UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE SISTEMAS DE INFORMAÇÃO

FRANCISCO GLEYSON DA SILVA GOMES

HEURÍSTICAS PARA O PROBLEMA DA ÁRVORE DE CUSTO MÍNIMO COM K-ARESTAS

QUIXADÁ
2013

FRANCISCO GLEYSON DA SILVA GOMES

HEURÍSTICAS PARA O PROBLEMA DA ÁRVORE DE CUSTO MÍNIMO COM K-ARESTAS

Monografia apresentada ao Curso de Sistemas de Informação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial para obtenção do Titulo de Bacharel em Sistemas de Informação.

Orientador: Prof. Msc. Fábio Carlos Sousa Dias

QUIXADÁ

2013

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

G612h Gomes, Francisco Gleyson da Silva
Heurísticas para o problema da árvore de custo mínimo com k-arestas / Francisco Gleyson da Silva Gomes – 2013.
39 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2013.
Orientação: Prof. MSc. Fábio Carlos Sousa Dias
Área de concentração: Computação

1. Árvores (Teoria dos grafos) 2. Programação heurística 3. Algoritmos I. Título.

CDD 511.8

FRANCISCO GLEYSON DA SILVA GOMES

HEURÍSTICAS PARA O PROBLEMA DA ÁRVORE DE CUSTO MÍNIMO COM K-ARESTAS

Monografia apresentada ao Curso de Sistemas de Informação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial para obtenção do Titulo de Bacharel em Sistemas de Informação.

Aprovada em: ___/___/_____.

BANCA EXAMINADORA

_____

Prof. Msc. Fábio Carlos Sousa Dias (Orientador)
Universidade Federal do Ceará (UFC)

_____

Prof. Dr. Críston Pereira de Souza
Universidade Federal do Ceará (UFC)

_____

Prof. Msc. Wladimir Araújo Tavares
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Em primeiro lugar, agradeço a aquele, que permitiu tudo isso, ao longo de toda a minha vida, e, não somente nestes anos de estudo da graduação, mas também durante toda minha vida. Obrigado meu DEUS pela força e coragem durante toda a minha vida.

São muitas as pessoas que desejo agradecer, primeiro vou agradecer aqueles que me ajudaram em todos os níveis da minha vida, aos meus pais, Eginaldo e Ivanilda, ao meu irmão Gleiton, e ao meu Professor Davi.

Em seguida, gostaria de agradecer ao meu orientador, o professor Fábio, que me orientou por quase três anos, e mostrou o maravilhoso problema da Árvore de Custo Mínimo com k – Arestas, ou simplesmente k – ACM. Ensinou-me a escrever textos científicos melhores, e teve paciência na minha demora em entregar os resultados. Um grade amigo e um excelente orientador.

Também gostaria de agradecer aos seguintes professores: Jefferson, Samy, Lavor, Wladimir, Tânia, Ciro, Marcos Oliveira, Alberto, Jeandro, Aragão, Ricardo, Críston, Andréia, Camilo, Atslands, Diana, Marcos Dantas e Arthur. Também gostaria de agradecer aos funcionários Evalda e Hamilton.

Concluindo, gostaria de agradecer aos amigos que fiz durante a graduação dentro e fora da UFC Quixadá. Dentre os amigos do PET, não irei esquecer de Dodô, Fernando, André, Rainara, Ismaily, Virginia, Wellington, Egberto, Leonara, Lina, Mardson, Daniel, Marcelo, Junior, Salamão, Klyssia, Filipe e William. Dentre os outros amigos, não irei esquecer do apoio de Paulo Ramon, Humberto, Pedro Henrique, Carlos, Cícero e Verônica.

RESUMO

O problema da árvore de custo mínimo com *k* arestas consiste em encontrar, para um grafo conexo e não-orientado com custo nos vértices e nas arestas, uma componente de custo mínimo que seja uma árvore com exatamente *k* arestas. Neste trabalho, apresentamos a formulação matemática proposta por Fischetti et al. (1994) e cinco heurísticas desenvolvidas, baseadas nos algoritmos Prim (1957) e Kruskal (1956) e Dijkstra (1959). Uma das heurísticas apresentadas neste trabalho, mostrou-se competitiva, obtendo soluções muito próximas das encontradas pela Heurística Lagrangeana no conjunto *bb* proposto por Blum e Blesa (2005). Os resultados das nossas heurísticas foram comparados com os obtidos da Heurística Lagrangeana de Quintão et al. (2008) e (2010), para as instâncias do conjunto *g* proposto por Blesa e Xhafa (2000) e conjunto *bb* proposto por Blum e Blesa (2005). As heurísticas foram desenvolvidas para a versão *k*-ACM-CA. Comparando os resultados, analisamos que as heurísticas apresentadas neste trabalho, apresentam o tempo de execução para cada instância, superior nos dois conjuntos quando comparados com os obtidos na Heurística Lagrangeana de Quintão et al. (2008) e (2010). Os tempos de execução das heurísticas poderão ser melhorados ao aproveitar a Union-Find, ao invés de utilizar a busca em profundidade, além disso, as arestas poderão ser colapsadas, para que o Dijkstra (1959) seja aplicado no grafo colapsado.

Palavras-chave: Otimização. k-ACM. Heurísticas HKP. Heurística KHPD.

ABSTRACT

The problem of minimum cost tree with *k* edges consists in finding for a connected graph and not cost-oriented with the vertices and edges a minimum cost component that is a tree with exactly *k* edges. We present the mathematical formulation proposed by Fischetti et al. (1994) developed five heuristics, based on the Prim algorithm (1957) and Kruskal (1956) and Dijkstra (1959). One of the heuristics presented in this paper, was competitive, getting very close to the solutions found by the Lagrangian heuristic set bb proposed by Blum and Blesa (2005). The results of our heuristics were compared with those obtained from the heuristic Lagrangian Quintão et al. (2008) e (2010) for the instances of the set g proposed by Blesa and Xhafa (2000) and set bb proposed by Blum and Blesa (2005). The heuristics were developed for version k-ACM-CA. Comparing the results, we analyze the heuristics presented in this paper presents the runtime for each instance, the top two sets compared with those obtained in the Lagrangian Heuristic Quintão et al. (2008) e (2010).The execution times of the heuristic can be improved by leveraging Union-Find, instead of using the depth-first search in addition the edges may be collapsed so that the Dijkstra (1959) is implemented in the collapsed graph.

Keywords: Optimization. k-ACM. HKP Heuristics. Heuristic KHPD.

LISTA DE ILUSTRAÇÕES

LISTA DE GRÁFICOS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Problemas de otimização são problemas em que desejamos encontrar uma solução que seja a melhor possível dado um conjunto de restrições. Esses problemas são considerados de maximização ou minimização de acordo com a necessidade de maximizar ou minimizar o conjunto de soluções. Para a melhor solução existente, damos o nome de solução ótima.

Ultimamente, os problemas de otimização tem ganhado grande visibilidade e os estudos nesta área têm avançado muito, principalmente por parte das indústrias e grandes empresas que tem encontrado em seu cotidiano aplicações para este tipo de problema, EXAME (2006), Nemhauser (2007).

O nível de dificuldade para encontrar a solução ótima é considerável em problemas mais complexos. Podemos medir esse nível de dificuldade pela complexidade destes problemas e/ou pelo porte dos problemas. Problemas considerados de grande porte são problemas cuja base de dados é muito grande. Por isso, vêm sendo cada vez mais importante o uso de algoritmos mais eficientes que possam fornecer soluções em tempo hábil, tendo em vista que os recursos computacionais são limitados.

Nesse sentido, o uso de heurísticas torna-se cada vez mais importante porque são algoritmos que não garantem encontrar a solução ótima, mas que em geral fornecem soluções muito próximas da ótima, além de fornecer essas soluções em tempo computacionalmente aceitável. Na literatura, encontramos estudos relacionados a heurísticas para resolver problemas de otimização, podem ser elas baseadas em relaxações lagrangeanas ou relacionadas a algoritmos gulosos dentre outros.

Neste trabalho desenvolvemos heurísticas para um problema de otimização. Este problema está definindo sobre um grafo conexo e não orientado $G = (V, E)$, onde V é o conjunto de vértices e E o conjunto de arestas. Para cada aresta $e \in E$ temos um custo associado c_e, e para todo vértice $v \in V$ associamos um custo d_v. O Problema da Árvore de Custo Mínimo com k arestas, ou simplesmente k-ACM, consiste em encontrar um subgrafo de custo mínimo com k arestas que seja uma árvore. O custo de uma árvore $T = (V_T, E_T)$ é dado por $\sum d_v + \sum c_e$. Existem duas variações desse problema, uma quando apenas consideramos os custos das arestas, chamado de k-ACM-CA, e outra quando apenas consideramos os custos dos vértices, chamado de k-ACM-PV. Nos três casos, o problema é NP – Difícil, Fischetti et al. (1994).

Para alguns casos particulares o problema é polinomial. Para $k = n - 1$ o problema reduz-se a encontrar uma Árvore Geradora Mínima (AGM), e quando o grafo G for uma árvore, o problema pode ser resolvido utilizando um algoritmo de programação dinâmica, Fischetti et al. (1994) e Blum (2007). Já para valores pequenos de k, o problema pode ser resolvido por inspeção.

Existem várias aplicações reais para o problema. Uma delas consiste no problema de Arrendamento de Poços de Petróleo, Hamcher e Joernster (1993). Nesta aplicação, uma empresa obtém do governo o direito de exploração de campos de petróleo por um determinado período. Ao final deste período a empresa deve devolver alguns campos, deste que exista um caminho entre esses campos. Com isso, a empresa tem que escolher os k campos conectados de menor valor comercial para ser devolvido ao governo. Também encontramos aplicações para o k-ACM em Telecomunicações com Garg (1997), Roteamento com Cheung (1994) e Mineração com Philportt (1997).

A dificuldade do problema pode ser medida pela pouca quantidade de algoritmos exatos existentes na literatura, sendo a maioria, heurísticas ou algoritmos aproximativos. Alguns autores se mostram céticos quanto às dimensões das instâncias que podem ser resolvidas com garantia de otimalidade em tempo polinomial.

Neste trabalho desenvolvemos heurísticas para o problema, heurísticas baseadas nos algoritmos do Prim (1957), Kruskal (1956) e Dijkstra (1959).

Neste trabalho propomos algumas heurísticas para o k-ACM-CA. O desenvolvimento de heurísticas tem por finalidade encontrar boas soluções com pequeno custo de tempo. Uma comparação com os melhores resultados da literatura é feito. Queremos que nossas heurísticas sejam competitivas com esses resultados, seja pelo valor da sua solução ou pelo consumo de tempo.

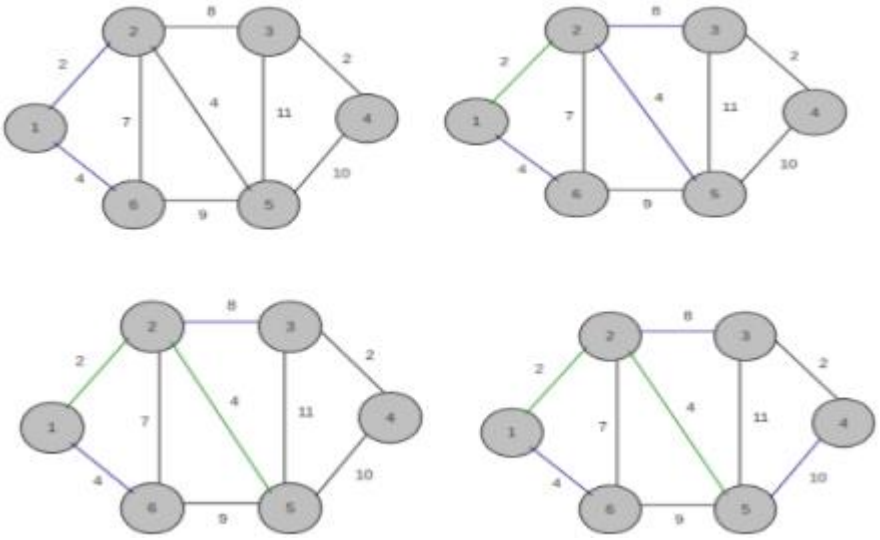
Nosso trabalho está voltado para o k-ACM-CA, mas todo resultado obtido para essa versão pode ser aplicado para o k-ACM através da transformação do grafo de entrada, conforme mostrado por Quintão et al. (2008) e (2010).

No desenvolvimento das nossas heurísticas utilizamos as características essenciais dos algoritmos Prim (1957), Kruskal (1956) e Dijkstra (1959). O algoritmo Prim (1957) garante que a cada iteração aumenta uma árvore única. A Figura 1 ilustra o resultado do algoritmo Prim (1957), para adicionar 2 arestas. O algoritmo Kruskal (1956), a cada iteração adiciona uma aresta de custo mínimo, desde que, a inserção dessa aresta não gere ciclo. Além disso, o Kruskal (1956) tem a característica que depois de inserida várias arestas na floresta, são criadas componentes que são sub-árvores de custo mínimo. A Figura 2 ilustra o resultado

do algoritmo Kruskal (1956), depois de adicionar 4 arestas na floresta. A característica importante do Dijkstra (1959) é que podemos verificar se uma dessas componentes geradas pelo o Kruskal tem custo pequeno para ser unida a outra componente. A característica do Prim (1957), em aumentar uma única árvore poderá ser usada, por exemplo, quando precisarmos adicionar mais arestas numa componente da floresta encontrada pelo o Kruskal (1956).

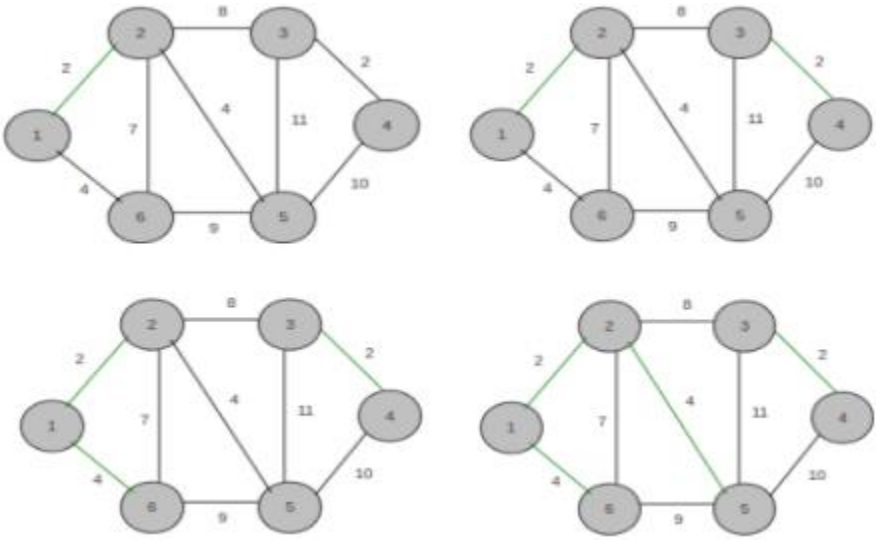
Realizamos um comparativo entre os resultados obtidos neste trabalho com os resultados de Ehrgott et al. (1997) e de Quintão et al. (2008) e (2010).

Figura 1 - Ilustra a execução do algoritmo Prim (1957), depois de adicionar 2 arestas.



Fonte: Elaborada pelos autores.

Figura 2 - Ilustra a execução do algoritmo Kruskal (1956), depois de adicionar 4 arestas.



Fonte: Elaborada pelos autores.

2 REVISÃO BIBLIOGRÁFICA

Na literatura encontramos uma diversidade de heurísticas para o problema. A grande maioria é baseada nos algoritmos Prim (1957) e Kruskal (1956).

No algoritmo de Prim (1957), a árvore inicial $H(U, A)$ tem o conjunto A de aresta inicialmente vazio e um conjunto U de vértice contendo um vértice raíz r. A escolha dessa raiz é feita de forma arbitrária. Em cada iteração o algoritmo seleciona no corte uma aresta de custo mínimo (u, v), e adiciona essa aresta à árvore H, assim começa uma nova iteração até que H seja uma AGM, Cormen et al. (2002).

No algoritmo de Kruskal (1956), o subgrafo $H(U, A)$ é uma floresta, em que A inicialmente é vazio. Em cada iteração, o algoritmo escolhe uma aresta (u, v) de menor peso que liga os vértices de duas componentes distintas de $H(U, A)$, desde que não gere ciclo. Assim o algoritmo adiciona (u, v) em A e começa uma nova iteração até que H seja uma AGM.

Uma heurística simples para encontrar uma solução viável do problema k-ACM-CA consiste em utilizar o algoritmo Prim até que ele adicione k arestas na componente. Chamaremos essa heurística de Prim-ACM.

Ehrgott et al. (1997), fornecem várias heurísticas para a versão k-ACM-CA. Uma em destaque é baseada na ideia de utilização do Prim-ACM para encontrar uma solução do problema, sendo que o Prim-ACM será executado n vezes, uma para cada vértice como sendo o vértice inicial do Prim-ACM, e no final retorna a árvore de menor custo entre todas encontradas.

Fischetti et al. (1994), fornecem a primeira formulação matemática para o problema. Eles definem duas variáveis binárias para representar a solução do problema. Seja $x_e, \forall e \in E$, igual a 1, se a aresta e for escolhida para pertencer à solução, e 0 caso contrário. Da mesma forma, seja $y_v, \forall v \in V$, igual a 1, se o vértice v for escolhido para pertencer a solução, e 0 caso contrário.

Figura 3 - Representa a formulação matemática proposta por Fischetti et al. (1994), para a versão k-ACM.

$$\text{(k - ACM)} \quad \min \sum_{e \in E} c_e x_e + \sum_{v \in V} d_v y_v \qquad (1)$$

$$\text{Sujeito a :} \quad \sum_{e \in E} x_e = k \qquad (2)$$

$$\sum_{v \in V} y_v = k + 1 \qquad (3)$$

$$\sum_{e \in E(S)} x_e \leq \sum_{v \in S \setminus j} y_v, \forall S \subseteq V, |S| \geq 2, \forall j \in S \qquad (4)$$

$$x_e \in \{0,1\}, \forall e \in E \text{ e } y_v \in \{0,1\}, \forall v \in V \qquad (5)$$

Fonte: Adaptada de Fischetti et al. (1994).

A função objetivo (1) minimiza os custos das arestas e dos vértices escolhidos. As restrições (2) e (3) garantem que *k* arestas e *k* + 1vértices serão selecionados, respectivamente. As restrições (4) garantem que a solução não terá ciclos. Elas são baseadas em desigualdades de eliminação de ciclos, logo aparecem em número exponencial de restrição. Reparem que quando consideramos as versões *k*-ACM-CA e *k*-ACM-PV devemos modificar a função objetivo considerando apenas os custos das arestas e dos vértices, respectivamente.

Quintão et al. (2008) e (2010) apresentaram uma heurística lagrangeana baseada na relaxação obtida da reformulação de Multi-Fluxo apresentada por eles. Essa heurística lagrangeana mostrou-se melhor que muitas heurísticas existentes da literatura. Neste mesmo trabalho os autores apresentaram duas formulações baseadas em uma transformação do grafo de entrada. O *k*-ACM é definido para um grafo $G = (V, E)$ não direcionado e ponderado nas arestas e vértices, e as duas formulações fazem uso de um grafo $D = (V, A)$ direcionado e ponderado apenas nas arestas obtido de *G* após algumas operações. As operações se resumem em adicionar novas arestas ao grafo e na transformação das arestas não direcionadas por duas arestas direcionadas com a incorporação dos custos dos vértices nos custos das arestas, Quintão et al. (2008). Uma das reformulações baseia-se em restrições de Multi-Fluxo para descrever árvore, e a outra foi baseada nas desigualdades de Miller, Tucker e Zemlir (1960).

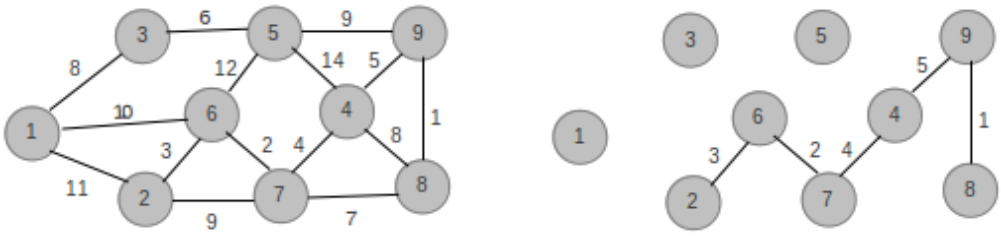
3 HEURÍSTICAS PROPOSTAS

Desenvolvemos cinco heurísticas para o problema. São heurísticas baseadas nos algoritmos do Prim (1957), Kruskal (1956) e Dijkstra (1959), onde buscamos utilizar as melhores características encontradas de cada algoritmo para ser aplicado em nosso problema.

3.1 Heurística de Kruskal-Prim (HKP)

A heurística de Kruskal-Prim (HKP) poderá ter duas fases. Na primeira fase aplicamos o Kruskal (1956) e, em cada iteração, verificamos se a última componente que recebeu aresta é uma solução viável do problema, ou seja, possui k arestas. Neste caso, a heurística termina retornando essa componente como solução. Na Figura 4 mostramos um grafo que será dado como entrada e a componente resultante encontrada na primeira etapa do HKP, para o caso de k = 5.

Figura 4 - Ilustra um grafo dado como entrada e a componente resultante encontrada na primeira etapa do HKP, para o caso de k = 5.



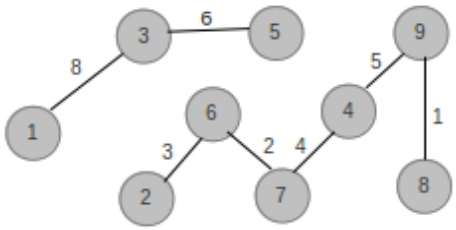
Fonte: Elaborada pelos autores.

Se após adicionarmos *n* - 2 arestas não encontrarmos uma solução, então iniciamos a segunda fase. A segunda fase inicia com as duas componentes da primeira fase e, em cada uma delas, aplicamos o algoritmo Prim-ACM para gerar uma solução.

Inicializamos o Prim-ACM com uma das componentes, e a cada iteração aumentamos esta componente com a aresta de menor custo no corte.

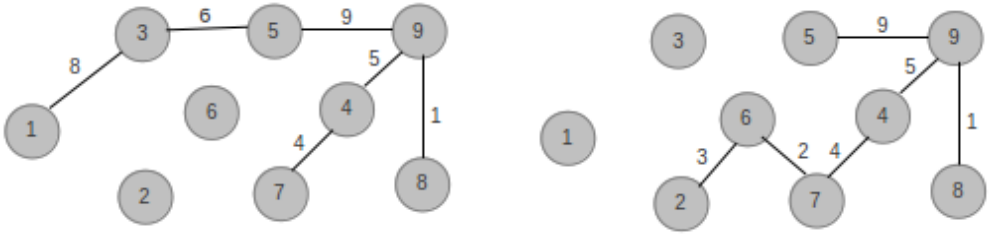
Na Figura 5 exemplificamos o resultado da primeira fase do algoritmo HKP sobre o grafo da Figura 4.

Figura 5 - Componentes formadas depois de serem adicionadas n – 2 arestas na primeira etapa da heurística HKP para k = 6.



Fonte: Elaborada pelos autores.

Figura 6 - Mostramos os resultados da aplicação do Prim-ACM em cada uma das componentes da Figura anterior.



Fonte: Elaborada pelos autores.

Comparamos as duas soluções encontradas, uma em cada componente, e devolvemos a de menor custo. A ideia dessa heurística é tentar usar o melhor dos algoritmos Kruskal (1956) e Prim (1957). A seguir apresentamos o pseudocódigo da HKP.

Figura 7 - Ilustra o pseudocódigo da HKP.

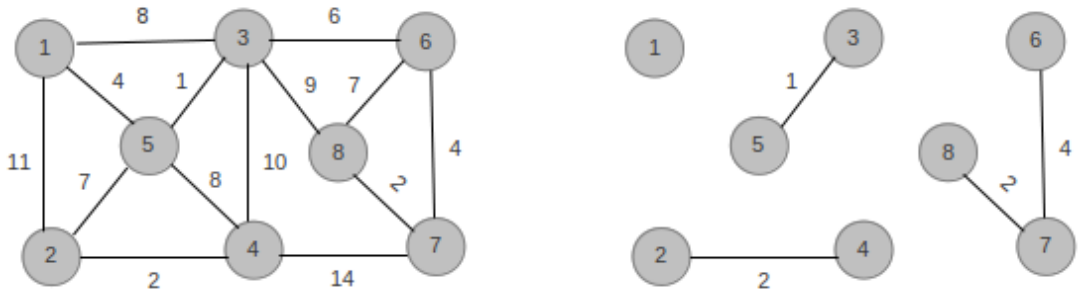
```
Estrutura componente

1   int n; //Número de vértices.
2   int m; //Número de arestas.
3   int *grau; //Grau de cada vértice dessa componente.
4   int *s; //Vértices dessa componente.
5   int *f; //Folhas dessa componente.
6   int *b; //Vértices borda da componente.
7   int nf; //Número de folhas.
8   int nvb; //Número de vértices borda.
9   int r; //Raiz dessa componente.
10  double w; //Custo dessa componente.
11  LsAdj *Adj; //Lista de Adjacência.
12  Aresta *arestas; //Arestas da componente.



MST_PRIM(c, k)

1  Inicializar a componente c
2  Para i = c.m até k
3       Expandindo a componente c com a menor aresta do corte E(u, v)
4       custo = incrementar o custo de c com o valor de E(u, v)
5       Selecionado as arestas que irão pertencer ao novo corte
6  retorna custo


 HKP(k, n, m)

 1  QuickSort(arestasOrd, m)
 2  Para i = 1 até m e c = 1 até n - 2
 3       u = arestasOrd[i].u
 4       v = arestasOrd[i].v
 5       Se FIND_SET(u) != FIND_SET(v)

 6           adicionar a aresta E(u, v) na floresta H
 7           pai = pegar o pai da componente que contém E(u, v) em H
 8           Se c >= k
 9              c1 = pegar componente de raíz pai
 10             Se c1.m == k
 11                retorna c1.w
 12 g = pegar os representantes das componentes da floresta H
 13 Para u = 1 até w faça
 15      c2 = pegar componente de raíz g[u] da floresta H
 17      S = MST_PRIM(c2, k)
 18      Se S < custo
 19         custo = S
 20 retorna custo
```

Fonte: Elaborada pelos autores.

3.2 Heurística de Kruskal-Prim 2 (HKP-2)

Na heurística Kruskal-Prim 2 (HKP-2) também teremos duas fases. Na primeira fase aplicamos o Kruskal (1956) e, após adicionarmos *k* arestas, teremos no máximo *k* componentes com pelo menos uma aresta, conforme o exemplo da Figura 8, para $k = 4$.

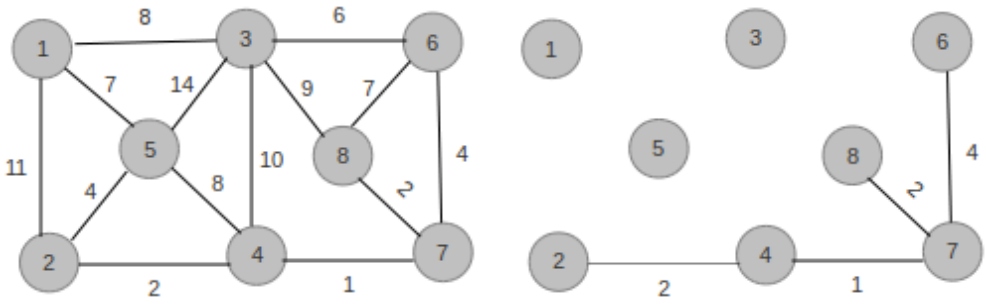
Figura 8 - Grafo dado como entrada e as componentes, resultado da inserção de *k* arestas para $k = 4$.



Fonte: Elaborada pelos autores.

Se adição da última aresta formou uma componente com exatamente *k* arestas, retornamos o peso dessa componente como nossa solução. A Figura 9 ilustra o caso em que a solução encontrada na primeira fase da heurística HKP-2 fornece a solução do problema k-ACM-CA.

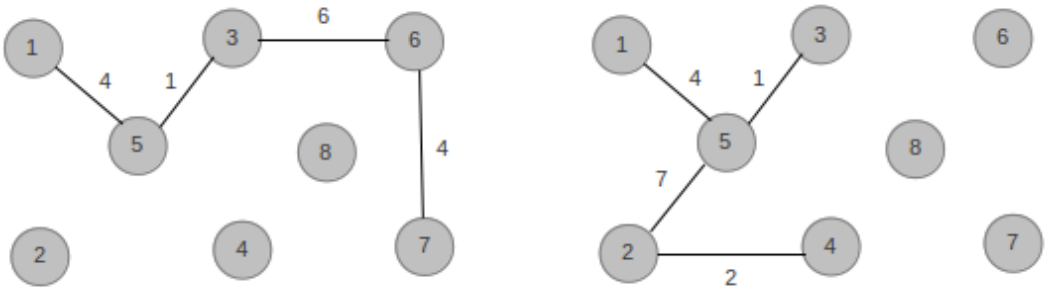
Figura 9 - Grafo dado como entrada e a componente resultante da inserção de *k* arestas retornada como solução.



Fonte: Elaborada pelos autores.

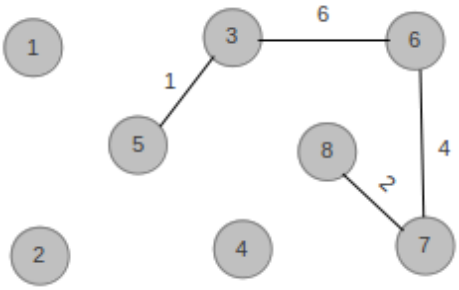
Caso a solução não tenha sido encontrada, iniciamos a segunda fase aplicando o Prim-ACM em cada componente, semelhante à heurística HKP. A Figura 10 ilustra um exemplo do algoritmo Prim-ACM sobre as componentes da Figura 8.

Figura 10 - Resultado da primeira e segunda componente após aplicar o Prim-ACM.



Fonte: Elaborada pelos autores.

Figura 11 - Resultado da terceira componente após aplicar o Prim-ACM.



Fonte: Elaborada pelos autores.

A ideia dessa heurística é procurar soluções em espaços de vizinhança diferentes. Pois ela utiliza vários pontos de partida. A seguir o pseudocódigo da HKP-2.

Figura 12 - Ilustra o pseudocódigo da HKP-2.

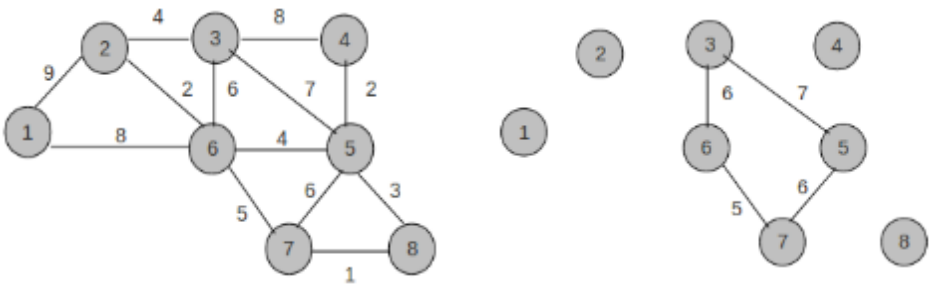
```
KRUSKAL-MOD()

1  QuickSort(arestasOrd, m)
2  Para i = 1 até m e c = 1 até k
3       u = arestasOrd[i].u
4       v = arestasOrd[i].v
5       Se FIND_SET(u) != FIND_SET(v)
6          adicionar a aresta E(u, v) na floresta H
7          pai = pegar o pai da componente que contém E(u, v) em H
8          Se c == k
9             c1 = pegar componente de raíz pai
10            Se c1.m == k
11               retorna c1.w
12 g = pegar os representantes das componentes da floresta H
13 Para u = 1 até w faça
15       c2 = pegar componente de raíz g[u] da floresta H
17       S = MST_PRIM(c2,k)
18       Se S < custo
19          custo = S
20 retorna custo
```

Fonte: Elaborada pelos autores.

3.3 Heurística de Kruskal-Prim 3 (HKP-3)

Na heurística Kruskal-Prim 3 (HKP-3), aplicamos o Kruskal (1956) para encontrar a AGM, e as arestas que vão sendo descartadas pelo Kruskal neste processo vão sendo armazenadas na floresta *H*, deste que não gere ciclos. A Figura 13 ilustra um caso em que uma aresta descartada pelo Kruskal forma ciclo na floresta H.

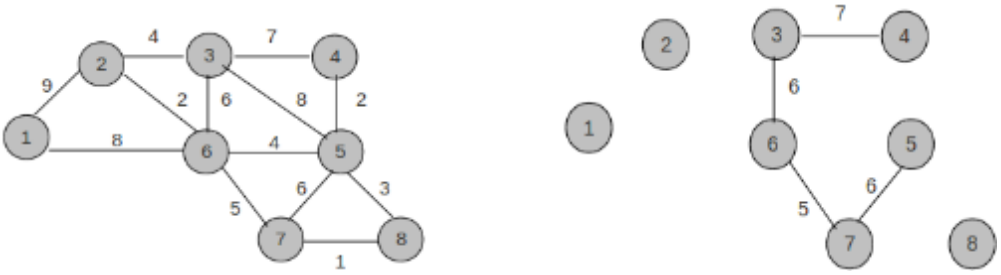
Figura 13 - Grafo para o exemplo em que a aresta descartada da floresta *F* e adicionada na floresta *H* gera um ciclo.



Fonte: Elaborada pelos autores.

Depois de inseridas *k* arestas em *H*, verificamos se *H* forma uma solução para o *k*-ACM-CA. A Figura 14 ilustra o caso em que as k arestas descartadas pelo Kruskal formam uma componente conexa sem ciclos, ou seja, uma solução válida para o k-ACM-CA.

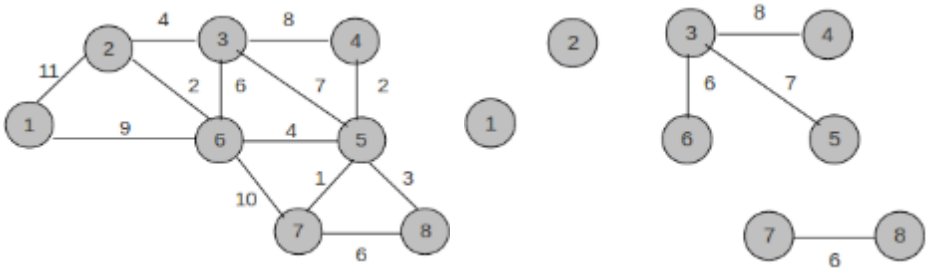
Figura 14 - Grafo dado como entrada para o exemplo em que as arestas descartadas e adicionadas na floresta *H* formam uma componente com exatas *k* arestas.



Fonte: Elaborada pelos autores.

Se após encontrar a AGM a solução ainda não foi encontrada, então para cada componente de *H*, aplicamos o Prim-ACM e, retornamos a componente de custo mínimo. A Figura 15 ilustra o caso em que as arestas descartadas na fase Kruskal (1956) e, adicionadas na floresta *H* formam componentes com menos de *k* arestas.

Figura 15 - Caso em que as arestas descartadas pelo o Kruskal (1956) e, adicionada na floresta *H* formam componentes com menos de *k* arestas, para *k* = 4.



Fonte: Elaborada pelos autores.

O argumento dessa heurística está no fato das arestas descartadas pelo Kruskal possuírem valores de custo pequenos comparadas com outras arestas na árvore geradora mínima, ou seja, estamos usando arestas de custo baixo para produzir novos pontos de partida para o Prim-ACM. A seguir apresentamos o pseudocódigo da HKP-3.

Figura 16 - Ilustra o pseudocódigo da HKP-3.

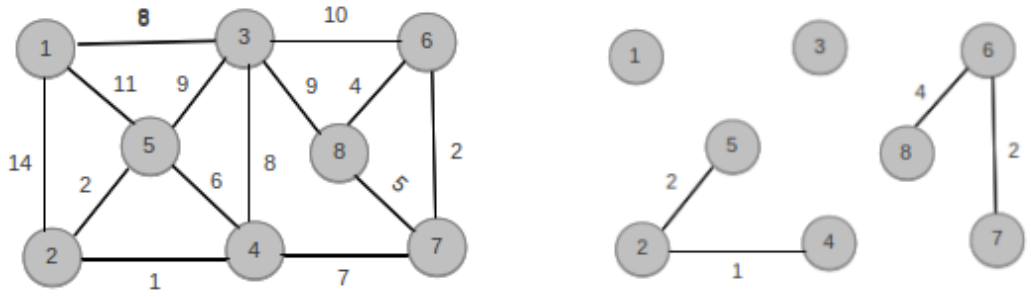
```
HKP-3(k, n, m)

1  QuickSort(arestasOrd, m)
2  Para i = 1 até m e c = 1 até n - 1
3       u = arestasOrd[i].u
4       v = arestasOrd[i].v
5       Se aresta E(u, v) não gerar ciclo ao ser adiconada em H
6          adicionar a aresta E(u, v) na floresta H
7          incrementa c
8       Senão
9          Se aresta E(u, v) não gerar ciclo ao ser adiconada em F
10            adicionar a aresta E(u, v) na floresta F
11            pai = pegar o pai da componente que contém E(u, v) em F
12            c1 = pegar componente de raíz pai
13            Se c1.m == k
14               retorna c1.w
15 g = pegar os representantes das componentes da floresta F
16 Para u = 1 até w faça
17      c2 = pegar componente de raíz g[u] da floresta F
18      S = MST_PRIM(c2, k)
19      Se S < custo
20         custo = S
21 retorna custo
```

Fonte: Elaborada pelos autores.

3.4 Heurística de Kruskal-Prim 4 (HKP-4)

Na heurística Kruskal-Prim 4 (HKP-4), aplicamos o algoritmo de Kruskal (1956) no grafo até que sejam adicionadas *k* arestas na floresta H. Depois de adicionadas essas *k* arestas teremos no máximo *k* componentes na floresta H. A Figura 17 ilustra o caso em que as arestas adicionadas na floresta H formam componentes com menos de *k* arestas, quando *k* = 4.

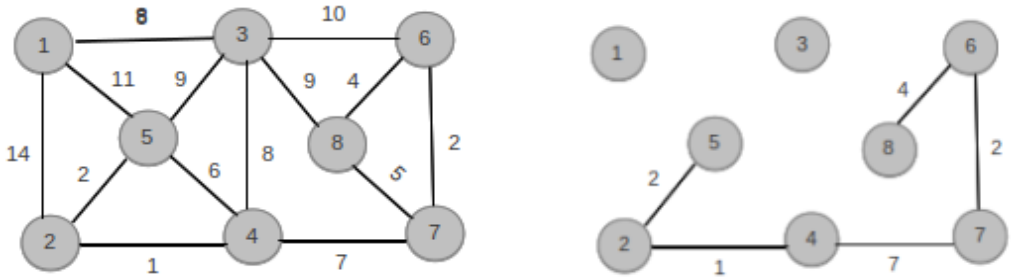
Figura 17 - Grafo dado como entrada e as componentes, resultado da inserção de *k* arestas.



Fonte: Elaborada pelos autores.

Aplicamos então uma busca em profundidade para verificar que H forma uma única componente conexa. A Figura 18 ilustra o caso em que a solução é encontrada nesta etapa, para $k = 5$.

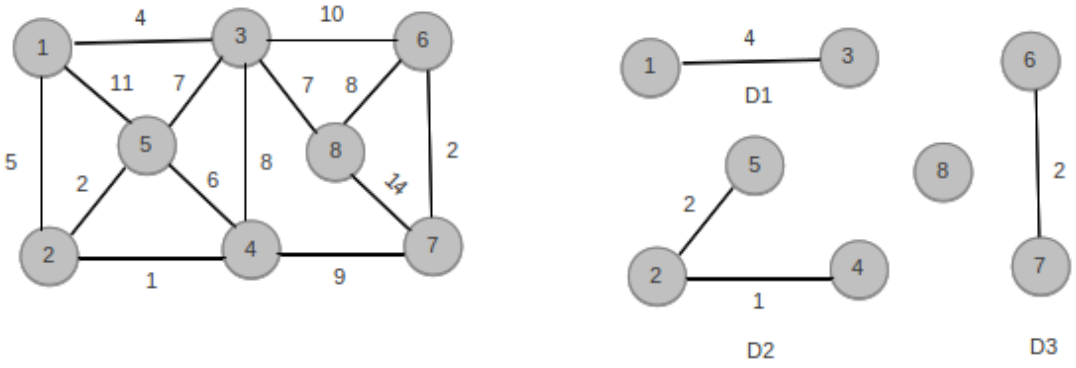
Figura 18 - Grafo dado como entrada e a árvore gerada depois de adicionadas *k* arestas na floresta *H*.



Fonte: Elaborada pelos autores.

Depois de concluída essa etapa, e a solução ainda não foi encontrada, aplicamos o algoritmo de Dijkstra (1959) entre todo par de vértices contidos em componentes distintas de *H*. A Figura 19 ilustra o caso em que a inserção de *k* arestas na floresta formam componentes com menos de *k* arestas.

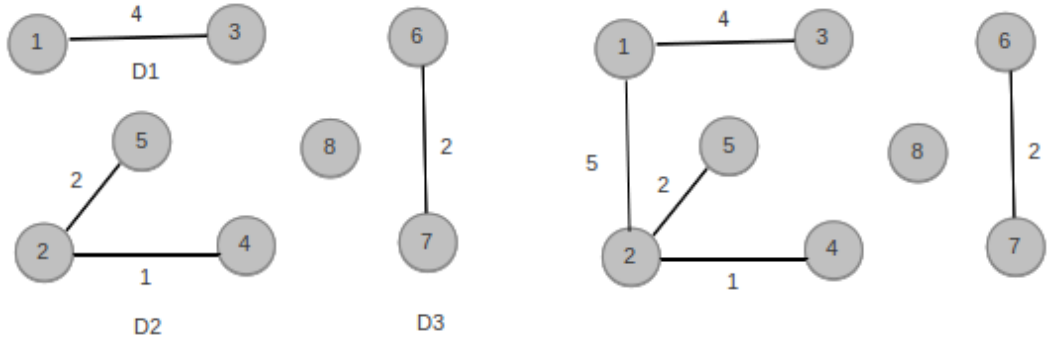
Figura 19 - Grafo dado como entrada e as componentes geradas depois que foram adicionadas *k* aresta na floresta *H*.



Fonte: Elaborada pelos autores.

Então para cada caminho mínimo *C* encontrado entre $u \in D_i$ e $v \in D_j$, onde D_i e D_j são componentes de H, calculamos o número de arestas que existem neste caminho. Se o número de arestas da sub-árvore $D_i U D_j U C$ for igual a *k* arestas, essa componente é retornada como solução. A Figura 20 ilustra o caso em que a união das componentes *D*1 e *D*2 na floresta *H* forma uma componente com exatamente *k* arestas, para *k* = 4.

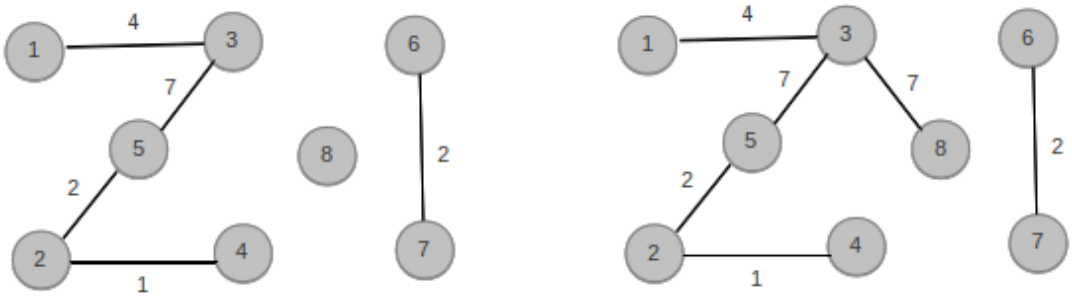
Figura 20 - Componentes encontradas depois de adicionadas *k* arestas na floresta *H* e a componente formada união das componentes *D1* e *D2* pelo o caminho de vértices 1 e 2.



Fonte: Elaborada pelos autores.

Caso $D_i U D_j U C$ tenha menos de *k* arestas, aplicamos o algoritmo Prim-ACM nesta componente para completarmos *k* arestas. A Figura 21 exemplifica o caso em que a união das componentes *D*1 e *D*2 forma uma componente com menos de *k* arestas, e a componente resultante depois de aplicado o Prim-ACM.

Figura 21 - União das componentes *D1* e *D2* pelos vértices 3 e 5, e a adição da aresta *G*(3 , 8) = 7 depois de aplicar o algoritmo Prim-ACM para formar a componente com *k* arestas.

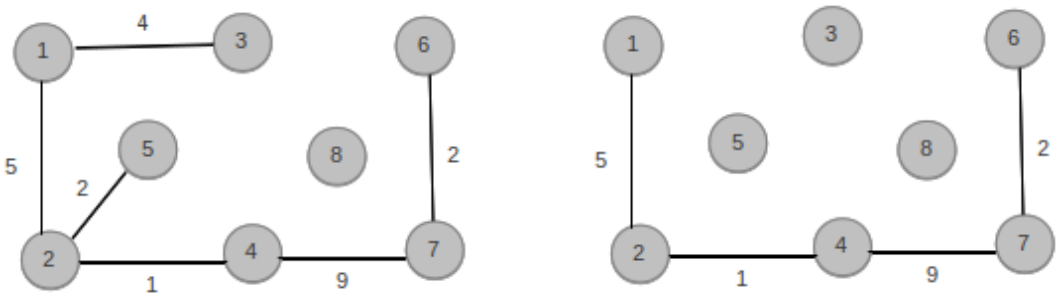


Fonte: Elaborada pelos autores.

Caso o número de arestas seja maior que *k*, aplicamos a função que remove arestas. Esta função é executada até que existam apenas *k* arestas nessa componente.

A função de remoção de arestas consiste na identificação e remoção da folha que possui aresta incidente de maior custo. Este processo é repetido até que restem apenas k arestas na solução. A Figura 22 ilustra a execução da função que remove arestas sobre a componente formada da união entre as componentes *D*1 e *D*3 através dos vértices 1 e 7.

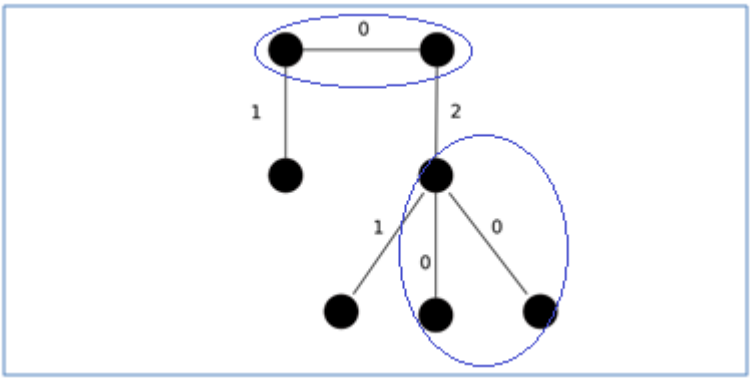
Figura 22 - União das componentes *D1* e *D3* pelos vértices 1 e 7, e a remoção das arestas *G*(1, 3) e *G*(1, 2), depois de aplicar a função que remove arestas.



Fonte: Elaborada pelos autores.

A ideia dessa heurística está na observação do caso em que o Prim-ACM não encontra a solução ótima, já que podemos ter duas componentes de custo mínimo que são desconectadas, mas são conectadas por uma aresta ou caminho de custo baixo. A Figura 23 ilustra o caso em que a aresta 2 de peso baixo impede que o Pim-ACM encontre à solução ótima, em seguida apresentamos o pseudocódigo da HKP-4.

Figura 23 - Exemplo para o qual a aplicação do algoritmo Prim (1957) em todos os vértices do grafo não encontra a solução ótima.



Fonte: Quintão et al. (2008).

Figura 24 - Ilustra o pseudocódigo da HKP-4.

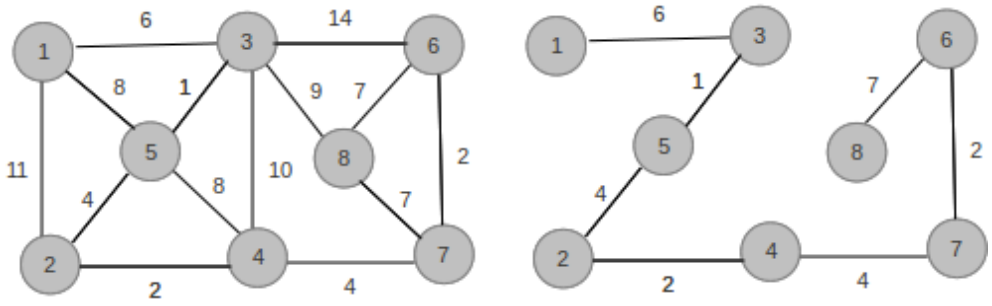
```
RemoverFolha(c3)

1 Encontrar a folha de maior custo em c3
2 Remover esta folha de c3
3 Atualizar o conjunto de folhas em c3
4 Retorna a folha de maior custo



HKP4(c1, c2, k)

1 Para u = 1 até c1.n faça
2      Para v = 1 até c2.n faça
3           cm = encontrar o menor caminho entre c1.s[u] e c2.s[v]
4           c3 = pegar componente da união entre c1, c2 e cm
5           Se c3.m == k
6              S = c3.w
7           Senão
8               Se c3.m > k
9                  Para w = 1 até c3.m - k faça


10                      c3.w -= RemoverFolha(c3)
11              S = c3.w
12         Senão
13              S = MST_PRIM(c3, k)
14         Se S < custo
15            custo = S
16 retorna custo
KRUSKAL-MOD()

1  QuickSort(arestasOrd, m)
2  Para i = 1 até m e c = 1 até k
3       u = arestasOrd[i].u
4       v = arestasOrd[i].v
5       Se FIND_SET(u) != FIND_SET(v)
6          adicionar a aresta E(u, v) na floresta H
7          pai = pegar o pai da componente que contém E(u, v) em H
8          Se c == k
9             c1 = pegar componente de raíz pai
10            Se c1.m == k
11               retorna c1.w
12 g = pegar os representantes das componentes da floresta H
13 Para u = 1 até w - 1 faça
14      Para v = 1 até w faça
15           c2 = pegar componente de raíz g[u] da floresta H
16           c3 = pegar componente de raíz g[v] da floresta H
17           S = HKP4(c2, c3, k)
18           Se S < custo
19              custo = S
20 retorna custo
```

Fonte: Elaborada pelos autores.

3.5 Heurística de Kruskal Prim Dijkstra (HKPD)

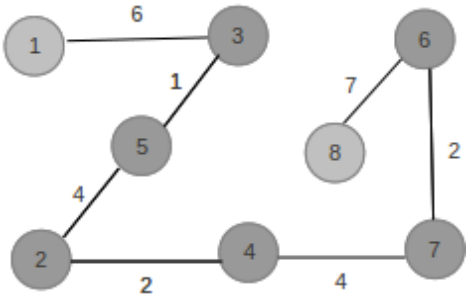
Na heurística Kruskal Prim Dijkstra (HKPD), aplicamos o algoritmo de Kruskal (1956) para encontrar a Árvore Geradora Mínima (AGM) do grafo *G*, e depois encontramos o conjunto *L* de todos os vértices não folha na AGM. Então, para cada dois vértices $u \in L$ e $v \in L$, aplicamos o algoritmo de Dijkstra (1959) para encontrar o caminho mínimo entre esses vértices, depois verificamos quantas arestas existem neste caminho. Se o número de arestas deste caminho for exatamente *k* arestas, retornamos o custo desse caminho como nossa solução. A Figura 25 exemplifica um grafo e sua AGM.

Figura 25 - Grafo dado como entrada e a sua AGM.



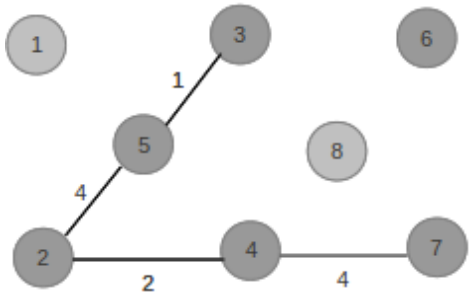
Fonte: Elaborada pelos autores.

Figura 26 - Ilustra os vértices não folha da AGM com a cor mais escura.



Fonte: Elaborada pelos autores.

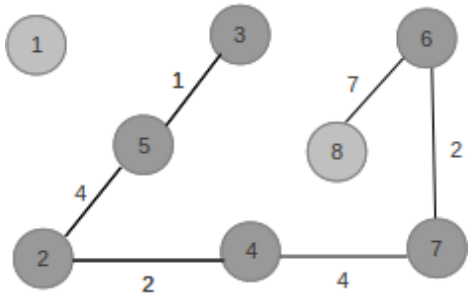
Figura 27 - Ilustra o caso em que o menor caminho entre dois vértices não folha tem exatamente *k* arestas.



Fonte: Elaborada pelos autores.

No entanto, se o caminho encontrado pelo Dijkstra (1959) for maior que *k* arestas, aplicamos a função que remove arestas, semelhante à utilizada na HKP-4. Assim obtemos *k* arestas de custo mínimo desse caminho e retornamos como solução. A Figura 28 ilustra o caso em que o menor caminho entre dois vértices não folha tem mais de *k* arestas.

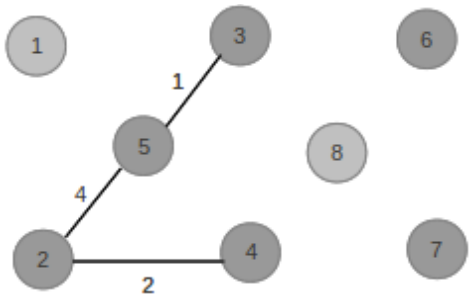
Figura 28 - Exemplo em que o menor caminho entre dois vértices não folha tem mais *k* arestas.



Fonte: Elaborada pelos autores.

Por outro lado, se o número de arestas for menor que *k* arestas, aplicamos o Prim-ACM para adicionar as arestas que faltam para completar a árvore com *k* arestas. A Figura 29 exemplifica o caso em que o número de arestas entre dois vértices não folha tem menos que *k* arestas.

Figura 29 - Exemplo em que o menor caminho entre dois vértices não folha tem menos de *k* arestas.



Fonte: Elaborada pelos autores.

Essa heurística busca através da utilização do Dijsktra contribuir para que a árvore de *k* arestas tenha o menor custo possível, visto que, utilizando apenas o algoritmo Prim (1957), podemos deixar de escolher uma componente de custo menor entre dois pares de vértices. Escolhendo apenas os vértices não folha, temos acesso a todos os outros vértices e ainda ganhamos em desempenho. Outro motivo de utilizar o algoritmo de Dijkstra (1959) é porque o caminho mínimo encontrado no grafo entre dois vértices poderá ser menor que o caminho entre os mesmo vértices na AGM. Apresentamos a seguir o pseudocódigo da HKPD.

Figura 30 - Ilustra o pseudocódigo da HKPD.

```
HKPD(k, u, v)

1  c = Encontra o menor caminho entre os vértices u e v
2  Se c.m == k
3     retorna c.w
4  Senão
5     Se c.m > k
6        Aplicamos a função que remove arestas de maior custo adjacêntes
7        a vértices folha até que existam apenas k arestas neste caminho
8        retorna c.w
9  Senão
10     retorna MST_PRIM(c, k)


KRUSKAL-MOD(k, n, m)

1  QuickSort(arestasOrd, m)
2  Para i = 1 até m e c = 1 até n - 1
3       u = arestasOrd[i].u
4       v = arestasOrd[i].v
5       Se aresta E(u, v) não gerar ciclo ao ser adiconada em H
6          adicionar a aresta E(u, v) na floresta H
7          incrementa c
8 g = pegar os vértices não folha da Árvore Geradora Mínima
9 Para u = 1 até w faça
10     Para v = u + 1 até w faça
11          S = HKPD(k, g[u], g[v])
12          Se S < custo
13             custo = S
14 retorna custo
```

Fonte: Elaborada pelos autores.

4 RESULTADOS COMPUTACIONAIS

Nossos resultados foram comparados com o melhor resultado existente na literatura para dois conjuntos de instâncias. O primeiro conjunto, chamada de g, foi proposto em Blesa e Xhafa (2000) e consiste de grafos 4-regulares. O segundo conjunto, chamado de bb, foi proposto em Blum e Blesa (2005) e consiste de grafos grid.

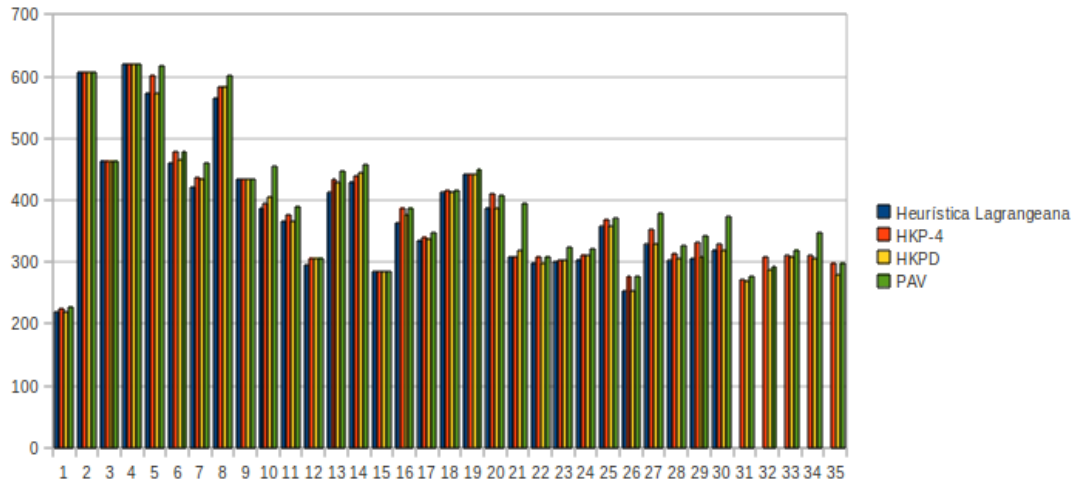
Comparamos nossos resultados com os resultados da heurística apresentada por Ehrgott et al. (1997), aqui chamada de Prim aplicado a todos os vértices (PAV), e a heurística lagrangeana apresentada por Quintão et al. (2008) e (2010).

Em todas as instâncias do conjunto *g*, a heurística HKPD ficou com soluções muito próximas da heurística lagrangeana, e em algumas instâncias obtivemos os mesmos resultados desta heurística. Um fator importante que devemos ressaltar é o tempo de execução das nossas heurísticas foi sempre menor que o da heurística lagrangeana. Estas duas heurísticas foram executadas em ambientes computacionais distintos. Entretanto, acreditamos que os resultados da nossa heurística teriam sido melhores se fosse executada no mesmo ambiente da Heurística Lagrangeana, pois utilizamos uma máquina 2.20 GHz, com 3GB de memória RAM.

Vale observar que para as instâncias do conjunto *g* com 1000 vértices, os autores reportam que a heurística lagrangeana estourou o tempo limite. Por outro lado, a heurística HKPD executou estas instâncias com um tempo médio de 175,264 segundos.

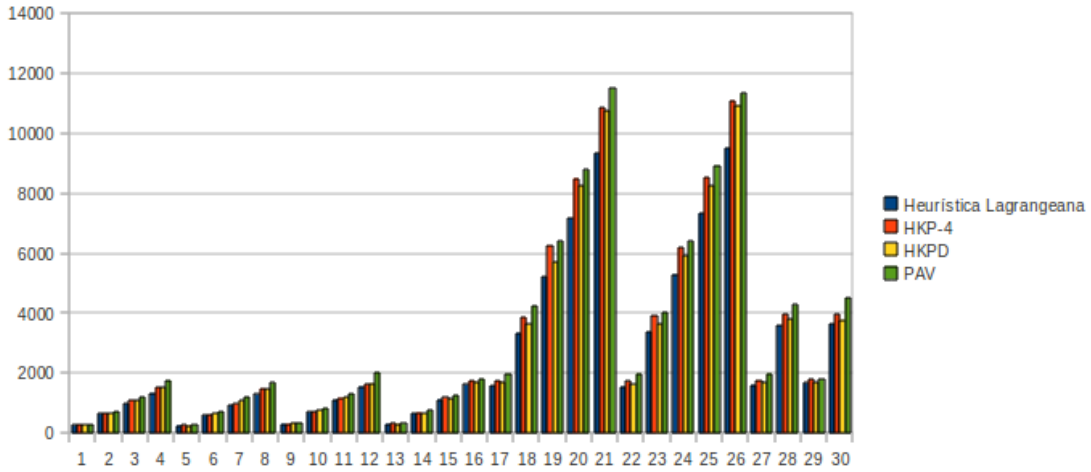
Para efeito de comparação, apresentamos a seguir a qualidade da solução fornecida pelas heurísticas lagrangeana, PAV, HKP-4 e HKPD, quando aplicada a todas as instâncias dos conjuntos *g* e *bb*. A Figura 5.1 fornece o gráfico comparando as soluções obtidas em cada instância do conjunto *g* das heurísticas Lagrangeana, HKP-4, HKPD e PAV.

Gráfico 1 – Compara as soluções das heurísticas Lagrangeana, HKP-4, HKPD e PAV obtidas em cada instância do conjunto g proposto por Blesa e Xhafa (2000).



Fonte: Elaborada pelos autores.

Gráfico 2 – Compara as soluções obtidas em cada instância do conjunto bb das heurísticas Lagrangeana, HKP-4, HKPD e PAV.



Fonte: Elaborada pelos autores.

Os resultados são também apresentados nas Tabelas 1 e 2 para cada instância disponível nos conjuntos g e bb.

Na Tabela 1 estão os resultados das heurísticas Lagrangeana, HKP-4, HKPD e PAV para as instâncias do conjunto *g* proposto por Blesa e Xhafa (2000). As heuríticas HKP-4 e HKPD alcançaram bons resultados porque em algumas soluções tiveram as mesmas soluções que a heurística Lagrangeana. Com relação ao tempo, nossas heurísticas obtiveram em todas as instâncias tempo menor que a Heurística Lagrangeana, mesmo executadas em

uma máquina de configuração inferior. Para as instâncias que não apresentaram resultado na heurística Lagrangeana foi devido à limitação de tempo computacional, segundo os autores.

Na Tabela 2 estão os resultados das heurísticas Lagrangeana, HKP-4, HKPD e PAV para as instâncias do conjunto *bb* proposto por Blum e Blesa (2005). As soluções obtidas na HKP-4 e HKPD ficaram próximas às encontradas pela a heurística Lagrangeana. Os tempos das heurísticas HKP-4 e HKPD são melhores que os da heurística Lagrangeana em todas as instâncias dos conjuntos *g* e *bb*.

Tabela 1 – Representa os resultados das heurísticas Lagrangeana, HKP-4, HKPD e PAV para as instâncias do conjunto *g*

Instâncias				Heurística Lagrangeana		HPK4		HPKD		PAV	
	N	*m*	*K*	Solução	T(s)	Solução	T(s)	Solução	T(s)	Solução	T(s)
g25-4-01	25	50	20	**219**	1.29	225	0.00	**219**	0.00	228	0.00
g25-4-02	25	48	20	**607**	1.12	**607**	0.00	**607**	0.00	607	0.00
g25-4-03	25	50	20	**464**	1.69	**464**	0.00	**464**	0.00	464	0.00
g25-4-04	25	48	20	**620**	1.79	**620**	0.00	**620**	0.00	620	0.00
g25-4-05	25	50	20	**573**	1.75	602	0.01	**573**	0.00	617	0.00
g50-4-01	50	98	20	**460**	7.04	478	0.02	466	0.01	478	0.00
g50-4-02	50	99	20	**421**	5.56	437	0.03	435	0.00	460	0.01
g50-4-03	50	99	20	**565**	6.79	583	0.03	584	0.00	601	0.00
g50-4-04	50	100	20	**434**	6.16	**434**	0.02	**434**	0.01	434	0.01
g50-4-05	50	100	20	**387**	3.78	394	0.02	406	0.00	454	0.00
g75-4-01	75	149	20	**366**	10.94	376	0.06	**366**	0.01	390	0.00
g75-4-02	75	150	20	**295**	8.42	307	0.04	307	0.01	307	0.01
g75-4-03	75	149	20	**412**	14.58	433	0.06	428	0.02	447	0.01
g75-4-04	75	150	20	**430**	14.00	439	0.05	444	0.01	457	0.01
g75-4-05	75	150	20	**284**	4.49	**284**	0.04	**284**	0.01	284	0.01
g100-4-01	100	200	20	**363**	18.01	388	0.09	377	0.04	388	0.02
g100-4-02	100	200	20	**335**	14.90	341	0.08	338	0.05	347	0.01
g100-4-03	100	200	20	**412**	17.35	416	0.11	**412**	0.04	416	0.02
g100-4-04	100	199	20	**442**	19.24	**442**	0.08	**442**	0.04	450	0.01
g100-4-05	100	199	20	**388**	19.88	410	0.10	**388**	0.03	408	0.02
g200-4-01	200	400	20	**308**	72.44	**308**	0.37	318	0.39	395	0.08
g200-4-02	200	400	20	**299**	63.81	309	0.35	**299**	0.41	309	0.07
g200-4-03	200	400	20	**300**	62.14	303	0.34	303	0.56	324	0.08
g200-4-04	200	399	20	**304**	75.50	312	0.31	310	0.45	321	0.08
g200-4-05	200	399	20	**357**	67.84	369	0.37	**357**	0.30	370	0.08
g400-4-01	400	800	20	**253**	261.26	277	1.66	**253**	5.05	277	0.42
g400-4-02	400	799	20	**328**	301.55	352	1.70	330	4.87	379	0.43
g400-4-03	400	799	20	**302**	279.99	314	1.66	306	5.85	326	0.43
g400-4-04	400	800	20	**306**	288.71	331	1.71	309	5.50	342	0.42
g400-4-05	400	800	20	**320**	315.71	328	1.59	**320**	4.70	373	0.42

g1000-4-01	1000	2000	20			271	13.44	**270**	179.84	277	4.99
g1000-4-02	1000	1999	20			308	14.50	**288**	172.33	292	4.99
g1000-4-03	1000	2000	20			310	14.37	**309**	166.29	320	4.97
g1000-4-04	1000	2000	20			310	13.42	**306**	188.86	347	4.96
g1000-4-05	1000	2000	20			299	13.37	**280**	169.00	299	4.95

Fonte: Elaborada pelos autores.

Tabela 2 – Representa os resultados das heurísticas Lagrangeana, HKP-4, HKPD e PAV para as instâncias do conjunto *bb*

Instâncias				Heurística Lagrangeana		HPK4		HPKD		PAV	
	N	*m*	*k*	Solução	T(s)	Soluçã o	T(s)	Soluçã o	T(s)	Soluç ão	T(s)
bb15x15_1	225	420	20	**257**	70.46	286	0.77	267	0.42	283	0.12
	225	420	40	**642**	90.71	661	1.49	652	0.50	714	0.18
	225	420	60	**977**	93.30	1099	2.88	1086	0.60	1196	0.21
	225	420	80	**1335**	100.70	1528	4.97	1515	0.65	1756	0.23
bb15x15_2	225	420	20	**253**	81.48	260	0.53	**253**	0.50	270	0.11
	225	420	40	**585**	86.24	616	1.57	666	0.72	709	0.16
	225	420	60	**927**	87.03	974	3.45	1074	0.80	1203	0.20
	225	420	80	**1290**	96.31	1487	5.62	1467	0.86	1666	0.22
bb45x5_1	225	400	20	**306**	79.28	309	1.02	342	0.43	342	0.12
	225	400	40	**695**	87.48	705	2.15	760	0.58	831	0.14
	225	400	60	**1107**	94.58	1175	3.07	1204	0.69	1305	0.20
	225	400	80	**1551**	103.38	1622	4.63	1658	0.76	2026	0.23
bb45x5_2	225	400	20	**302**	80.26	319	0.95	307	0.44	319	0.11
	225	400	40	**654**	86.23	688	1.72	657	0.59	754	0.16
	225	400	60	**1122**	95.390	1185	3.27	1137	0.71	1265	0.20
	225	400	80	**1617**	101.850	1749	5.70	1706	0.81	1808	0.23
bb33x33_1	1089	2112	100	**1577**	6633.150	1746	355.11	1703	287.46	1978	11.62
	1089	2112	200	**3342**	7079.540	3872	1318.36	3626	348.77	4245	17.13
	1089	2112	300	**5207**	7755.700	6257	2556.77	5716	390.29	6386	22.14
	1089	2112	400	**7172**	9011.410	8498	3925.81	8261	430.19	8774	20.75
	1089	2112	500	**9332**	6710.190	10858	6301.80	10758	470.40	11499	17.02
bb33x33_2	1089	2112	100	**1547**	5548.970	1724	363.18	1617	357.72	1985	6.12
	1089	2112	200	**3379**	5797.900	3919	1295.30	3643	416.61	4031	8.87
	1089	2112	300	**5296**	6096.480	6188	2416.86	5905	457.05	6387	11.35
	1089	2112	400	**7339**	6259.840	8536	4133.17	8254	493.43	8920	13.89
	1089	2112	500	**9510**	6559.050	11090	5943.30	10938	535.77	11327	16.13
bb100x10_1	1000	1890	100	**1605**	7369.16	1759	349.25	1682	176.26	1988	5.01
	1000	1890	200	**3593**	8881.59	3962	1069.52	3794	234.21	4272	7.56
bb100x10_2	1000	1890	100	**1668**	9809.180	1795	326.58	1688	193.71	1795	5.02
	1000	1890	200	**3637**	9157.960	3959	1375.04	3775	248.51	4510	7.56

Fonte: Elaborada pelos autores.

5 CONCLUSÃO E TRABALHOS FUTUROS

As heurísticas HKP-4 e HKPD mostraram-se competitivas, obtendo soluções muito próximas das encontradas pela heurística lagrangeana no conjunto *bb* proposto por Blum e Blesa (2005). Em algumas instâncias encontraram os mesmos valores de soluções no conjunto *g* proposto por Blesa e Xhafa (2000). O tempo obtido por nossas heurísticas são melhores em todas as instâncias do conjunto *g* proposto por Blesa e Xhafa (2000) e *bb* proposto por Blum e Blesa (2005), comparado com os da heurística Lagrangeana.

Como trabalho futuro, propomos o desenvolvimento de procedimentos de trocas de arestas para ser aplicado nas soluções encontradas pelas heurísticas com a finalidade de melhorar tal solução. Para melhorar o tempo de execução das nossas heurísticas poderão ser utilizadas estruturas mais apropriadas, por exemplo, a heap. Uma alternativa para acelerar o tempo de execução da HKP-4 é aproveitando a union-find, ao invés de aplicar a busca em profundidade. Além disso, outra ideia para acelerar a HKP-4 é colapsando as arestas, e aplicando o Dijkstra (1959) no grafo colapsado.

REFERÊNCIAS

BLUM, C**. Revisiting dynamic programming for finding optimal subtrees in trees.** European journal of operational research. v. 177, p.102-115. 16 feb. 2007.

BLUM, C; Blesa, M. J. **New metaheuristic approaches for the edge-weighted k-cardinality tree problem.** Computers and operations research. v. 32, p.1355-1377**.** jun. 2005.

BLESA, M; XHAFA, F. **A C++ Implementation of Tabu Search for *k*-Cardinality Tree Problem based on Generic Programming and Component reuse.** In Net Objective Days 2000. p.648–652. 2000.

CHEUNG, S; KUMAR, A. **Efficient Quorumcast Routing Algorithms.** In INFOCOM'94: Proceedings of the IEEE Conference on Computer Communications, Piscataway, NJ, USA. IEEE Society Press. p.840–847. 1994.

CORMEN, T. H. **Algoritmos:** teoria e prática. Rio de Janeiro: Campus, 2002.

DIJKSTRA, E. W. **A note on two problems in connexion with graphs.** Numerische Mathematik. v. 1, p.269–271. sep. 1959.

EHRGOTT, M. *et al*. **Heuristics for the *k*-Cardinality Tree and Subgraph Problems.** *Asia-Pacific Journal of Operations Research*. v. 14, p.87–114. 1997.

O Matemático das Empresas. EXAME, São Paulo, 21 set 2006. Disponível em: <http://exame.abril.com.br/revista-exame/edicoes/0877/noticias/o-matematico-das-empresas-m0111601>. Acesso em: 15 fev. 2013.

FISCHETTI, M. *et al*. **Weighted k-Cardinality Trees:** Complexity and polyhedral structure. Networks. v. 24, p.11–21. 1994.

GARG, N; HOCHBAUM, D. **An $O(\log_k)$ Approximation Algorithm for the k Minimum Spanning Tree Problem in the Plane.** Algorithmica. v. 18, p.111–121. 1997.

HAMACHER, H. W; JOERNSTEN, K. **Optimal Relinquishment According to the Norwegian Petrol Law:** A Combinatorial Optimization Approach. Norwegian School of Economics and Business Administration – Technical Report, 7/93. 1993.

KRUSKAL, J. **On the shortest spanning subtree of a graph and the travelling salesman problem.** Proceedings of the American Mathematical Society. v. 7, p.48–50. 1956.

MILLER, C; TUCKER, A; ZEMLIN, R. **Integer programming formulation of traveling salesman problems.** Journal of ACM. v. 7, p.326–329. 1960.

NEMHAUSER, G. **Sheduling on Demand Passenger Air Service.** In LAGOS'07: IV Latin American Algorithms, Graphs and Optimization Symposium, Puerto Varas, Chile. 2007.

PHILPOTT, A; WORMALD, W. **On the Optimal Extraction of ore from an open-cast mine.** University of Auckland – Technical Report. 1997.

PRIM, R.C. **Shortest connection networks and some generalizations**. Bell Sys. Tech. J. v. 36, p.1389–1401. 1957.

QUINTÃO, F.P. **O problema da árvore de custo mínimo com k arestas:** reformulações e relaxação lagrangeana. 2008. 54 f. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Minas Gerais. Belo Horizonte, 2008.

QUINTÃO, F.P; CUNHA, A.S; MATEUS, G.R; LUCENA, ABILIO. **The k-cardinality tree problem:** reformulations and lagrangian relaxation. Discrete applied mathematics. v. 158, p.1305-1314. 28 jun. 2010.