i=2}^{n-1} (d_i) + d_1 + d_n = \sum_{i=2}^{n-1} (d_i) + 2 = 2(n-1) \tag{3}$$

E, logo:

$$\sum_{i=2}^{n-1} d_i = 2(n-2) \tag{4}$$

Semelhantemente, a partir da equação 1,

$$z(t) = \sum_{i=2}^{n-1} (d_i(d_i-1)) + 1(1-1) + 1(1-1) = \sum_{i=2}^{n-1} d_i(d_i-1) \tag{5}$$

Agora, note que, no domínio $\{d_2, ..., d_{n-1}\}$ limitado por $\sum_{i=2}^{n-1} d_i = 2(n-2)$ e $d_i \geq 1$, para todo d_i (limites dados pela equação 4 e pela definição de árvore geradora, respectivamente), a função $z(x)$ é estritamente convexa.[1] Assim, $z(x)$ apresenta único mínimo quando os graus da árvore x são $d_2 = ... = d_{n-1} = 2$ e $d_1 = d_n = 1$, com valor $2(n-2)$.

Caso G permita uma configuração ótima para $d_2, ..., d_{n-1}$ de modo que, para nossa solução t, $z(t) = 2(n-2)$, então t será um caminho hamiltoniano (os vértices 1 e n são os extremos do caminho enquanto os vértices $2, ..., n-1$ são os nós internos). Caso contrário, t continuará sendo uma AGMQA, porém não configurará um caminho hamiltoniano.

□

4 Algortimo Heurístico

Foi implementado um dos três algoritmos heurísticos propostos em [1]. O algoritmo calcula um novo custo para a aresta $l'(x)$ em função de seus custos

[1] Não foi apresentada uma justificativa para tal na demonstração original. Esse é um assunto abordado em materiais de otimização convexa. `http://www.ifp.illinois.edu/~angelia/L3_convfunc.pdf` apresenta uma introdução.

quadráticos. Então, é usado o algoritmo de Kruskal [5] para achar a AGM do grafo considerando o custo apresentado. Definimos

$$l'(e) = \frac{\sum_{e' \in E} q(e, e')}{m - 1}$$

Assim, $l'(e)$ é a média dos custos quadráticos de e. O custo considerado para e na ordenação do algortimo Kruskal é então $l(e) + (n-2)l'(e)$. A constante $(n-2)$ simula então o custo quadrático para outras arestas formando uma árvore.

4.1 Pseudoalgoritmo

Algoritmo 1 Heurística

```
1: Leia o grafo G
2: Leia os custos lineares das arestas, l(x)
3: Leia os custos quadráticos dos pares de arestas, q(x)
4: for cada aresta e ∈ E do
5:     acumulado = 0
6:     for cada aresta e' ∈ E do
7:         acumulado += q(e, e')
8:     end for
9:     Defina l'(e) = acumulado/(m−1)
10: end for
11: Ordene E de forma crescente usando l(x) + (n − 2)l'(x) como critério
12: Kruskal(V, E)
```

Não foi descrito o funcionamento da função Kruskal, visto que esse já é um algoritmo muito conhecido na literatura. Foi assumido que o algoritmo tem como entrada uma lista de vértices e arestas, sendo a última já ordenada em função do critério em questão. Então, será gerada uma AGM iterando sobre as arestas seguindo a ordem passada.

5 Algoritmos exatos desenvolvidos

Foram desenvolvidos dois algoritmos exatos para o problema utilizando diferentes técnicas: um *backtrack* e outro *branch-and-bound*. Em ambos algoritmos, cada nó (estado) processado indica se uma aresta será considerada ou não. Dessa forma, cada estado está em função de três conjuntos X, Y, Z representando uma partição de E, onde

- X: nós que farão parte da árvore
- Y: nós que não farão parte da árvore
- Z: nós que se sabe ainda se farão parte da árvore

Assim, ao entrarmos num estado, uma aresta de Z será adicionada Y e, caso sua adição não gere um ciclo na árvore (esse passo é similar a [5]), a X também. Toda vez em que é acessado um novo estado, é checado se foram adicionadas $n - 1$ arestas. Como só são adicionadas arestas caso não seja formado um ciclo, temos então que uma árvore foi formada. O custo dessa árvore é então recuperado e considerado como solução. Caso a condição descrita não tenha sido satisfeita e não temos mais arestas para considerar, o estado corrente não consegue formar uma árvore. Em ambos algoritmos, também é checado se a quantidade de arestas ainda a serem processadas é suficiente para formar uma árvore.

5.1 Algoritmo *backtrack*

O algoritmo *backtrack* desenvolvido pode ser descrito no seguinte pseudocódigo.

Algoritmo 2 Backtrack

```
1: inseridas (vetor com arestas inseridas)
2: function BACKTRACK(aresta e, custo)
3:     se foram inseridas n arestas, retorne custo acumulado
4:     se foram processadas m arestas, retorne INF
5:     se m - inseridas < n - 1 - inseridas, retorne INF
6:     u, v ← vértices da aresta
7:     if adicionar a aresta não forma ciclo then
8:         adicione e a inseridas
9:         acumulado = custo linear de e
10:        for cada aresta e' já inserida do
11:            acumulado += q(e, e')
12:        end for
13:        resposta = min(resposta, BACKTRACK(próx. aresta, cost + acumulado)
14:        remova a aresta
15:    end if
16:    resposta = min(resposta, BACKTRACK(próx. aresta, cost)
17: end function
```

Podemos ver que sua implementação é de fato simples. Uma análise de complexidade local indica que o custo de tempo para cada chamada é de $O(m)$. Para checagem se a adição de uma aresta gera ciclo, foi utilizada a estrutura *Union-Find with rollback* [3], que permite operações de *find* e *join* em $O(log\ n)$ e *undo* em $O(1)$.

5.2 Algoritmo *branch-and-bound*

O algoritmo *branch-and-bound* desenvolvido é inspirado no apresentado em [1]. A função de *lower-bound* foi implementada exatamente como sugerida no artigo e então não há necessidade de ser descrita nesse documento.

Diferente da implementação sugerida em [1], sempre é acessado o nó com menor *lower-bound* da fila de prioridade. Foi utilizada uma *heap* binária para implementação da fila. Os estados podem ser descritos como uma estrutura com os seguintes atributos

- Próxima aresta a ser visitada
- Custo acumulado
- *Bitmask* indicando quais arestas já foram visitadas
- *Bitmask* indicando quais arestas estão inseridas na árvore corrente
- Quantas arestas já estão na árvore

Note que a quantidade de arestas já inseridas pode ser calculada a partir das *bitmasks*, mas foi optado por criar uma variável para armazenar seu valor a fim de poupar custo computacional. Abaixo, o pseudocódigo da função implementada.

Algoritmo 3 Branch and bound

1: ub ← INF
2: Q: fila de prioridade
3: insira o estado inicial a Q
4: **while** Q não está vazia **do**
5: **se** o custo do estado atual > *upperbound*, desconsidere esse estado
6: **if** foram inseridas $n - 1$ arestas na arvore **then**
7: *upperbound* ← min(*upperbond*, custo acumulado do estado)
8: **end if**
9: **se** foram consideradas todas as arestas, desconsidere o estado
10: **se** m - inseridas < n - 1 - inseridas, **retorne** INF
11: crie o estado em que a aresta atual não é considerada e calcule seu *lowerbound*
12: se *lowerbound* calculado < *upperbound*, insira-o em Q
13: **if** adicionar a aresta atual não gera ciclos **then**
14: crie o estado em que a aresta atual é considerada e calcule seu *lowerbound*
15: **se** lowerbound calculado < *upperbound*, insira-o em Q
16: **end if**
17: **end while**

Uma análise de complexidade local indica que cada passo do ciclo tem custo temporal $O(m + m(m\ log\ m))$, devido ao custo de calcular a contribuição ao custo quadrático da aresta a ser adicionada ($O(m)$) e o custo do cálculo do *lowerbound* ($O(m(m\ log\ m))$).

6 Banco de instâncias

O banco de instâncias a ser utilizado será o mesmo que em [2] e [4], podendo ser acessado em `http://homes.di.unimi.it/cordone/research/qmst.html`. As instâncias variam em quantidade de nós n (de 10 a 50 vértices) e em relação à densidade das arestas d (33%, 66% e 100%). Os custos lineares C e quadráticos Q são gerados randomicamente num intervalo de [1;10] (indicado por 10) ou [1;100] (indicado por 100).

6.1 Comparação de resultados

O computador utilizado para realização dos testes consiste em um Desktop rodando Arch Linux x86_64, utilizando o kernel 5.3.1-arch1-1-ARCH, com

um processador AMD Ryzen 7 3700X à 3.6GHz, uma placa de video AMD ATI Radeon RX Vega 56 e 16GB de Memória RAM DDR4 à 3200MHz.

6.1.1 Algoritmo heurístico

Foram comparados resultados obtidos pela heurística desenvolvida (NR) com os melhores resultados na literatura (NR) para instâncias com $n \in \{10, 15, 20\}$.

Como esperado, nossa heurística não obteve resultados próximos aos melhores na literatura. Porém, podemos notar que o algoritmo desenvolvido acaba obtendo melhores resultados quando a densidade do grafo é menor.

6.1.2 Algoritmos exatos

Para simplicidade de informação no gráfico, optamos por renomear alguns dos casos testes que utilizamos para comparação de resultados. Para isso, adaptamos o nome dos casos para o seguinte padrão: `[numero de arestas]a[numero do caso]`. Por exemplo, `45a1` representa o primeiro caso teste com 45 arestas.

Como pode ser visto na Figura 1, os resultados seguem o esperado. Como o algoritmo *branch and bound* poupa o cálculo de alguns nós, seu tempo de execução foi sempre menor que o backtrack, mesmo com o custo de cálculo de lowerbound para cada estado inserido na fila de prioridade.

A diferença entre a quantidade de estados visitados (Figura 2), obtida a partir de um contador, reafirma a superioridade do algoritmo branch-and-bound, visto que esse abre menos nós que o bracktrack (que abre todos os possíveis nós).

7 Conclusão e trabalhos futuros

Nesse trabalho foi apresentado o problema da AGMQ e algumas de suas variantes, junto da demonstração que o problema é NP-Completo. Trabalhos relacionados e algoritmos heurísticos e exatos também foram apontados e comentados.

Uma das heurísticas apresentadas em [1] foi desenvolvida e testada no banco de instâncias apresentado por [2]. Como visto em 6.1.1, os resultados dessa pouco se aproximam dos melhores resultados na literatura. Também foram implementados dois algoritmos exatos, um utilizando a técnica de *backtracking* e outro de *branch and bound*, o último utilizando a função de *lower-bound* apresentada em [1].

n	d	C	Q	MR	NR
10	33	10	10	350	363
10	33	10	100	3122	3122
10	33	100	10	646	702
10	33	100	100	3486	3129
10	67	10	10	255	371
10	67	10	100	2042	3151
10	67	100	10	488	527
10	67	100	100	2404	3717
10	100	10	10	238	357
10	100	10	100	1815	3145
10	100	100	10	426	477
10	100	100	100	2197	3639
15	33	10	10	745	943
15	33	10	100	6539	7721
15	33	100	10	1236	1289
15	33	100	100	7245	9383
15	67	10	10	659	821
15	67	10	100	5573	6821
15	67	100	10	966	1029
15	67	100	100	6188	7963
15	100	10	10	620	972
15	100	10	100	5184	8303
15	100	100	10	975	1216
15	100	100	100	5879	9853
20	33	10	10	1379	1769
20	33	10	100	12425	15137
20	33	100	10	1972	2248
20	33	100	100	13288	18095
20	67	10	10	1252	1770
20	67	10	100	10893	14108
20	67	100	10	1792	2165
20	67	100	100	11893	17940
20	100	10	10	1174	1637
20	100	10	100	10215	15690
20	100	100	10	1544	1825
20	100	100	100	11101	16545

Tabela 1: Comparação entre resultados obtidos pela heurística e resultados ótimos

Referências

[1] Assad, A., and Xu, W. The quadratic minimum spanning tree pro-

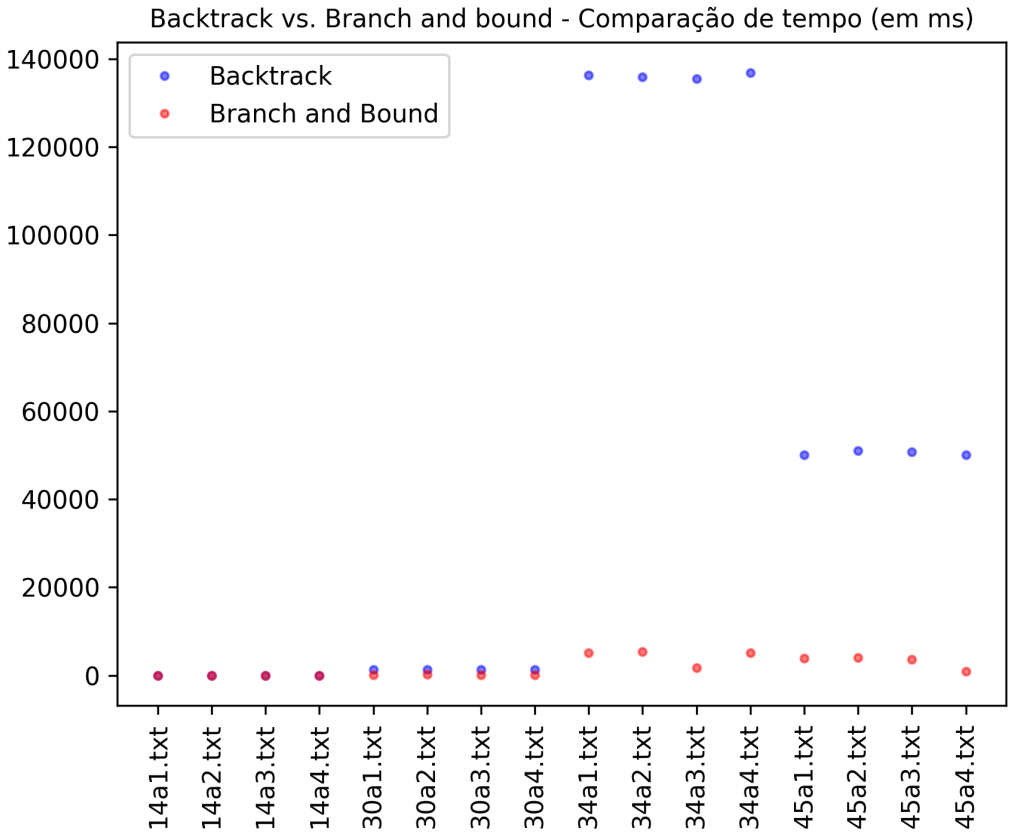

Figura 1: Comparação entre tempo de execução dos algoritmos exatos

blem. *Naval Research Logistics (NRL) 39*, 3 (1992), 399–417.

[2] Cordone, R., and Passeri, G. Solving the quadratic minimum spanning tree problem. *Applied Mathematics and Computation 218* (08 2012), 11597–11612.

[3] Galil, Z., and Italiano, G. F. Data structures and algorithms for disjoint set union problems. *ACM Computing Surveys (CSUR) 23*, 3 (1991), 319–344.

[4] Isabel, F. M. P. A transgenetic algorithm for the quadratic minimum spanning tree problem. B.S. thesis, Universidade Federal do Rio Grande do Norte, 2018.

[5] Kruskal, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society 7*, 1 (1956), 48–50.

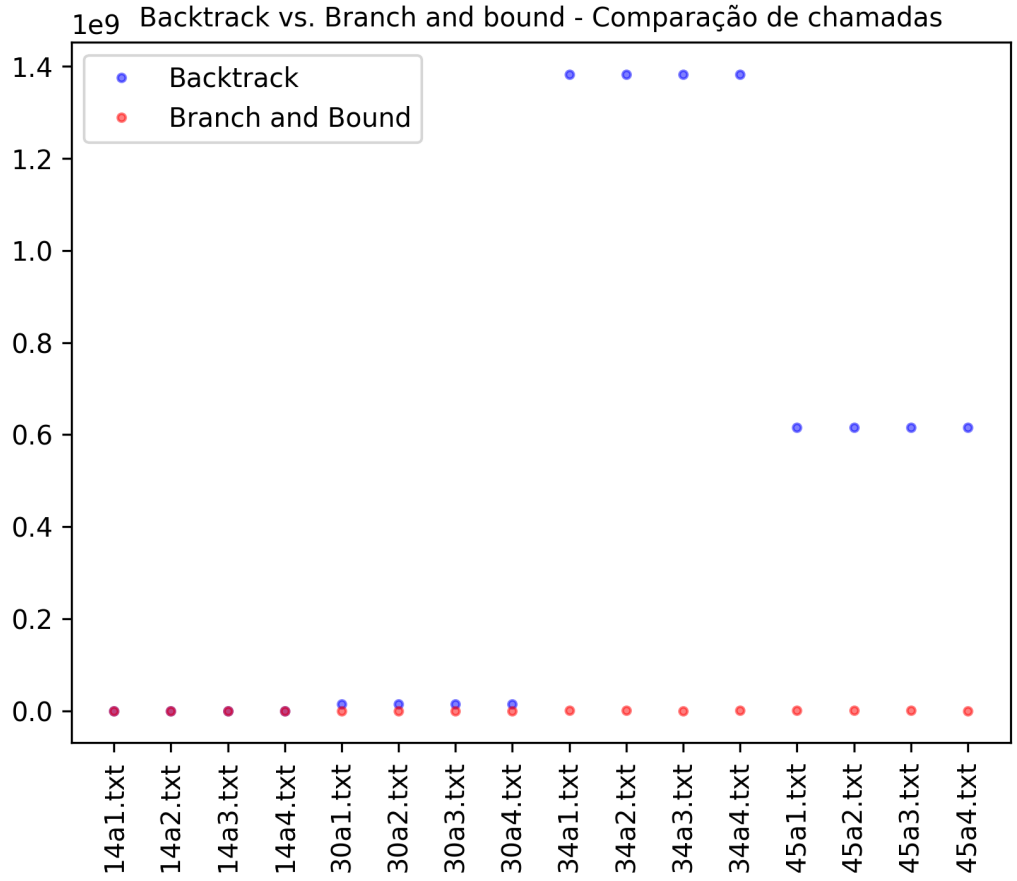

Figura 2: Comparação entre a quantidade de estados visitados pelos algoritmos exatos

[6] Lozano, M., Glover, F., García-Martínez, C., Rodríguez, F. J., and Martí, R. Tabu search with strategic oscillation for the quadratic minimum spanning tree. *IIE Transactions 46*, 4 (2014), 414–428.

[7] Maia, S. M. D. M. *O problema biobjetivo da árvore geradora quadrática em adjacência de arestas*. PhD thesis, Universidade Federal do Rio Grande do Norte, 2013.

[8] Palubeckis, G., Rubliauskas, D., and Targamadzė, A. Metaheuristic approaches for the quadratic minimum spanning tree problem. *Information Technology and Control 39* (01 2010), 257–268.

[9] Pereira, D. L., Gendreau, M., and da Cunha, A. S. Lower bounds and exact algorithms for the quadratic minimum spanning tree problem. *Computers & Operations Research 63* (2015), 149–160.

[10] Soak, S.-M., Corne, D. W., and Ahn, B.-H. The edge-window-decoder representation for tree-based problems. *IEEE Transactions on Evolutionary Computation 10*, 2 (2006), 124–144.

[11] Sundar, S., and Singh, A. A swarm intelligence approach to the quadratic minimum spanning tree problem. *Inf. Sci. 180*, 17 (Sept. 2010), 3182–3191.

[12] Zhout, G., and Gen, M. An effective genetic algorithm approach to the quadratic minimum spanning tree problem. *Computers & Operations Research 25*, 3 (1998), 229 – 237.