UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE ESTATÍSTICA E MATEMÁTICA APLICADA

PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM E MÉTODOS QUANTITATIVOS

MESTRADO ACADÊMICO EM MODELAGEM E MÉTODOS QUANTITATIVOS

MIRAH ALVES FERREIRA

NOVOS DESENVOLVIMENTOS PARA A SOLUÇÃO DO PROBLEMA DE PARTIÇÃO DE UM GRAFO EM ÁRVORES K-CAPACITADAS

FORTALEZA

2018

MIRAH ALVES FERREIRA

NOVOS DESENVOLVIMENTOS PARA A SOLUÇÃO DO PROBLEMA DE PARTIÇÃO DE UM GRAFO EM ÁRVORES K-CAPACITADAS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Modelagem e Métodos Quantitativos do Programa de Pós-Graduação em Modelagem e Métodos Quantitativos do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Modelagem e Métodos Quantitativos. Área de Concentração: Inteligência Computacional e Otimização

Orientador: Prof. Dr. Tibérius de Oliveira e Bonates

Co-Orientador: Prof. Dr. Manoel Bezerra Campêlo Neto

FORTALEZA

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F442n Ferreira, Mirah Alves.
Novos Desenvolvimentos para a Solução do Problema de Partição de um Grafo em Árvores k-Capacitadas / Mirah Alves Ferreira. – 2018.
54 f. : il.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Modelagem e Métodos Quantitativos, Fortaleza, 2018.
Orientação: Prof. Dr. Tibérius de Oliveira e Bonates.
Coorientação: Prof. Dr. Manoel Bezerra Campêlo Neto.

1. Branch-and-Bound. 2. Microagregação. 3. Particionamento de Grafos. 4. FlorestaGeradora. I. Título.
CDD 510

MIRAH ALVES FERREIRA

NOVOS DESENVOLVIMENTOS PARA A SOLUÇÃO DO PROBLEMA DE PARTIÇÃO DE UM GRAFO EM ÁRVORES K-CAPACITADAS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Modelagem e Métodos Quantitativos do Programa de Pós-Graduação em Modelagem e Métodos Quantitativos do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Modelagem e Métodos Quantitativos. Área de Concentração: Inteligência Computacional e Otimização

Aprovada em: 09 de Março de 2018

BANCA EXAMINADORA

Prof. Dr. Tibérius de Oliveira e Bonates (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Manoel Bezerra Campêlo Neto (Co-Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Albert Einstein Fernandes Muritiba
Universidade Federal do Ceará (UFC)

Prof. Dr. Pablo Mayckon Silva Farias
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Agradeço ao meu maravilhoso amigo, autor da minha fé e parceiro de todas as horas, Deus. Obrigada por me amar primeiro e por ter sonhado comigo antes mesmo de eu nascer. Obrigada por me mostrar que nada se compara a estar contigo e que um dia ao seu lado vale mais que uma vida inteira sem te conhecer. A Ti dedico meu trabalho, tudo o que tenho e tudo o que sou.

A minha família, minhas irmãs e minha mãe, por sonharem comigo e por sempre acreditarem em mim e em cada escolha que eu faço. Obrigada por me ensinarem tantas coisas e por me ajudarem a nunca desistir.

Ao meu orientador, professor Tibérius Bonates, por todo o aprendizado e paciência.

A todos os professores e servidores do Departamento de Estatística e Matemática Aplicada por toda ajuda que eu recebi durante os anos que passei aqui.

Aos meus colegas de turma e de laboratório que sempre estiveram prontos para dar auxílio naquilo o que sabiam e naquilo que não sabiam.

Finalmente, agradeço à Fundação Cearense de Apoio ao Desenvolvimento Cientifico e Tecnológico (FUNCAP) pelo financiamento da bolsa de estudos para a realização deste trabalho.

“Quem julga as pessoas não tem tempo para amá-las.”

(Madre Teresa de Calcutá)

RESUMO

O problema de partição de um grafo em árvores k-capacitadas (PAkC) tem como finalidade particionar um grafo em duas ou mais árvores, de forma a gerar componentes com um determinado número mínimo de vértices, em que a soma dos custos das arestas dessas componentes seja a menor possível. Nesse trabalho, realizamos um breve estudo bibliográfico sobre alguns métodos propostos na literatura para a resolução do PAkC. Dentre esses métodos, damos destaque a um algoritmo *Branch-and-Bound* e propomos melhorias a serem implementadas ao longo desse algoritmo. Apresentamos duas novas heurísticas para PAkC que buscam criar árvores que têm aproximadamente a mesma cardinalidade. Implementamos e avaliamos computacionalmente o algoritmo *Branch-and-Bound* e as melhorias propostas. Nossos experimentos computacionais mostram a eficiência de algumas das melhorias propostas ao método, pois o mesmo detém os melhores custos na função objetivo (limites) e os menores tempos de execução quando o comparamos aos algoritmos disponíveis na literatura e as heurísticas propostas nesse trabalho.

Palavras-chave: Branch-and-Bound, Microagregação, Particionamento de Grafos, Floresta Geradora.

ABSTRACT

The minimum k-capacitated tree partition problema (MkCTP) asks for a partition of a graph into two or more trees, in a way that each component spans a minimum number of vertices, whose sum of the edge weights is smallest possible. In this work, we briefly review some heuristics for solving the MkCTP. Among these, we give emphasis to a *Branch-and-Bound* algorithm and propose some improvements on it. We propose two novel heuristics for MkCTP that seek to create trees that have approximately the same cardinality. We evaluate, from a computational point of view, the branch-and-bound algorithm, as well as the proposed improvements. Our computational experiments show the efficiency of some of the proposed improvements to this method, since it has the best costs in the objective function (bounds) and the smallest running times when compared to the algorithms available in the literature and the heuristics proposed in this work.

Keywords: Branch-and-Bound, Microaggregation, Partitioning of Graph, Constrained Forest.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O estudo de grafos possui muitas aplicações que vão desde rotas de distribuições de produtos até redes de computadores. Dentre essas aplicações, em uma visão mais ampla, temos que escolhas realizadas por um indivíduo podem diferenciá-lo ou aproximá-lo de outros em uma rede de relações interpessoais. Dessa maneira, decidir como separar indivíduos em grupos pode não ser uma tarefa simples.

Dado um grafo não-orientado, ponderado e um inteiro positivo k, o Problema de Partição de um Grafo em Árvores k-Capacitadas (PAkC) consiste em particionar um grafo em árvores, cada uma das quais contendo, pelo menos, k vértices. Naturalmente, para termos um problema bem definido, é fundamental que o valor de k seja inferior ou igual ao número de vértices do grafo original. Além disso, o grafo de entrada pode ser desconexo desde que todas as componentes tenham pelo menos k vértices. O objetivo do PAkC é gerar um subconjunto de arestas, cuja soma dos custos seja mínima.

O PAkC é referido na literatura de distintas maneiras. Em (IMIELIŃSKA *et al.*, 1993), (GOEMANS; WILLIAMSON, 1995), (LASZLO; MUKHERJEE, 2005a) e (COUËTOUX, 2011) o PAkC é conhecido como Problema da Floresta Restrita (*Constrained Forest Problem*, ou simplesmente, CFP). O mesmo problema é descrito em (JI, 2004) como Problema da Floresta Restrita de Peso Mínimo (*Minimum Weight Constrained Forest Problem*). Esses trabalhos exploram o PAkC apresentando as heurísticas *Lightest Edge First* e *Heaviest Edge First*, o algoritmo de aproximação $\frac{3}{2}$ e o algoritmo de aproximação geral de Goemans e Williamson, brevemente descritos a seguir.

As heurísticas disponíveis na literatura para o PAkC mencionadas nesse trabalho divergem em seus procedimentos de duas formas: a heurística *Heaviest Edge First* retira as arestas de maior custo da árvore geradora mínima do grafo de entrada, desde que as árvores geradas permaneçam com pelo menos k vértices; a heurística *Lightest Edge First* e o algoritmo de aproximação $\frac{3}{2}$ selecionam arestas do grafo entrada em ordem crescente de custos para compor a solução, sem que haja formação de ciclos. O algoritmo de aproximação geral de Goemans e Williamson apresentada em (GOEMANS; WILLIAMSON, 1995) une as duas ideias descritas acima, selecionando arestas para compor a solução e, depois de obter uma solução completa, o algoritmo percorre a floresta retirando arestas desnecessárias.

O objetivo desse trabalho é fazer um estudo dos algoritmos combinatórios para resolver o PAkC propostos na literatura e propor melhorias para um algoritmo *Branch-and-*

Bound apresentado em (BONATES *et al.*, 2011). Essas melhorias dividem-se em: propostas para modificar o conjunto inicial de arestas utilizadas como ponto de partida para o algoritmo; aperfeiçoamentos no próprio algoritmo, por exemplo, melhorando o critério de poda da árvore *Branch-and-Bound*. Propomos, também, outros métodos de resolução para o PAkC.

O PAkC possui diferentes aplicações, em (DOMINGO-FERRER; MATEO-SANZ, 2002) e (LASZLO; MUKHERJEE, 2005b), por exemplo, um aplicação chamada de *Microagregação* é apresentada. Naqueles trabalhos um problema das agências estatísticas é relatado, onde estas têm o compromisso de expor dados reais preservando a confidencialidade das informações individuais, que podem ser empresas, pessoas, etc. Microagregação é uma técnica de limitação de divulgação é usada para proteger um conjunto de dados contendo registros específicos preservando o conteúdo informativo tanto quanto possível. Antes da divulgação dos dados, as informações são agrupadas em conjuntos de k ou mais registros individuais e os valores reais dos registros de cada grupo são substituídos por um novo registro que consiste da média destes. Em virtude disto, cada grupo deve conter um número mínimo k de registros, para que a confidencialidade de cada indivíduo seja protegida. Além disso, ao mantermos o valor de k pequeno, o agrupamento de registros semelhantes resulta em uma perda mínima de informações.

O presente trabalho está organizado como descrito a seguir. No Capítulo 2, fazemos uma breve introdução sobre os conceitos básicos utilizados em teoria dos grafos. No Capítulo 3, apresentamos definições e exemplos sobre o PAkC mencionando os principais algoritmos combinatórios existentes na literatura. No Capítulo 4, expomos um algoritmo *Branch-and-Bound* (BONATES *et al.*, 2011), definindo alguns conceitos e discutindo sobre os aspectos desta versão inicial do algoritmo. No Capítulo 5, propomos novos métodos de resolução para o PAkC e analisamos as etapas do algoritmo *Branch-and-Bound* propondo melhorias. O Capítulo 6, exibe os resultados computacionais referentes ás nossas propostas de melhorias. Finalmente concluindo este trabalho, o Capítulo 7 compreende a análise e discussão dos resultados obtidos durante o estudo.

2 CONCEITOS BÁSICOS

Neste capítulo, apresentamos conceitos e notações sobre Teoria dos Grafos e algoritmos que são utilizados ao longo do trabalho. Todos os conceitos podem ser encontrados com maior detalhe em (BONDY *et al.*, 1976), (JUNGNICKEL, 2007) e (BOLLOBÁS, 2013).

Um *grafo* G é um par $G = (V, E)$ que consiste de um conjunto finito $V^2 \neq \emptyset$ e um conjunto E de pares de elementos de V. Os elementos de V são chamados de *vértices*. Um elemento $e = \{a, b\}$ de E é chamado de *aresta* com *extremidades* a e b. Dizemos que a e b são *incidentes* a e e que a e b são *adjacentes* ou *vizinhos* um do outro. O número de vértices do grafo G é chamado de *cardinalidade* de V e é representado por $|V|$. De maneira análoga, o número de arestas de G é chamado de cardinalidade de E e denotado por $|E|$.

Seja $G = (V, E)$ um grafo e V' um subconjunto de V. Denotamos por $E|V'$ o conjunto de todas as arestas $e \in E$ em que ambos os vértices de extremidade estão em V'. O grafo $(V', E|V')$ é chamado de *subgrafo induzido* em V' e é denotado por $G|V'$. De mesma maneira, este grafo pode ser obtido por um subconjunto de arestas e seus respectivos vértices, nesse caso será denominado induzido por arestas. Cada grafo (V', E') em que $V' \subset V$ e $E' \subset E|V'$ é dito *subgrafo* de G, e um subgrafo onde $V' = V$ é chamado de *subgrafo gerador*.

Seja $(e_1, ..., e_n)$ uma sequência de arestas em um grafo G. Se houver vértices $v_0, ..., v_n$ tais que $e_i = \{v_{i-1}, v_i\}$ para $i = 1, ..., n$, a sequência é chamada de *passeio*. Se $v_0 = v_n$, então temos um passeio fechado. Para qualquer vértice v_a, consideramos (v_a) como um passeio trivial de comprimento 0, de modo que qualquer vértice esteja conectado a si mesmo. Um passeio no qual as arestas são distintas é chamado de *trilha*, e um passeio fechado com arestas distintas é uma *trilha fechada*. Se, além disso, os v_j são distintos, a trilha é um *caminho*. Uma trilha fechada com $n \geq 3$, no qual os vértices são distintos (exceto $v_0 = v_n$), é chamado de ciclo. Em qualquer um desses casos, usamos a notação

$$W: \qquad v_0 \overset{e_1}{\text{——}} v_1 \overset{e_2}{\text{——}} v_2 \text{ —— } \cdots \text{ —— } v_{n-1} \overset{e_n}{\text{——}} v_n$$

e dizemos que n o *comprimento* de W. Os vértices v_0 e v_n são chamados de *vértice de origem* e *vértice de destino* de W, respectivamente.

Dois vértices a e b de um grafo G são ditos *conexos* se existir um passeio com o vértice de origem a e com o vértice de destino b. Se todos os pares de vértices de G estiverem conectados, o grafo G dito *conexo*. A conexão é uma relação de equivalência no conjunto de

vértices de G. As classes de equivalência desta relação são chamadas de *componentes conexas* de G. Assim, G está conexo se, e somente se, o seu conjunto de vértices V é sua única componente conexa. As componentes que contêm apenas um vértice também são chamados *vértices isolados*.

Um grafo é chamado de *acíclico* se não contiver um ciclo. Para um subconjunto H do conjunto de vértices V de um grafo G, denotamos por $G \setminus H$ o subgrafo induzido em $V \setminus H$. Este grafo decorre de G, omitido todos os vértices em H e todas as arestas incidentes com esses vértices. Um grafo acíclico é chamado de *floresta*. Se uma floresta $T = (V', E')$ também for conexa, ou seja, se for possível estabelecer um caminho de qualquer vértice para qualquer outro vértice do grafo, então denominamos de *árvore*. Se T é um subgrafo de G e se $V' = V$, chamamos T uma *árvore geradora* de G.

Seja $w : E \to R$ uma função associando um número real a cada aresta de G. Para $e \in E$, definimos $w(e)$ como o peso da aresta e. Denotamos por $w(G) = \sum_{e \in E} w(e)$ o peso do grafo G. Uma árvore geradora T de G é dita mínima se $w(T) \leq w(T')$, para toda árvore geradora T' de G.

Um grafo direcionado ou um digrafo é um par $G = (V, E)$ consistindo de um conjunto finito V e um conjunto E de pares ordenados (a, b), onde $a \neq b$ são elementos de V. Os elementos de V são novamente chamados de vértices, no entanto o termo *arco* é usado no lugar de aresta para distinguir entre o caso direcionado e o não direcionado. O vértice a é chamado de *origem*, e b de *destino* de e.

Um algoritmo guloso é um procedimento computacional que toma decisões com base em um único critério, em vez de uma análise global que consideraria o efeito das decisões em outras etapas do algoritmo, ou seja, esse método realiza a melhor escolha local mesmo que o conjunto destas não produza uma solução globalmente ótima. Dado um número real $t \geq 1$, um algoritmo de aproximação t para um problema de minimização é um algoritmo que aceita qualquer instância do problema como entrada, e retorna uma solução viável, cujo valor de função objetivo é menor ou igual a t vezes o valor ótimo dessa instância. Quanto menor o valor de t, melhor será a aproximação.

3 REVISÃO BIBLIOGRÁFICA

Nesse capítulo, apresentamos a definição formal do PAkC e a sua complexidade computacional. Exibimos algoritmos combinatórios e métodos exatos para resolução do PAkC, discutindo algumas de suas divergências, seus aspectos em comum ilustrando com exemplos simples. Estes métodos possuem divergências quanto ao conjunto inicial de arestas usadas como ponto de partida, a classificação das arestas e a forma de seleção de arestas para compor a solução. Particularmente, discutimos a qualidade da solução gerada pela heurística *Heaviest Edge First* em comparação com a heurística *Lightest Edge First*.

3.1 Definição Formal do Problema

Seja o grafo conexo $G = (V, E)$, onde a cardinalidade de V é igual ao inteiro positivo n. Considere uma função $w : E \to \mathbb{R}^+$ de ponderação sobre E. Além disso, admita uma constante inteira k, sendo $2 \leq k \leq n$.

O PAkC consiste em encontrar um subconjunto $S \subseteq E$ de arestas, em que o custo total é o menor possível, dado por $\sum_{e \in S} w(e)$. É necessário que o subgrafo $F \subseteq G$ induzido por S, chamado de floresta k-capacitada, defina uma partição $\{C_1, C_2, ..., C_p\}$ de G, em que cada componente C_i induza uma árvore contendo pelo menos k vértices, ou seja, $|V(C_i)| \geq k$, $i \in \{1, 2, \ldots, p\}$

Em (IMIELIŃSKA *et al.*, 1993), o PAkC é apresentado como um problema $\mathcal{NP}$-difícil para $k \geq 4$. É possível provar essa complexidade computacional fazendo uso da redução do problema de emparelhamento tridimensional. A prova pode ser realizada em duas etapas. A primeira etapa consiste em provar que o problema é $\mathcal{NP}$-difícil para $k = 4$, e então é possível generalizar a demonstração para $k > 4$.

Em (GOEMANS; WILLIAMSON, 1995), foi proposto um algoritmo para esse problema que possui um fator $\left(2 - \frac{1}{|V|}\right)$ de aproximação e em (COUËTOUX, 2011), essa aproximação foi reduzida para $\frac{3}{2}$ com um novo algoritmo combinatório para o PAkC.

3.2 Métodos de Resolução do PAkC

3.2.1 Heurística Lightest Edge First

Considere uma árvore geradora mínima $T = (V, E_1)$ de um grafo $G = (V, E)$, em que $E_1 \subseteq E$. A heurística gulosa *Lightest Edge First* (LEF) estudada em (IMIELIŃSKA *et al.*, 1993) e em (LASZLO; MUKHERJEE, 2005a) busca construir um subgrafo F a partir de T, em que F é uma floresta k-capacitada, ou seja, uma solução para o PAkC. A heurística tem como estratégia inserir em F as arestas de menor custo de T e que ainda não pertencem a F, sempre que pelo menos uma das extremidades da aresta for uma árvore com menos de k vértices.

A entrada para essa heurística é o conjunto de arestas E_1 e um inteiro positivo k. A saída da heurística é um subconjunto de arestas $S \subset E_1$ que induz o subgrafo F. A cada iteração o método seleciona a aresta e de menor custo de E_1 que não compõe S, e verifica se a inserção dessa aresta em S não conecta duas árvores de tamanho pelo menos k.

Algoritmo 1: *Lightest Edge First*

Entrada: E_1(conjunto de arestas), $k \in \mathbb{Z}_+$.
Saída: S(conjunto de arestas da floresta k-capacitada).

```
1  início
2      S ← ∅
3      repita
4          Selecione uma aresta e ∈ E1 de menor custo.
5          E1 ← E1 \ {e}
6          se e não conecta duas árvores com pelo menos k vértices no grafo induzido por S
               então
7              S ← S ∪ {e}
8          fim
9      até E1 = ∅;
10 fim
11 retorna S
```

A Figura (1) representa uma árvore geradora mínima T de um grafo G e a solução gerada pela heurística LEF, em que apenas a aresta $\{b, g\}$ não é incluída.

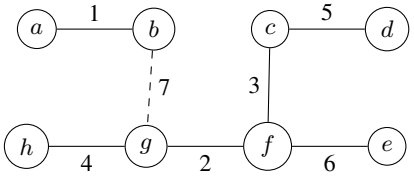

Figura 1 – Árvore geradora mínima/Solução obtida pelo LEF para k=2.

3.2.2 Heurística Heaviest Edge First

A heurística gulosa *Heaviest Edge First* (HEF) foi proposta em (LASZLO; MUKHERJEE, 2005a). Seguindo o mesmo objetivo do algoritmo combinatório da Seção (3.2.1), a heurística HEF tem como ponto de partida uma árvore geradora mínima T de um grafo de entrada G. No entanto, de maneira contrária ao LEF, a heurística HEF busca chegar a uma floresta k-capacitada F retirando da árvore geradora mínima T a aresta de maior custo sempre que esta unir duas árvores com k vértices ou mais.

A heurística tem como entrada o conjunto de arestas E_1 da árvore T. A cada iteração a heurística seleciona a aresta e de maior custo de E_1 e verifica se é possível retirá-la de forma a gerar duas árvores de tamanho pelo menos de k. Quando não houver mais arestas que possam ser removidas, chegamos a um subconjunto de arestas S que é minimal, já que induz uma floresta k-capacitada e a remoção de qualquer outra aresta de S acarretaria a existência de uma componente menor que k no subgrafo induzido por S.

Algoritmo 2: *Heaviest Edge First*

Entrada: E_1(conjunto de arestas), $k \in \mathbb{Z}_+$.
Saída: S(conjunto de arestas da floresta k-capacitada).

```
1  início
2      S ← E_1
3      repita
4          Selecione uma aresta e ∈ E_1 de maior custo.
5          E_1 ← E_1 \ {e}
6          se e conecta duas árvores de cardinalidade pelo menos k no grafo induzido por S
            então
7              S ← S \ {e}
8          fim
9      até E_1 = ∅;
10 fim
11 retorna S
```

Fazendo uso do mesmo grafo da Figura (1), temos na Figura (2) a representação da solução gerada pelo HEF a partir de uma árvore geradora mínima.

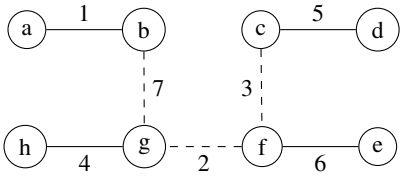

Figura 2 – Árvore geradora mínima/Solução obtida pelo HEF para k=2.

Seja $T = (V, E_1)$ uma árvore geradora mínima de um grafo $G = (V, E)$, onde $E_1 \subseteq E$. A partir do conjunto de arestas E_1, temos que a floresta k-capacitada produzida pela heurística HEF é um subgrafo da floresta produzida pela heurística LEF. Este resultado pode ser obtido através do Teorema (3.1) e a demonstração do respectivo teorema pode ser obtida com maior detelhe em (LASZLO; MUKHERJEE, 2005a).

Teorema 3.1. *Seja F_1 uma floresta k-capacitada geradora produzida pelo método Heaviest Edge First e F_0 uma floresta k-capacitada geradora produzida pelo método Lightest Edge First. Então $F_1 \subset F_0$.*

3.2.3 Algoritmo de Aproximação Geral de Goemans e Williamson

Em (GOEMANS; WILLIAMSON, 1995) foi proposto um algoritmo combinatório para uma grande classe de problemas em grafos, um deles é o PAkC. No algoritmo proposto, foi considerado um grafo $G = (V, E)$, uma função $f : 2^{|V|} \rightarrow \{0, 1\}$ e uma função de custo não negativa $w : E \rightarrow \mathbb{Q}^*$, obtendo-se o seguinte modelo de programação inteira:

$$\text{(IP)} \quad \text{minimizar} \quad \sum_{e \in E} w(e)x_e$$

sujeito a:

$$x(\delta(H)) \geq f(H) \;\; \forall \, \emptyset \neq S \subset V \tag{3.1}$$

$$x_e \in \{0, 1\} \;\; e \in E, \tag{3.2}$$

Onde $\delta(H)$ representa o conjunto de arestas, no qual exatamente uma das extremidades da aresta pertence a H e $x(\delta(H)) = \sum_{e \in \delta(H)} x_e$. Este modelo de programação inteira interpreta o PAkC como um problema de cobertura, no qual é necessário cobrir todos os subconjuntos H, cujo $f(H) = 1$. A função $f(H)$ segue duas propriedades:(i) De simetria, onde $f(H) = f(V - H)$ $\forall \; H \subseteq V$; de disjunção, em que se A e B são disjuntos, então $f(A) = f(B) = 0$ implica que $f(AUB) = 0$. Em outras palavras, temos que a função $f(H) = 1$ para todo subconjunto $H \subset G$ que possui menos que k vértices. Além disso, temos o modelo dual da relaxação linear de (IP) é dado pelo modelo a seguir.

(D) maximizar $\sum_{H \subset V} f(H) y_H$

sujeito a:

$$\sum_{H: e \in \delta(H)} y_H \leq c_e \quad e \in E \tag{3.3}$$

$$y_H \geq 0 \quad \emptyset \neq H \subset V, \tag{3.4}$$

O algoritmo é $\left(2 - \frac{1}{|V|}\right)$-aproximativo e a base para sua construção consiste na criação de uma floresta F', que está inicialmente vazia. As arestas de F' serão candidatas a compor F. Em cada iteração o algoritmo seleciona uma aresta $\{i, j\}$ entre duas componentes distintas C_1 e C_2 de F'. Então $\{i, j\}$ é adicionada a F' unindo as componentes C_1 e C_2. O ciclo termina quando $f(C) = 0$ para toda componente conexa C de F'. Finalmente, todas as arestas são novamente avaliadas a fim de ratificar sua real necessidade na solução F.

Comparando com as heurísticas anteriormente mencionadas, temos que esse método aplica uma variação da heurística LEF e, tendo alcançado uma solução completa, aplica a heurística HEF com o intuito de retirar arestas incluídas desnecessariamente. Esse algoritmo é considerado de aproximação geral pelo fato de a função própria f poder ser adaptada para diversos problemas de grafos.

Algoritmo 3: Algoritmo de Aproximação Geral de Goemans e Williamson

Entrada: G, c, f
Saída: F, LB
1 **início**
2 $F \leftarrow \emptyset$
3 *Comentário: Implicitamente* $y_H \leftarrow 0$ *para todo* $H \subset V$.
4 $LB \leftarrow 0$
5 $\mathcal{C} \leftarrow \{\{v\} : v \in V\}$
6 **para** $\forall v \in V$ **faça**
7 $d(v) \leftarrow 0$
8 **fim**
9 **repita**
10 Ache uma aresta $e = \{i, j\}$ com $i \in C_p \in \mathcal{C}$, $j \in C_q \in \mathcal{C}$, $C_p \neq C_q$ que minimize $\epsilon = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)}$
11 $F' \leftarrow F' \cup \{e\}$
12 **para** $\forall v \in C_r \in \mathcal{C}$ **faça**
13 $d(v) \leftarrow d(v) + \epsilon \cdot f(C_r)$
14 **fim**
15 *Comentário: Implicitamente* $y_C \leftarrow y_C + \epsilon \cdot f(C)$ *para todo* $C \subset \mathcal{C}$.
16 $LB \leftarrow LB + \epsilon \cdot \sum_{C \in \mathcal{C}} f(C)$
17 $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_p \cup C_q\} - \{C_p\} - \{C_q\}$
18 **até** $\forall C \in \mathcal{C} : f(C) = 0$;
19 $F \leftarrow \{e \in F'$: para alguma componente conexa N de $(V, F' - \{e\})$, $f(N) = 1\}$
20 **fim**
21 **retorna** F

Seja dita *ativa* qualquer componente C de F no qual $f(C) = 1$. Em cada iteração, o algoritmo tenta aumentar y_C uniformemente para cada componente ativa C com o maior valor possível sem violar a restrição (3.3). Logo,

$$d(i) + d(j) + \epsilon \cdot f(C_p) + \epsilon \cdot f(C_q) \leq c_e$$

para todo $e = (i, j) \in E, i \in C_p, j \in C_q$, em que $C_p \neq C_q$. Assim, o maior aumento viável em uma iteração é dado pela instrução no passo 10. Uma vez que os vértices i e j de uma aresta $e = \{i, j\}$ estão na mesma componente, a soma $\sum_{H : e \in \delta(H)} y_H$ é não crescente no decorrer das iterações.

3.2.4 Algoritmo de Aproximação $\frac{3}{2}$

Em (COUËTOUX, 2011), é apresentado um novo método para resolver o PAkC que tem como objetivo reduzir a aproximação de fator 2 das heurísticas LEF e HEF e de fator $\left(2 - \frac{1}{|V|}\right)$ do algoritmo de aproximação geral de Goemans e Williamson para $\frac{3}{2}$, além de

construir uma floresta k-capacitada F. Contudo, antes de dar inicio ao método, aquele trabalho, apresenta duas definições que são utilizadas durante o algoritmo: a primeira é que uma árvore com menos de k vértices é considerada *pequena*, caso contrário a árvore é dita *grande*; a segunda é que uma aresta é dita *boa* se, e somente se, cada uma das suas extremidades está em uma árvore pequena e se a inclusão desta arestas na solução parcial do PAkC produz uma árvore grande, caso contrário a aresta é dita *ruim*.

O algoritmo parte do conjunto E e busca construir um subconjunto de arestas $S \subseteq E$ que induz o subgrafo F. O algoritmo consiste em selecionar uma aresta boa, de menor custo, $e \in E$ e uma ruim, de menor custo $e' \in E$ que não pertençam a S. Se $w(e) \leq 2w(e')$, então essa aresta e é inserida em S e retirada de E. Caso contrário, a aresta ruim e' é avaliada. Se e' é incidentes a um vértice com um único vizinho em uma das suas extremidades e se a sua inclusão em S não cria um ciclo em F, então essa aresta e' é incluída em S e excluída de E. Caso a aresta e' não respeite essas duas condições, então apenas ocorre o seu descarte.

Note que de forma diferente das heurísticas LEF e HEF, o algoritmo de aproximação $\frac{3}{2}$ não necessita de uma árvore geradora mínima como entrada. A razão dessa divergência é o fato de que o algoritmo de aproximação $\frac{3}{2}$ verifica se as arestas formam ciclos antes de inclui-las na solução.

Algoritmo 4: Algoritmo de aproximação $\frac{3}{2}$

Entrada: E (conjunto de arestas).
Saída: S (conjunto de arestas da floresta k-capacitada).

```
1  início
2      S ← ∅
3      repita
4          e ← aresta boa de menor custo de E
5          e' ← aresta ruim de menor custo de E
6          se w(e) ≤ 2w(e') então
7              S ← S ∪ {e}
8              E ← E \ {e}
9          fim
10         senão
11             se e' possui uma folha em uma árvore pequena do subgrafo F induzido por S e
                F ∪ {e'} não contém ciclos então
12                 S ← S ∪ {e'}
13             fim
14             E ← E \ {e'}
15         fim
16     até E = ∅;
17 fim
18 retorna S
```

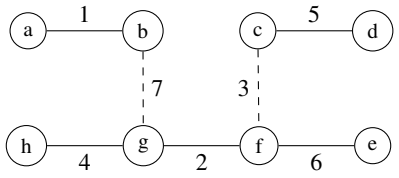

Figura 3 – Grafo de entrada/Solução obtida pelo algoritmo de aproximação $\frac{3}{2}$ para k=2.

Na figura (3), temos que a aresta $\{f, g\}$ faz a união duas árvores que já possuem k-vértices. Na iteração em que essa aresta é avaliada, na Figura (4), temos que $\{f, g\}$ é considerada uma aresta boa, visto que os vértices f e g ainda são componentes de cardinalidade igual a 1, e a aresta ruim de menor custo é $\{b, g\}$. Como $w(\{f, g\}) \leq 2w(\{b, g\})$, a aresta $\{f, g\}$ é inserida.

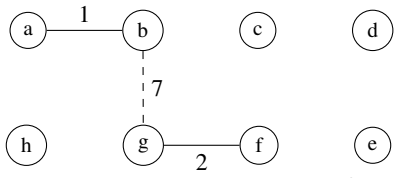

Figura 4 – Segunda iteração do algoritmo de aproximação $\frac{3}{2}$ para k=2.

3.2.5 Modelo de Programação Matemática

Partindo do grafo original G definido na Seção (3.1), a formulação matemática para o PAkC apresentada em (BONATES *et al.*, 2011) faz uso de um grafo direcionado $D = (V, A)$, onde A é o conjunto de arcos e (v_i, v_j) e $(v_j, v_i) \in A$, para cada aresta $\{v_i, v_j\} \in E$. Acrescentado um vértice artificial v_0 consideramos os arcos artificiais, (v_0, v_i) para todo $i = 1...n$. Agora, temos que enviar n unidades de fluxo de v_0 para os vértices pertencentes a V. Para garantir a viabilidade da solução, supomos que todos os ciclos têm custo positivo. O modelo matemático elege um subconjunto de arestas do digrafo D e estabelece uma partição do mesmo.

(ND) minimizar $\sum_{e \in E} w(e) y_e$

sujeito a:

$$\sum_{j \in V} x_{0,j} = n \tag{3.5}$$

$$x_{0,i} + \sum_{(j,i) \in A} x_{j,i} - \sum_{(i,j) \in A} x_{i,j} = 1, \quad \forall\, i \in V \tag{3.6}$$

$$k z_i \leq x_{0,i} \leq n z_i, \quad \forall\, i \in V \tag{3.7}$$

$$\sum_{i \in V} z_i \leq \left\lfloor \frac{n}{k} \right\rfloor \tag{3.8}$$

$$\sum_{e \in E} y_e + \sum_{i \in V} z_i = n \tag{3.9}$$

$$\left.\begin{array}{l} x_{i,j} \leq (n-1) y_e \\ \\ x_{j,i} \leq (n-1) y_e \end{array}\right\} \quad \forall\, e = \{i,j\} \in E \tag{3.10}$$

$$x_{i,j} \geq 0, \quad \forall\, (i,j) \in A \tag{3.11}$$

$$y_e \in \{0,1\}, \quad e \in E \tag{3.12}$$

$$z_i \in \{0,1\}, \quad i \in V. \tag{3.13}$$

Nesse modelo o grafo direcionado $D = (V, A)$ deve respeitar restrições de fluxo (3.5), (3.6) e (3.10) e de conectividade das árvores (3.7), (3.8) e (3.9), que devem possuir, pelo menos, k vértices.

É possível que o modelo gere diversas soluções que de maneiras diferentes representam a mesma solução viável do PAkC. Isso é devido ao fato de que o modelo escolhe apenas uma variável z_i, para algum i da componente $C \subseteq V$, para ter valor não nulo. Logo, existem $|C|$ possibilidades de representação para cada componente C. Para evitar esse problema, chamado de *simetria do modelo*, fazemos uso da ideia de *vértice de referência*. Esse vértice é um nó escolhido dentro de cada componente $C \subseteq V$ para representá-la.

Na tentativa de reduzir a quantidade de soluções simétricas, dois novos grupos de restrições poderiam ser inseridas no modelo: a restrição (3.14) determina que os vértices de referências devem ser folhas; as restrições (3.15) e (3.16) o vértice de menor índice passa a ser o vértice de referência.

$$\sum_{e=\{v_i,v_j\}\in E} y_e \leq 1 + (n-2)(1-z_i), \ \ \forall\, v_i \in V \tag{3.14}$$

$$y_e + z_j \leq 1, \ \ \forall\, e = \{v_i, v_j\} \in E, \ \ i \leq j \tag{3.15}$$

$$\sum_{i=n-k+2}^{n} z_i = 0 \tag{3.16}$$

Utilizar ambos os grupos de restrições para os vértices de referência pode tornar o problema inviável, devido ao fato de que os conjuntos de restrições aplicados simultaneamente obrigam o vértice de menor índice a ser, também, uma folha. Por isso, apenas as restrições (3.15) e (3.16) sobre os vértices de referência são utilizadas na implementação desse modelo.

4 ALGORITMO *BRANCH-AND-BOUND*

Como descrito na Seção (3.1), seja $G = (V, E)$ um grafo não-orientado e ponderado. É necessário que o subgrafo $F \subseteq G$ induzido por S seja uma coleção de árvores k-capacitadas e que o conjunto de arestas S tenha custo total mínimo. O algoritmo *Branch-and-Bound* (B&B) apresentado em (BONATES *et al.*, 2011) busca explorar as possibilidades de inclusão ou de descarte de uma aresta na solução, e as consequências dessas escolhas.

Esse algoritmo *Branch-and-Bound* parte do conjunto de arestas E e a cada iteração considera a possibilidade de incluir e de não incluir uma aresta em S fazendo uso de uma árvore de *Branch-and-Bound*. Dessa forma, é possível verificar que soluções podem ser geradas a partir de cada escolha e se estas são viáveis ou não.

Antes de dar inicio a de criação e ramificação de nós da árvore de *Branch-and-Bound* verificamos a existência de arestas que devem fazer parte de qualquer solução viável do PAkC com $k \geq 2$. As arestas incidentes a vértices que possuem apenas um vizinho devem, necessariamente, fazer parte de S (Figura 5). Caso contrário, esses vértices não fariam parte de nenhuma árvore com k ou mais vértices.

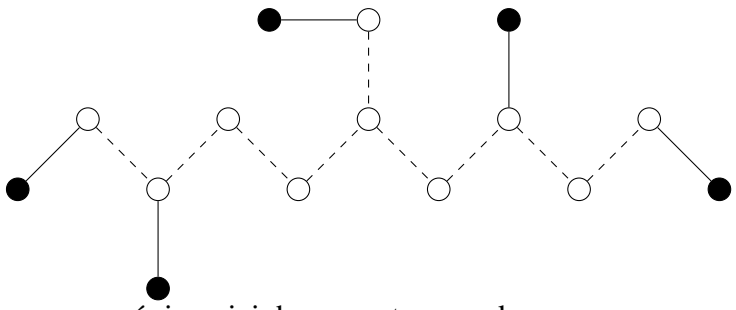

Figura 5 – Vértices com um único vizinho e arestas que devem compor qualquer solução com $k \geq 2$.

Aplica-se esse procedimento até que se esgotem todos os vértices com apenas um vizinho. É possível que após esse procedimento tenhamos obtido uma solução viável para o PAkC, nesse caso o algoritmo será finalizado. Caso contrário, temos uma primeira solução parcial para o PAkC, ou seja, um conjunto de arestas que induz uma floresta, na qual pelo menos uma componente tem menos de k vértices. Além desse conjunto de arestas que incidem sobre vértices com apenas um vizinho, é possível que exista um outro conjunto de arestas necessárias a uma solução viável. Essas arestas são incidentes a vértices chamados de *vértices de ligação*, a serem descritos a seguir.

Definição 4.1. *Um vértice v é considerado de ligação se, e somente se, v pertence a uma componente da solução parcial atual F com menos de k vértices e possui apenas uma aresta incidente que não pertence a S, chamada de aresta de ligação.*

Dado que a solução parcial contém arestas de ligação, estas devem, necessariamente, compor a solução. Isso devido ao fato que o descarte de uma aresta de ligação gera uma componente inviável. Caso exista algum vértice de ligação, a aresta incidente a ele, que ainda não pertence a S, deve ser inserida em F (Figura 6). Dessa maneira atualizamos a solução parcial com arestas de ligação.

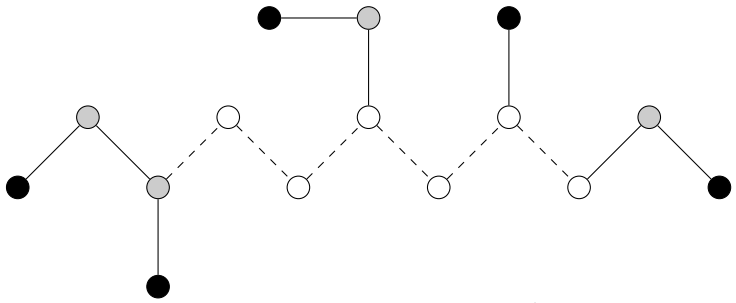

Figura 6 – Vértices (em cinza) e arestas de ligação para um $k \geq 3$.

Após esse pré-processamento, iniciamos o processo de criação e ramificação de nós da árvore de *Branch-and-Bound* de maneira recursiva. O primeiro nó da árvore de *Branch-and-Bound* representa a solução parcial gerada a partir desse pré-processamento.

Cada nó da árvore de *Branch-and-Bound* guarda um conjunto de arestas que ainda são candidatas a participar da solução e o conjunto de arestas que compõem a solução parcial, denotado por S. Dentro do conjunto de arestas que ainda são candidatas seleciona-se uma aresta, que não forme ciclo na solução parcial que está sendo analisada, para realizar uma ramificação na árvore de branch-and-bound criando dois novos nós.

No primeiro nó gerado a partir da ramificação, a aresta de ramificação é incluída na solução parcial. No segundo nó, esta aresta de ramificação é simplesmente descartada. Em ambos nós gerados a aresta de ramificação passa a não fazer mais parte do conjunto de arestas ainda candidatas.

Após a inclusão ou não da aresta de ramificação, o algoritmo busca arestas incidentes a vértices com um único vizinho e arestas de ligação, as inclui na solução parcial e as retira do conjunto de arestas ainda candidatas. Isso é devido ao fato que ao fixar uma aresta, outras arestas podem tornar-se de ligação ou incidentes a vértices com um único vizinho. Dessa maneira, é

possível fixar arestas na solução parcial sem a necessidade de gerar um novo nó na árvore de *Branch-and-Bound*.

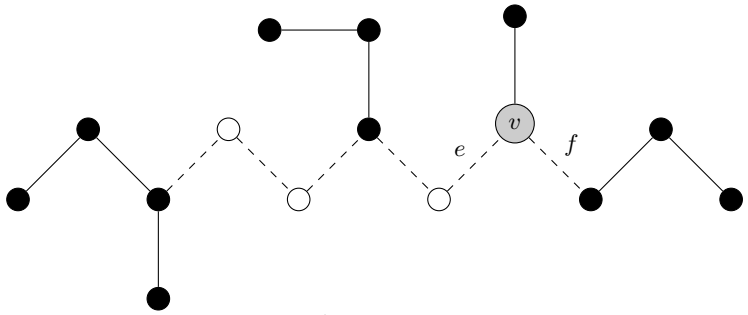

Figura 7 – Vértice de ramificação para um $k \geq 3$.

Na Figura (7), o vértice v está em uma componente inviável, ou seja, uma árvore com menos de k vértices. As arestas e e f podem ser selecionadas para solução a fim tornar esta componente viável. Diante disso, podemos tomar a aresta e para ramificação e criar dois novos nós na árvore de *Branch-and-Bound*, onde cada nó corresponde a uma solução parcial do PAkC. Ver Figura (8).

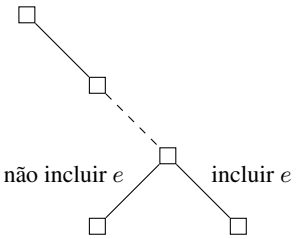

Figura 8 – Árvore de *Branch-and-Bound*.

Na Figura (9), temos que ao descartar a aresta e, a aresta f tornou-se de ligação e esta pode ser acrescentada à solução parcial sem a necessidade de ser avaliada na subárvore da árvore de *Branch-and-Bound* gerada a partir da não inclusão da aresta e.

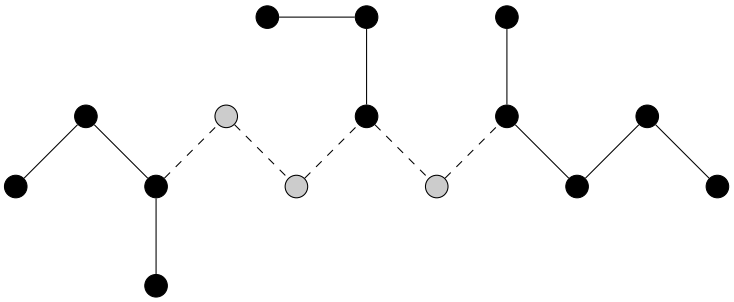

Figura 9 – Solução parcial em que a aresta e não é incluída.

A partir de cada nó criado na árvore de *Branch-and-Bound*, o algoritmo irá ramificar

este nó gerando dois novos nós na árvore de *Branch-and-Bound* até que todas as soluções possíveis sejam geradas. No entanto, antes de ramificar cada nó da árvore de *Branch-and-Bound*, o custo da função objetivo desse nó deverá ser comparado com o custo da melhor solução viável encontrada até a iteração atual do algoritmo. Caso o valor da solução parcial seja maior ou igual ao custo da melhor solução completa encontrada, é possível descartar esse nó. Isso devido ao fato que essa solução parcial não poderá gerar uma solução completa com um custo menor que o custo da melhor solução viável atual. No final desse processo a solução retornada pelo algoritmo será a solução viável com o menor custo de função objetivo encontrada.

4.1 Conjunto Inicial de Arestas

O algoritmo *Branch-and-Bound* pode ser aplicado sobre qualquer grafo não-orientado e ponderado. O grafo poderá ser desconexo desde que todas as suas componentes tenham pelo menos k vértices. No entanto diante de um grafo denso, o algoritmo pode não ter um bom desempenho devido à quantidade de vizinhos que cada vértice possui, uma vez que seu resultado depende da sua capacidade de fixar as variáveis que representam as arestas. Em um grafo denso, cada vértice tende a possuir diversos vizinhos e, por isso, a ideia de buscar vértices com apenas um vizinho e vértices de ligação é dificilmente aplicada. Uma solução para esse problema é aplicar o algoritmo a subgrafos esparsos de G. Desta maneira, uma boa escolha de subgrafo esparso de G pode gerar uma solução viável para o problema original. Na primeira versão desse algoritmo, discutida em (BONATES *et al.*, 2011), os grafos inicialmente completos, ou seja, com todos os vértices ligados entre si, são substituídos pelas árvores geradoras mínimas correspondentes gerando uma redução significativa no tempo de execução.

4.2 Limitação

Em (BONATES *et al.*, 2011), é apresentado um procedimento de limitação do algoritmo *Branch-and-Bound* que tem como objetivo determinar uma quantidade mínima de arestas necessárias para obter uma solução viável a partir de uma solução parcial do problema. Para tanto, usa-se o lema (4.1) apresentado naquele trabalho.

Lema 4.1. *Sejam $G = (V, A)$ um grafo, com $|V| = n \geq k$ vértices, e S uma solução parcial para uma instância do PAkC sobre G. Seja $I = \{C_1, C_2, ..., C_t\}$ o conjunto das t componentes inviáveis de S, isto é, as componentes de S que possuem menos de k vértices. A seguinte*

expressão fornece um limite inferior para o número mínimo de arestas necessárias para se construir uma solução viável S^* a partir de S:

$$N(I) = t - \left\lfloor \frac{\sum_{i=1}^{t} |C_i|}{k} \right\rfloor .$$

Fazendo uso desse conhecimento é possível chegar a um limite inferior para o custo de uma solução viável F com as $N(I)$ arestas de menor custo de G. Além da vantagem de ter uma estimativa da solução final para o PAkC antes mesmo de iniciar o algoritmo, esse limite inferior é utilizado para realizar podas na árvore de *Branch-and-Bound*.

4.3 Ordem de Ramificação

Como citado anteriormente, é necessário definir a ordem de prioridade segundo a qual as arestas serão analisadas. Dependendo da aresta que é ramificada primeiro, é possível fixar arestas de ligação e arestas incidentes a vértices com um único vizinho. Dessa maneira, o algoritmo gera soluções completas que poderão ser utilizadas como um limite superior para outros nós da árvore de *Branch-and-Bound*. Analisando a heurística LEF fica nítido como a solução final é sensível à ordem de seleção na qual as arestas são submetidas, principalmente se existir em arestas com o mesmo custo.

5 CONTRIBUIÇÕES PARA O PAKC

Nesse capítulo, descrevemos as contribuições propostas para algumas etapas do algoritmo *Branch-and-Bound*. Propomos, também, duas novas heurísticas para o PAkC.

5.1 Heurística de Árvores Disjuntas de Prim

Muitas das heurísticas existentes para o PAkC geram dentro de uma solução viável algumas árvores com mais de k vértices. Gerar uma solução viável onde os vértices estão distribuídos de maneira mais homogênea entre as componentes pode proporcionar uma redução no número de arestas necessárias para formar uma solução e, com isso, é possível ocorrer uma redução no custo da floresta. Nessa heurística buscamos fazer essa distribuição dos vértices entre as componentes da solução.

Seja um grafo inicial $G = (V, E)$. Aplicaremos o algoritmo de Prim sobre cada vértice do grafo G da seguinte maneira. O algoritmo de Prim terá início a partir de um vértice $i \in V$ e será interrompido quando a árvore que está sendo produzida possuir exatamente k vértices. A árvore de tamanho k gerada a partir do vértice i será denotada por T_i. Aplicando o algoritmo dessa maneira sobre cada vértice $i \in V$, teremos um conjunto de árvores T_i denotado por $\mathcal{T}$. Após a construção dessas árvores, aplica-se uma formulação em programação inteira, em que um conjunto de árvores disjuntas será selecionado. É possível que exista um conjunto de vértices que não fazem parte de nenhuma árvore T_i selecionada, com isso esses vértices devem ser ligados a uma das árvores por meio da aresta incidente de menor custo.

A variável binária x_j corresponde à inclusão ou descarte da árvore $T_j \in \mathcal{T}$. A soma dos custos das arestas de $T_j \in \mathcal{T}$ é igual a c_j. Para cada vértice i do grafo G é constituída uma variável binária y_i. Tal variável representa os vértices que ficaram isolados na solução parcial. A variável binária z_{ij} representa se um vértice $i \in V$ será ligado a árvore $T_j \in \mathcal{T}$. A constante w_{ij} é o custo da aresta de menor custo que liga o vértice $i \in V$ a árvore $T_j \in \mathcal{T}$.

(DP) minimizar $\sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} z_{ij}$

sujeito a:

$$y_i + \sum_{j:i\in T_i} x_j = 1, \ \ \forall\, i \in V \tag{5.1}$$

$$\sum_{\{i,j\}\in E} z_{ij} = y_i, \ \ \forall\, i \in V \tag{5.2}$$

$$z_{ij} \leq x_j, \ \ \forall\, i,j \in V \tag{5.3}$$

$$x_j \in \{0,1\}, \ \ j = 1,...,n \tag{5.4}$$

$$y_i \in \{0,1\}, \ \ i = 1,...,n \tag{5.5}$$

$$z_{ij} \in \{0,1\}, \ \ i = 1,...,n \ \ j = 1,...,n. \tag{5.6}$$

A restrição (5.1) determina que se um vértice $i \in V$ não está em, exatamente, uma das árvore $T_j \in \mathcal{T}$ selecionadas, a variável binária y_i deve receber valor igual a 1. A restrição (5.2) estabelece que cada vértice i que está isolado, ou seja, $y_i = 1$, deve necessariamente unir-se a alguma das árvores selecionadas. A restrição (5.3) garante que um vértice $i \in V$ que ficou isolado poderá unir-se apenas a alguma das árvores $T_j \in \mathcal{T}$ selecionadas.

A intuição por trás do uso desse modelo é tentar distribuir de maneira homogênea os vértices nas árvores, no entanto a ligação dos vértices isolados às árvores selecionadas não garante esse equilíbrio das cardinalidades. Em determinados grafos o número de árvores de cardinalidade k disjuntas é muito inferior se comparado ao número desejado que é de $\lfloor \frac{n}{k} \rfloor$. Tal circunstância gera um elevado número de nós isolados, que por sua vez, são sujeitos a unir-se a alguma das árvores selecionadas produzindo poucas árvores com muitos vértices. A Figura (12) ilustra como um grafo com um pequeno número de árvores disjuntas gera uma solução com um alto valor de função objetivo.

Utilizando a instância da Figura (10), temos que a heurística gera as árvores da Figura (11).

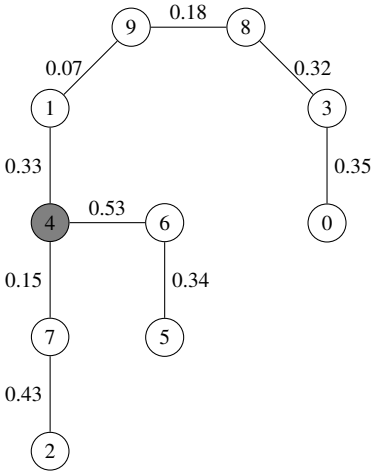

Figura 10 – AGM da Instância $n = 10$ e $k = 3$ e Centróide.

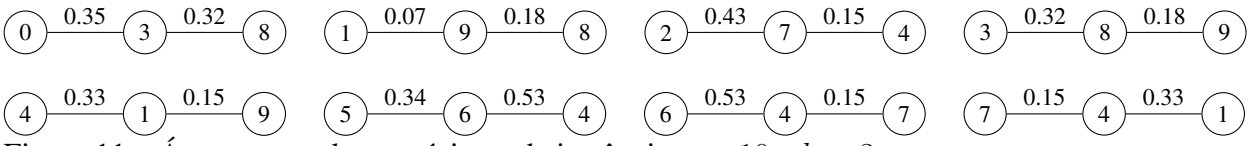

Figura 11 – Árvores geradoras mínimas da instância $n = 10$ e $k = 3$.

A primeira árvore selecionada para fazer parte da solução é a gerada a partir do vértice 1, que é idêntica à árvore gerada a partir do vértice 8 e, também 9, com $V_1 = \{1, 9, 8\}$. A segunda é a árvore gerada a partir do 6 e possui $V_6 = \{6, 4, 7\}$.

As Figuras (12) e (13) mostram, respectivamente, o resultado gerado pela heurística de árvores disjuntas de Prim com custo total igual a 2,63 e um solução gerada a partir de uma árvore geradora mínima do grafo inicial com custo igual a 2,37.

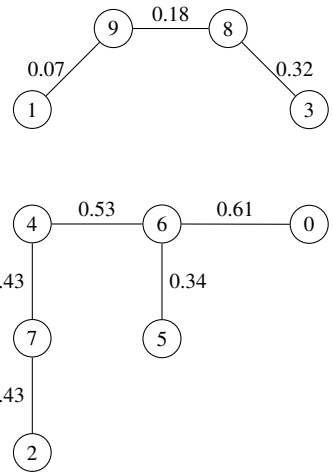

Figura 12 – Solução da instância $n = 10$ e $k = 3$ gerada pelo modelo (DP).

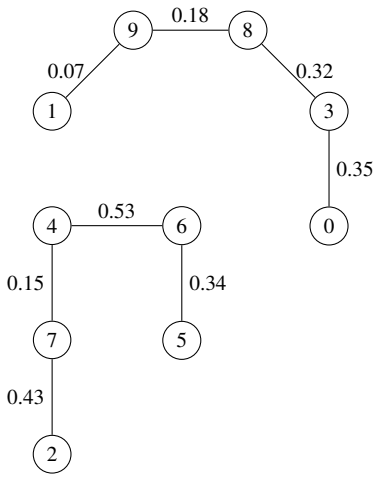

Figura 13 – Solução gerada a partir de uma AGM da instância $n = 10$ e $k = 3$.

Sob esse entendimento, o modelo proposto pode ser melhor explorado em trabalhos futuros a fim de se encontrar um recurso que diminua o número de nós isolados. Outra forma de melhorar esse modelo é buscar uma maneira mais eficiente de unir os vértices isolados a outras componentes ou até mesmos uni-los entre si, caso o número de vértices isolados seja superior ou igual a k. Uma possível variação na forma de unir os vértices isolados a outras componentes pode ser facilmente realizada pelo fato de termos a possibilidade de recuperar todas as arestas que são incidentes a cada vértice do grafo.

5.2 Heurística de Cobertura Mínima para Árvores k-Capacitadas

O PAkC é um caso especial do problema de cobertura mínima de arestas, onde o objetivo é construir um conjunto de arestas que seja capaz de cobrir todos os vértices do grafo a um custo mínimo. Para dar início ao algoritmo proposto nessa seção, partimos de um grafo completo $G = (V, E)$. Cada vértice é considerado uma árvore com cardinalidade 1 e a cada iteração as árvores serão conectadas a fim de gerar componentes maiores, desde que a aresta a ser inserida não gere ciclo.

Se uma árvore contém k vértices ou mais será considerada já coberta, caso contrário será considerada não coberta, pois ainda é uma componente inviável. Nesse processo de cobertura o algoritmo seleciona a aresta de menor custo que não gera ciclo, em que cada extremidade pertence a uma árvore ainda não coberta.

Ao final desse processo, poderá existir uma ou mais árvores inviáveis (não cobertas) que não possui nenhuma aresta apta a uni-la a uma outra componente inviável. Nesse caso, essa árvore deve ser conectada a uma árvore já coberta fazendo uso da aresta de menor custo que não

gera ciclo, com um extremo em cada uma das componentes. No caso de não existir mais árvores a serem cobertas, temos uma solução completa para o PAkC.

Analisando os passos desse algoritmo fica claro que, depois da soma mínima de custo das arestas, o principal objetivo é priorizar a união de árvores inviáveis com o interesse em não gerar componentes que ultrapassem excessivamente o tamanho mínimo de k vértices em cada árvore.

Seguindo a mesma instância dos exemplos anteriores temos na Figura (14) a representação da solução gerada por esse algoritmo. Essa heurística gera uma solução com custo total de 2,10, enquanto a solução gerada a partir de uma árvore geradora do grafo inicial produz uma floresta com custo igual a 2,37.

A heurística de cobertura mínima para árvores k-capacitadas alcançou, nesse exemplo, uma solução com 3 árvores, onde todas as árvores possuem k vértices ou um número próximo a esse. Logo, a heurística parece realmente construir soluções que condizem com a intuição por trás do seu funcionamento.

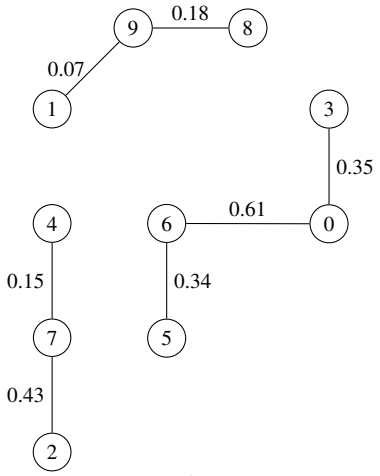

Figura 14 – Um solução da instância $n = 10$ e $k = 3$.

No entanto, é evidente que existirão casos onde será mais vantajoso manter uma árvore com mais vértices e fazendo uso de arestas de menor custo. Por exemplo, se a aresta $\{6, 0\}$ tivesse um custo maior ou igual a 0,86 seria mais vantajoso dispor da solução representada na Figura (13). No algoritmo (5) temos com mais detalhes como essa heurística constrói uma solução viável para o PAkC.

Algoritmo 5: Heurística de Cobertura Mínima para Árvores k-Capacitadas

Entrada: E(conjunto de arestas), V(conjunto de vértices), $k \in \mathbb{Z}_+$.
Saída: S(conjunto de arestas da floresta k-capacitada).

início
  $S \leftarrow \emptyset$
  para $\forall v \in V$ **faça**
    $T_v \leftarrow \{v\}$
    $\mathcal{T} \leftarrow T_v$
  fim
  $E \leftarrow ordenar(E)$
  Comentário: Ordenar em ordem crescente de custo das arestas.
  para $\forall e = \{i, j\} \in E$ **faça**
    se $i \in T_v \in \mathcal{T}$, *onde* $|T_v| = 1$ ***e*** $j \in T_u \in \mathcal{T}$, *onde* $|T_u| = 1$ ***e*** $T_v \neq T_u$ **então**
      $T_z \leftarrow T_v \cup T_u$
      $\mathcal{T} \leftarrow \mathcal{T} \cup T_z - T_v - T_u$
      $S \leftarrow S \cup \{e\}$
    fim
  fim
  para $\forall e = \{i, j\} \in E$ **faça**
    se $e \notin S$ ***e*** $i \in T_v \in \mathcal{T}$, *onde* $|T_v| < k$ ***e*** $j \in T_u \in \mathcal{T}$, *onde* $|T_u| < k$ ***e*** $T_v \neq T_u$ ***e*** $F \cup \{e\}$ *não contém ciclos* **então**
      $T_z \leftarrow T_v \cup T_u$
      $\mathcal{T} \leftarrow \mathcal{T} \cup T_z - T_v - T_u$
      $S \leftarrow S \cup \{e\}$
    fim
  fim
  para $\forall T_v \in \mathcal{T}$ **faça**
    se $|T_v| < k$ **então**
      Ache a aresta $e = \{i, j\}$ de menor custo, onde $i \in T_v$ **e** $j \in T_u \in \mathcal{T}$ **e** $T_v \neq T_u$ **e** $F \cup \{e\}$ não contém ciclos
      $T_z \leftarrow T_v \cup T_u$
      $\mathcal{T} \leftarrow \mathcal{T} \cup T_z - T_v - T_u$
      $S \leftarrow S \cup \{e\}$
    fim
  fim
fim
retorna S

5.3 Contribuições para o Algoritmo *Branch-and-Bound*

As melhorias propostas para algoritmo *Branch-and-Bound* apresentado em (BONATES *et al.*, 2011) são divididas em quatro grupos: o primeiro, compara funções de avaliação para podas na árvore de *Branch-and-Bound*; o segundo, avalia como diferentes ordens de seleção da aresta de ramificação pode contribuir para o aumento no número de nós podados na árvore de

Branch-and-Bound; o terceiro, analisa o uso de outras arestas do grafo original além das arestas da árvore geradora mínima; por fim, faremos a análise de possíveis pré-processamentos sobre o conjunto de arestas iniciais do algoritmo. Na implementação discutida nesse trabalho, assim como em (BONATES *et al.*, 2011), uma árvore geradora mínima é utilizada para reduzir o grafo de entrada do algoritmo *Branch-and-Bound*, sendo essa produzida pelo algoritmo de Kruskal.

5.3.1 Função de Avaliação

A função de avaliação tem como finalidade gerar uma estimativa do custo da função objetivo da melhor solução viável constituída a partir de uma solução parcial, associada a um nó na árvore *Branch-and-Bound*. Com isso, é possível realizar uma poda na árvore de *Branch-and-Bound*, se o valor da estimativa for superior ao custo da melhor solução viável encontrada. Sendo conhecido o número de arestas ainda necessárias para construir uma solução viável, representado por $N(I)$ conforme discutido na Seção (4.2) do cápitulo anterior, é possível gerar diferentes estimativas para uma solução parcial.

A primeira função de avaliação a ser estudada, apresentada em (BONATES *et al.*, 2011), seleciona as $N(I)$ arestas de menor custo da árvore de entrada e as utilizam para estimar o custo da função objetivo da melhor solução completa associada a solução parcial que está sendo avaliada. Denotaremos por ϕ_1 essa primeira função de avaliação. A partir ϕ_1, propomos duas variações na escolha das $N(I)$ arestas a fim de gerar uma estimativa mais próxima ao custo da função objetivo de uma solução completa.

A primeira variação considerada para a função de avaliação ϕ_1 é substituir as $N(I)$ arestas de menor custo pelas $N(I)$ arestas de menor custo que ainda são candidatas a participar da solução, ou seja, arestas que ainda não foram fixadas ou descartas na solução parcial. Esta nova função de avaliação denotaremos por ϕ_2. Da maneira apresentada em (BONATES *et al.*, 2011), temos que mesmo que estejamos no primeiro nó da árvore de *Branch-and-Bound*, provavelmente já inserimos alguma aresta incidente a um vértice com um único vizinho ou alguma aresta de ligação. Assim, possivelmente, as $N(I)$ arestas selecionadas não completam uma solução. Por outro lado, selecionar as $N(I)$ arestas de menor custo que ainda são candidatas gera no melhor caso a solução ótima associada aquele nó, isto é, se ao inserir as $N(I)$ arestas candidatas de menor custo produzirmos uma solução viável, temos a melhor solução associada a solução parcial. Assim, podemos podar o nó de forma prematura fixando todas as $N(I)$ arestas candidatas. No pior caso, temos uma estimativa mais realista do custo da função objetivo da

solução completa.

Dado que somente as arestas que ainda são candidatas são utilizadas para calcular o limite inferior, aplicamos a condição existente na heurística LEF, onde uma aresta candidata deve ser incluída na solução se suas extremidades estão em componentes diferentes e, pelo menos uma das extremidades pertence a uma componente inviável; no caso contrário, essa aresta deve ser descartada. Assim, temos que a segunda variação da função de avaliação consiste em utilizar as $N(I)$ arestas de menor custo que ainda são candidatas e que são incidentes a pelo menos uma componente inviável da solução parcial, denotada por ϕ_3.

5.3.2 Ordem de Seleção das Arestas de Ramificação

O algoritmo *branch-and-bound* gera um número exponencial de nós na árvore de *branch-and-bound*. Em grafos com um grande número de vértices, avaliar todas as soluções possíveis pode ser impraticável. A partir dessa premissa, pode-se considerar que determinado critério de seleção de arestas para ramificação gere uma árvore com altura reduzida, que aliado a uma boa função de avaliação produza podas substanciais na árvore de *branch-and-bound*.

Nesse segundo grupo de alterações no algoritmo *Branch-and-Bound*, iremos experimentar diferentes ordens de seleção das arestas na tentativa de reduzir o número de nós gerados na árvore de *Branch-and-Bound*. O algoritmo *branch-and-bound*, em sua primeira versão, segue uma ordem crescente de custo na seleção das arestas. A ideia utilizada como comparativo para esta ordem tem base no centróide da árvore de entrada.

Definição 5.1. *Centróide é um vértice de uma árvore que, se removido, a divide em uma floresta, de modo que qualquer árvore dessa floresta tenha no máximo metade do número de vértices da árvore original.*

O primeiro passo dessa ordenação com base no centróide é dar rótulo 0 (zero) a todas as arestas incidentes ao centróide do grafo original, que aqui chamaremos de c. Continuando com o processo de rotulação, as arestas incidentes aos vértices vizinhos a c recebem rótulo 1, as arestas adjacentes as arestas de rótulo 1 e que não possuem rótulo 0 recebem rótulo 2, e assim por diante. Seja r o maior rótulo dado a alguma aresta. Temos, portanto, rótulos 0, 1, ..., r.

Diante de um vetor de arestas rotuladas, define-se quais rótulos devem ter prioridade na sequência de ramificação. Primeiramente, ramificam-se todas as arestas com rótulo 0 (zero), com o intuito de organizar a árvore de *Branch-and-Bound* em uma estrutura que permita reduzir

o número de nós necessários para chegar a cada solução completa. Quando uma aresta incidente ao centróide é fixada fora da solução parcial é possível que gere muitas arestas de ligação e arestas incidentes a vértices com um único vizinho. No caso onde as arestas de rótulo 0 não são selecionadas para compor a solução parcial, temos que a árvore de entrada é dividida em subárvores com no máximo metade do tamanho da árvore original.

O centróide divide a árvore de entrada em subárvores menores, no entanto encontrar o centroíde nessas subárvores não é uma tarefa simples. Usamos os rótulos das arestas para buscar as arestas incidentes aos possíveis centróides das subárvores ou arestas próximas a estas. Se uma subárvore possui arestas com rótulos de 1 a 10, por exemplo, estipulamos que as arestas com rótulo 5 sejam incidentes ao centroíde dessa subárvore. Com base nessa ideia, para as subárvores geradas a partir da remoção de c, atribui-se ordem 1 as possíveis arestas incidentes a esses novos centróides, e assim em diante. Por exemplo, se o valor de r for 20 as próximas arestas que serão ramificadas, depois daquelas com rótulo zero, serão as arestas com rótulo 10, porque possuem ordem 1. Depois virão as arestas com rótulo 5 e 15, porque possuem ordem 2.

Na Figura (15), temos um exemplo de como fica organizada a ordem de ramificação das arestas segundo o rótulo que elas recebem. Em um grafo com arestas com rótulo até 20, supomos que as arestas com rótulo igual a mediana do intervalo entre o menor e o maior rótulo são incidentes aos novos centroídes.

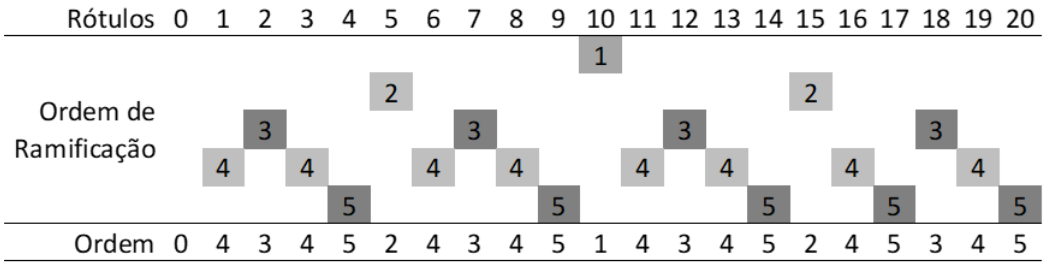

Figura 15 – Ordem de ramificação atribuída a cada rótulo.

Ordenando os rótulos de acordo com a ordem atribuída a eles, tem-se a sequência de ramificação representada na Tabela (1).

0	10	5	15	2	7	12	18	1	3	6	8	11	13	16	19	4	9	14	17	20

Tabela 1 – Rótulos ordenados segundo a ordem de ramificação.

No processo de atribuir ordem aos rótulos, temos que alguns rótulos recebem a mesma ordem. Por essa razão, não é necessário se importar com a ordem relativa de rótulos que

possuem a mesma ordem, pois estes estão em subárvores diferentes. Por exemplo, os rótulos de ordem 3 (2, 7, 12 e 18) podem estar em qualquer ordem entre si. O importante é que eles venham depois dos rótulos com ordem 2 e antes dos rótulos com ordem 4.

Apresentaremos um exemplo de como atribuímos a ordem de ramificação aos rótulos das arestas fazendo uso de um grafo $G - (V, E)$, onde $|V| = 10$ e com $k = 3$. Após ordenar as arestas de forma crescente de custos, temos o vetor de arestas representado na Tabela (2).

Posição	0	1	2	3	4	5	6	7	8
Aresta	{9,1}	{7,4}	{9,8}	{8,3}	{4,1}	{6,5}	{3,0}	{7,2}	{6,4}
Custo	0.07	0.15	0.18	0.32	0.33	0.34	0.35	0.43	0.53

Tabela 2 – Arestas ordenadas em ordem crescente de custos.

Esse conjunto de arestas pode ser representado pelo seguinte grafo, onde o vértice 4 é um dos centróides dessa árvore.

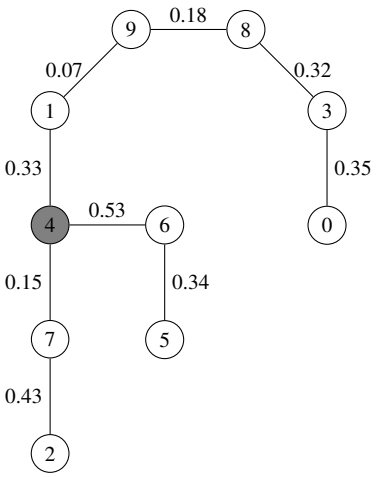

Figura 16 – AGM da Instância $n = 10$ e $k = 3$ e Centróide.

O primeiro passo para construir um vetor de arestas, que também definirá a ordem de ramificação para árvore de *Branch-and-Bound*, é rotular os vértices segundo a sua proximidade ao centróide da árvore aplicando uma busca em largura para rotular as arestas em 0, 1, 2, e assim por diante, onde cada valor é a distância de cada aresta até o centróide. Arestas incidentes ao centróide terão rótulo 0. Após esse processo, tem-se os seguintes rótulos para cada aresta desse exemplo (Tabela 3).

Arestas	{9,1}	{7,4}	{9,8}	{8,3}	{4,1}	{6,5}	{3,0}	{7,2}	{6,4}
Rótulos	1	0	2	3	0	1	4	1	0

Tabela 3 – Rótulos das arestas.

O segundo passo consiste em gerar uma ordem para os rótulos. Nesse exemplo, 4 é o maior rótulo obtido no passo anterior. Diante disso, um vetor $ordem$ de tamanho 5 será criado. A posição i deste vetor, $ordem[i]$, irá dizer a ordem que as arestas rotuladas com i terão na ordem de ramificação. As arestas com rótulo 0 serão ramificadas antes das demais. Não depende de cálculo algum: arestas vizinhas ao centróide vão ser ramificadas antes das demais. Logo, a posição 0 do vetor $ordem$ receberá valor 0, ou seja, se uma aresta recebe ordem 0 significa que esta tem prioridade máxima na ordem de ramificação.

Rótulos	0	1	2	3	4
Ordem	0	2	1	2	3

Tabela 4 – Ordem dos rótulos.

Nesse exemplo, as arestas que possuem rótulo 2 serão as próximas a serem ramificadas (Tabela 4). Somente depois que todas elas forem ramificadas é que as arestas de rótulos 1 e 3 também serão ramificadas e somente depois delas é que as demais arestas serão ramificadas, sempre seguindo a ordem dada por $ordem[i]$. Adiante tem-se a ordem de ramificação das arestas (Tabela 5).

Arestas	{9,1}	{7,4}	{9,8}	{8,3}	{4,1}	{6,5}	{3,0}	{7,2}	{6,4}
Ordem	2	0	1	2	0	2	3	2	0

Tabela 5 – Ordem das arestas.

Portanto, esse será o novo vetor de arestas é representado na Tabela (6).

Posição	0	1	2	3	4	5	6	7	8
Aresta	{7,4}	{4,1}	{6,4}	{9,8}	{9,1}	{8,3}	{6,5}	{7,2}	{3,0}
Custo	0.15	0.33	0.53	0.18	0.07	0.32	0.34	0.43	0.18

Tabela 6 – Arestas ordenadas pela ordem de ramificação dos rótulos.

Com base nessa ideia de seleção das arestas de ramificação, é possível variar a ordem de seleção mais uma vez. Assim constitui-se uma terceira ordem, onde as arestas seguem em ordem crescente de custo, no entanto quando duas arestas possuem o mesmo custo, a aresta com menor ordem é ramificada primeiro.

5.3.3 Conjunto Inicial de Arestas

O algoritmo *Branch-and-Bound* pode ter como entrada uma árvore geradora com o objetivo de ter um grafo inicial esparso. No entanto, é possível que existam arestas que não

são utilizadas para constituir uma árvore geradora e que fazem parte da solução ótima, logo essa solução não é um subgrafo de uma árvore geradora de G.

Na implementação do algoritmo *Branch-and-Bound*, o algoritmo de Kruskal foi utilizado para gerar a árvore geradora mínima. Diante disso, propomos considerar não somente as arestas da árvore geradora mínima, mas também todas as arestas que foram cogitadas durante o algoritmo de Kruskal como ponto de partida para o algoritmo *Branch-and-Bound*. Dessa maneira, é possível avaliar as arestas de um ciclo e, eventualmente, decidir pelas arestas que mesmo não sendo as de menor custo do ciclo geram uma solução completa com menor custo de função objetivo.

Outro conjunto inicial de arestas possível é dado pela união do conjunto de arestas da árvore geradora mínima com as arestas selecionadas pela heurística de cobertura mínima para árvores k-capacitadas citadas na Seção (5.2).

5.3.4 Pré-processamento

Dentro de uma busca em largura ou profundidade do tipo *Branch-and-Bound* é fundamental ter um bom limite superior que permita gerar podas significativas na árvore. De modo geral, o limite superior é dado pelo custo da melhor solução encontrada até o momento da avaliação de cada nó.

Diante do fato de que o algoritmo *Branch-and-Bound* parte de um conjunto vazio de arestas e que, a cada iteração novas arestas são acrescentadas, é razoável dizer que o algoritmo percorre vários nós da árvore de *Branch-and-Bound* até que a primeira solução viável seja encontrada e, assim possa ser usada como limite superior. Computacionalmente, completar uma solução parcial usando o LEF para gerar um limite para função objetivo pode exigir muito esforço quando executado nos primeiros níveis da árvore. Com isso, é legítimo que ter um bom limite superior no inicio do processo do algoritmo gere grandes podas na árvore de *Branch-and-Bound* e uma redução significativa no tempo de execução. Com essa motivação, propomos utilizar o custo da função objetivo gerado pelo algoritmo HEF como primeiro limite superior em cada instância executada.

6 RESULTADOS COMPUTACIONAIS

Os testes computacionais discutidos nesse trabalho consideram o conjunto de instâncias do PAkC usado em (BONATES *et al.*, 2011). As instâncias consistem em grafos completos, onde cada vértice é associado ao um ponto no plano, obtido aleatoriamente, e o custo da aresta é dado pela distância Euclidiana entre cada par de vértices. Naquele trabalho, as instâncias são divididas em dois grupos denominados de *Grupo 1* e *Grupo 2*. Em cada instância temos o número de vértices dado por n e o valor de k.

Nesse capítulo, apresentamos experimentos computacionais realizados com base nas melhorias ao algoritmo *Branch-and-Bound* e as heurísticas propostas no Capítulo 5. Para tanto, apenas as instâncias do Grupo 2 foram utilizadas. O Grupo 1 possuim instâncias com o número de vértices variando entre 10 e 50. Com esse número de vértices não conseguimos obter variações no custo da função objetivo nos diferentes experimentos explorados nesse trabalho, por essa razão optamos por não apresentar tais resultados.

Seja um grafo $G = (V, E)$. Todos os testes desse trabalho tomam como ponto de partida um subgrafo gerador $H = (V, E')$ de G, onde $E' \subseteq E$. Em muitos utilizamos uma árvore geradora mínima.

6.1 Características do ambiente de computação

Os testes foram executados em um computador com processador Intel Core i7 de oito núcleos de 3.40 GHz, 16.0 GB de memória RAM DDR3 1333 MHz e sistema operacional Ubuntu Linux 14.04 LTS de 64 bits. Para os modelos de programação inteira fizemos uso do pacote de otimização ILOG CPLEX versão 12.6.1. Além disso, utilizamos a biblioteca Concert Technology para desenvolvimentos na linguagem C++. Todas as heurísticas e algoritmos foram implementados na linguagem C++, cujos os tempos de execução foram limitados em 5 segundos.

6.2 Análise dos Resultados

Os experimentos computacionais a seguir utilizam uma árvore geradora mínima T como ponto de partida ou, em alguns casos, um subgrafo que contém essa árvore geradora. O modelo de programação inteira (ND) foi aplicado sobre uma árvore geradora mínima de cada instância, a fim de termos o custo da melhor solução para o PAkC que é um subconjunto de uma árvore geradora mínima selecionada.

Na Tabela (7), temos na coluna *Valor da Função Objetivo* o custo da melhor solução encontrada até o tempo de 300 segundos ser alcançado ou até encontrar a solução ótima que é subgrafo da árvore de entrada. Na coluna *Status*, temos se o mdelo (ND) conseguiu completar a busca pela solução ótima do PAkC gerada a partir da árvore de entrada ou se teve que ser interrompido obtendo apenas uma solução viável.

n	k	Valor da Função Objetivo	Status	Tempos de Execução (s)
100	5	6.37	Ótimo	4.51
120	5	6.59	Ótimo	19.59
150	5	8.02	Ótimo	28.52
200	5	8.86	Viável	300.00
300	5	11.49	Viável	300.00
400	5	13.39	Viável	300.00
500	5	15.76	Viável	300.00
100	10	6.74	Ótimo	4.36
120	10	7.24	Ótimo	21.52
150	10	8.92	Viável	300.00
200	10	10.09	Viável	300.00
300	10	12.86	Viável	300.00
400	10	15.35	Viável	300.00
500	10	17.88	Viável	300.00
120	20	8.01	Ótimo	11.71
150	20	8.89	Ótimo	63.40
200	20	10.83	Viável	300.00
300	20	13.57	Viável	300.00
400	20	16.20	Viável	300.00
500	20	18.74	Viável	300.00
150	40	9.61	Ótimo	13.01
200	40	11.37	Viável	300.00
300	40	14.34	Viável	300.00
400	40	16.86	Viável	300.00
500	40	19.49	Viável	300.00

Tabela 7 – Valores do modelo ND obtidos a partir da AGM.

De acordo com a Tabela (7), observamos que mesmo com um tempo de resolução de 5 minutos não foi possível encontrar uma solução ótima nas instâncias com mais de 150 vértices. A partir desse ponto, temos a importância do uso de outros métodos como o algoritmo *Branch-and-Bound*, que partindo de uma árvore geradora mínima do grafo inicial G consegue atingir valores de função objetivo próximos ao custo da solução ótima que é subgrafo de uma árvore geradora mínima de G.

6.2.1 Função de Avaliação

Nesse primeiro grupo de testes foram analisadas as duas funções de avaliação propostas nesse trabalho em comparação com a função apresentada em (BONATES *et al.*, 2011), discutidas na Seção (5.3.1) do capítulo anterior. Consideramos o algoritmo *Branch-and-Bound* de (BONATES *et al.*, 2011) no teste $A0$. Este teste representa a função de avaliação ϕ_1 dada pela soma dos custos das $N(I)$ arestas de menor custo da árvore geradora mínima. O teste $A1$ representa a função de avaliação ϕ_2 que utiliza como limite inferior a soma dos custos das $N(i)$ arestas de menor custo que ainda são candidatas a participar da solução. Finalmente, no teste $A2$ temos a função de avaliação ϕ_3, onde o algoritmo LEF é aplicado sobre a solução parcial a fim de completar a solução com $N(I)$ arestas.

Na Tabela (8), temos na coluna *Número de Nós Avaliados* a quantidade de nós da árvore de *Branch-and-Bound* que são avaliados durante a execução do algoritmo até que o tempo de execução limite seja atingido ou até que não existam mais nós para serem avaliados. Na coluna *Número de Nós Prematuros*, temos o número de nós que são podados de forma prematura, ou seja, quando as $N(I)$ arestas selecionadas nas funções de avalição ϕ_2 ou ϕ_3 são fixadas na solução parcial e geram uma solução viável.

Comparando os resultados gerados pelos testes $A1$ e $A2$ em relação ao teste $A0$ na Tabela (8), verificamos que o $A1$ obteve o menor custo de função objetivo em 7 das 25 instâncias e $A2$ teve melhor desempenho em 9. A redução no número de nós da árvore de *Branch-and-Bound* foi de 56% em $A1$ e de 58% em $A2$. Além disso o número de nós podados de forma prematura no teste $A2$ é mais que o dobro que no teste $A1$. O tempo de execução teve redução de 34,4% no teste $A1$ e 34,7% em $A2$. Portanto, a função de avaliação utilizada em $A2$ teve um melhor desempenho e o tempo a mais utilizado em cada iteração é mais do que compensado pela redução do número de nós avaliados. Diante dessas observações temos um argumento convincente a favor de adotarmos a função de avaliação de $A2$ como um passo do algoritmo.

6.2.2 Ordem de Seleção das Arestas de Ramificação

Fazendo uso do teste com melhor desempenho da Tabela (8), estabelecemos um novo comparativo. Lembre que no teste $A2$, as arestas são selecionadas em ordem crescente de custos. Realizamos dois outros testes: em $A3$ a ordem de seleção segue a ideia que se baseia

		Valor da Função Objetivo			Número de Nós Avaliados			Nº de Nós Prematuros		Tempos de Execução (s)		
n	k	$A0$	$A1$	$A2$	$A0$	$A1$	$A2$	$A1$	$A2$	$A0$	$A1$	$A2$
100	5	6.48	**6.37**	**6.37**	203289	49084	46524	1	1	5.00	2.48	2.40
120	5	6.64	**6.63**	**6.63**	159999	109955	104095	4	11	5.00	5.00	5.00
150	5	8.21	**8.15**	**8.15**	221470	131703	126451	4	6	5.00	5.00	5.00
200	5	9.12	9.00	**8.99**	249324	134818	118425	8	9	5.00	5.00	5.00
300	5	11.87	11.87	11.87	158661	118737	110621	4	4	5.00	5.00	5.00
400	5	14.09	**14.03**	**14.03**	138922	108055	97701	5	5	5.00	5.00	5.00
500	5	16.38	16.38	16.38	88199	87331	79451	0	2	5.00	5.00	5.00
100	10	6.74	6.74	6.74	11687	473	339	0	0	0.31	0.02	0.01
120	10	7.24	7.24	7.24	103234	2735	2661	4	6	3.72	0.17	0.16
150	10	8.92	8.92	8.92	246689	63557	60319	0	0	5.00	2.52	2.42
200	10	10.13	10.13	10.13	237820	127295	124927	2	10	5.00	5.00	5.00
300	10	13.11	**13.00**	**13.00**	155549	110485	108705	5	5	5.00	5.00	5.00
400	10	15.64	**15.59**	**15.59**	154847	138221	126345	0	4	5.00	5.00	5.00
500	10	18.20	**18.16**	**18.16**	114986	65331	63023	0	6	5.00	5.00	5.00
120	20	8.01	8.01	8.01	2626	71	69	0	0	0.08	0.00	0.00
150	20	8.89	8.89	8.89	5128	174	158	0	2	0.12	0.00	0.00
200	20	10.83	10.83	10.83	270031	25775	23882	3	3	5.00	1.29	1.22
300	20	13.57	13.57	13.57	173527	97753	96531	2	4	5.00	5.00	5.00
400	20	16.28	16.24	**16.20**	203506	121161	115517	0	3	5.00	5.00	5.00
500	20	18.62	18.62	18.62	119335	71229	70547	4	4	5.00	5.00	5.00
150	40	9.61	9.61	9.61	204	45	45	1	1	0.00	0.00	0.00
200	40	11.37	11.37	11.37	13650	7	7	1	1	0.32	0.00	0.00
300	40	14.34	14.34	14.34	158409	1622	1580	1	4	5.00	0.16	0.16
400	40	16.90	16.90	16.90	191171	149513	149751	0	3	5.00	5.00	5.00
500	40	19.48	19.48	19.48	124675	58269	58275	0	4	5.00	5.00	5.00

Tabela 8 – Resultados das variações realizadas na função de avaliação dos nós da árvore de *Branch-and-Bound*.

no centróide da árvore para atribuir uma ordem para as arestas; o teste $A4$ as arestas seguem a mesma ordem crescente de custos, no entanto a aresta com menor ordem tem prioridade sobre uma outra aresta de mesmo custo.

Na Tabela (9) seguimos o mesmo padrão da Tabela (8). No entanto, acrescentamos a coluna *Números de Nós Podados*, onde contabilizamos o número de nós da árvore de *Branch-and-Bound* em que a solução parcial possui custo superior ao custo da melhor solução encontrada.

Comparando $A3$ e $A4$ com $A2$ na Tabela (9), verificamos que o $A3$ não gera bons valores na função objetivo, tendo o pior desempenho em todas as instâncias. No entanto, faz o que a ideia inicial propõe gerando um grande número de nós podados e prematuros. Por exemplo, na instância $n = 100$ e $k = 5$ o teste $A3$ poda mais nós que o teste $A2$ e, além disso, enquanto $A3$ gera 16 nós de forma prematura, $A2$ gera apenas 1. O teste $A4$ segue valores médios entre $A2$ e $A3$ em todos os critérios de avaliação, no entanto o acréscimo no tempo e os valores na função objetivo não justificam a adoção dessa ordem de seleção como parte permanente do algoritmo de *Branch-and-bound*. Portanto, é aceitável adotar a ordem de seleção de $A2$ como parte do

algoritmo.

		Valor da Função Objetivo			Número de Nós Avaliados			Número de Nós Podados			Nº de Nós Prematuros			Tempos de Preprocessamento (s)			Tempos de Execução (s)		
n	k	$A2$	$A3$	$A4$	$A2$	$A3$	$A4$	$A2$	$A3$	$A4$	$A2$	$A3$	$A4$	$A2$	$A3$	$A4$	$A2$	$A3$	$A4$
100	5	6.37	6.37	6.37	46524	63429	52303	23092	30026	26083	1	16	6	0.00	0.00	0.00	2.40	3.47	2.78
120	5	**6.63**	6.77	6.64	104095	114030	111619	50598	54737	53877	11	35	12	0.00	0.00	0.00	5.00	5.00	5.00
150	5	**8.15**	8.30	**8.15**	126451	158673	129601	61413	78472	63746	6	15	4	0.00	0.00	0.00	5.00	5.00	5.00
200	5	**8.99**	9.22	9.03	118425	148075	130721	57927	72706	65260	9	29	8	0.00	0.00	0.00	5.00	5.00	5.00
300	5	**11.87**	12.21	**11.87**	110621	93997	112860	52855	36166	54803	4	17	4	0.01	0.01	0.01	5.00	5.00	5.00
400	5	**14.03**	14.37	**14.03**	97701	139117	105155	48687	59074	52413	5	4	5	0.02	0.03	0.03	5.00	5.00	5.00
500	5	**16.38**	16.75	16.39	79451	80093	82889	37148	32670	41264	2	9	2	0.04	0.04	0.05	5.00	5.00	5.00
100	10	6.74	6.74	6.74	339	742	358	166	313	173	0	18	0	0.00	0.00	0.00	0.01	0.03	0.01
120	10	7.24	7.24	7.24	2661	13057	2759	1304	5787	1345	6	11	2	0.00	0.00	0.00	0.16	0.70	0.16
150	10	8.92	8.92	8.92	60319	59234	35105	29880	27244	17338	0	17	0	0.00	0.00	0.00	2.42	2.60	1.47
200	10	**10.13**	10.26	**10.13**	124927	143086	124755	62158	64270	61790	10	15	9	0.00	0.00	0.00	5.00	5.00	5.00
300	10	**13.00**	13.20	**13.00**	108705	113411	105859	53942	44644	51664	5	8	5	0.01	0.01	0.01	5.00	5.00	5.00
400	10	**15.59**	15.70	15.74	126345	113199	141381	63052	55118	70567	4	5	1	0.02	0.03	0.03	5.00	5.00	5.00
500	10	**18.16**	18.54	18.24	63023	59469	64555	31360	29534	32118	6	16	0	0.04	0.04	0.05	5.00	5.00	5.00
120	20	8.01	8.01	8.01	69	135	65	32	60	28	0	5	2	0.00	0.00	0.00	0.00	0.01	0.00
150	20	8.89	8.89	8.89	158	528	334	74	197	163	2	8	2	0.00	0.00	0.00	0.00	0.02	0.00
200	20	10.83	10.83	10.83	23882	77058	27111	11664	35384	13464	3	7	3	0.00	0.00	0.00	1.22	3.70	1.40
300	20	**13.57**	13.59	**13.57**	96531	100442	91315	47545	47851	45130	4	5	3	0.01	0.01	0.01	5.00	5.00	5.00
400	20	**16.20**	16.30	**16.20**	115517	98402	98201	56101	48720	48221	3	7	8	0.02	0.03	0.03	5.00	5.00	5.00
500	20	**18.62**	18.77	18.80	70547	93513	101317	35149	39507	49773	4	7	1	0.04	0.04	0.05	5.00	5.00	5.00
150	40	9.61	9.61	9.61	45	26	31	18	10	13	1	0	1	0.00	0.00	0.00	0.00	0.00	0.00
200	40	11.37	11.37	11.37	7	1	109	3	0	54	1	1	0	0.00	0.00	0.00	0.00	0.00	0.02
300	40	14.34	14.34	14.34	1580	8221	1554	767	3958	756	4	5	3	0.01	0.01	0.01	0.16	0.68	0.17
400	40	16.90	16.87	**16.86**	149751	94562	160202	72996	46149	78700	3	6	3	0.02	0.03	0.03	5.00	5.00	5.00
500	40	**19.48**	19.55	**19.48**	58275	68430	60971	28641	30529	29471	4	13	4	0.04	0.04	0.05	5.00	5.00	5.00

Tabela 9 – Resultados das variações na ordem seleção das arestas ramificação do algoritmo *Branch-and-Bound*.

Nas instâncias em que os testes avaliam todos os nós da árvore de *Branch-and-Bound* e finalizam o algoritmo antes de atingir o limite de tempo, temos que o teste $A3$ encontra a solução ótima avaliando menos nós que $A2$ nas instâncias $n = 150$ e $n = 200$ com $k = 40$. No entanto, nas instâncias $n = 120$ e $n = 150$ com $k = 20$ temos o contrário, de maneira que o teste $A3$ avaliou mais que o dobro de nós que $A2$.

A razão pelo qual o teste $A3$ não consegue obter o desempenho esperado é atribuído ao número de vértices que possuem grau maior que 2. Para ilustrar de maneira mais clara, temos que Figura (17) representa o grafo da árvore geradora mínima da instância $n = 150$ e $k = 20$.

O algoritmo tem o seu desempenho baseado na sua capacidade de fixar arestas. No nó da árvore de *Branch-and-bound* em que a aresta de ramificação é fixada fora da solução parcial, o algoritmo tenta encontrar arestas de ligação geradas a partir dessa decisão. Quando um vértice possui muitos vizinhos, será necessário avaliar cada uma das arestas incidentes a ele e é menos provável conseguir encontrar uma aresta de ligação a partir do descarte de uma aresta.

Na ideia de ordenação das arestas aplicada no teste $A3$, arestas incidentes a um determinado vértice v recebem a mesma ordem de ramificação. Por essa razão, o algoritmo

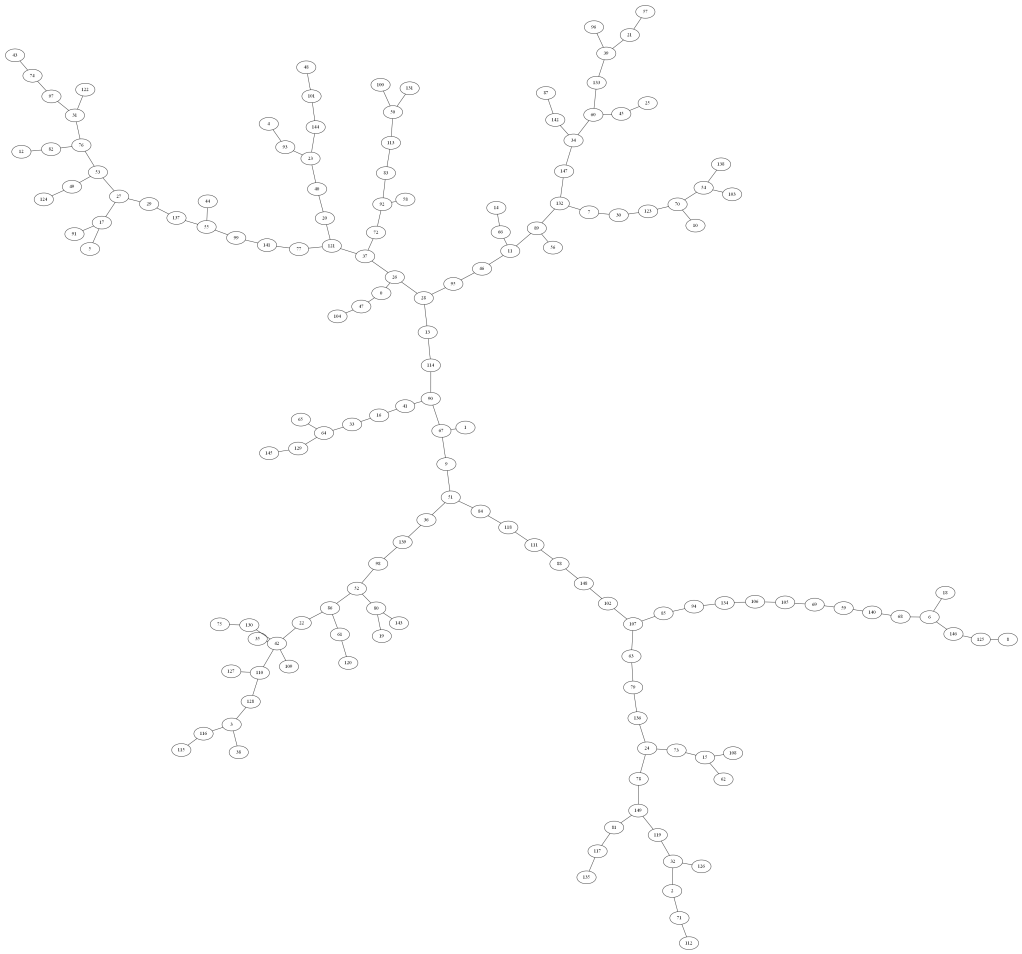

Figura 17 – AGM da instância $n = 150$ e $k = 20$.

primeiramente avalia cada uma das arestas incidentes ao vértice v para então escolher uma outra subárvore para continuar a busca. Assim, em instâncias com muitos vértices com grau maior que 2, o algoritmo utiliza vários nós da árvore de *Branch-and-Bound* para escolher as arestas que irão ligar um vértice a outra componente e por consequência, utiliza um número bem maior de nós para encontrar uma solução completa.

6.2.3 Conjunto Inicial de Arestas

Os testes $A1.1$ e $A2.1$, correspondem aos testes $A1$ e $A2$, já apresentados na Tabela (8), porém tendo como ponto de partida todas as arestas selecionadas como resultado do algoritmo de Kruskal e todas as arestas que foram cogitadas, mas que por formarem ciclo foram descartadas. Os resultados são apresentados na Tabela (10).

É notório que os algoritmos que utilizam o LEF como função de avaliação ainda são os que possuem melhor desempenho em relação ao custo da função objetivo e também sobre o tempo de execução. Na Tabela (10), o teste $A2.1$ avalia e poda mais nós de forma prematura

que $A2$, em média, 66% e 21% de nós a mais, respectivamente. No entanto, em média não existe uma melhora significativa no valor da função objetivo. Além disso, o tempo de execução é superior, com exceção das instâncias $n = 100$ e $n = 150$ com $k = 10$. Diante disso, não é possível afirmar que $A2.1$ é melhor que $A2$, contudo, certamente é melhor que $A1$ e $A1.1$.

		Valor da Função Objetivo				Número de Nós Avaliados				Número de Nós Prematuros				Tempos de Execução (s)			
n	k	$A1$	$A1.1$	$A2$	$A2.1$	$A1$	$A1.1$	$A2$	$A2.1$	$A1$	$A1.1$	$A2$	$A2.1$	$A1$	$A1.1$	$A2$	A2.1
100	5	6.37	6.37	6.37	6.37	49084	68796	46524	67575	1	6	1	7	2.48	3.56	2.40	3.49
120	5	6.63	6.63	6.63	6.63	109955	114731	104095	112034	4	8	11	14	5.00	5.00	5.00	5.00
150	5	**8.15**	8.21	**8.15**	8.21	131703	147544	126451	141785	4	0	6	5	5.00	5.00	5.00	5.00
200	5	9.00	9.03	**8.99**	9.03	134818	122693	118425	116335	8	0	9	6	5.00	5.00	5.00	5.00
300	5	11.87	**11.82**	11.87	**11.82**	118737	122376	110621	117501	4	2	4	3	5.00	5.00	5.00	5.00
400	5	14.03	**13.96**	14.03	**13.96**	108055	128113	97701	120273	5	0	5	1	5.00	5.00	5.00	5.00
500	5	16.38	**16.35**	16.38	**16.35**	87331	91455	79451	81285	0	0	2	2	5.00	5.00	5.00	5.00
100	10	6.74	6.74	6.74	6.74	473	479	339	343	0	0	0	0	0.02	0.02	0.01	0.01
120	10	7.24	7.24	7.24	7.24	2735	3350	2661	3253	4	2	6	5	0.17	0.20	0.16	0.19
150	10	8.92	8.92	8.92	8.92	63557	31652	60319	30315	0	0	0	0	2.52	1.37	2.42	1.29
200	10	10.13	10.13	10.13	10.13	127295	134662	124927	133461	2	2	10	11	5.00	5.00	5.00	5.00
300	10	13.00	**12.95**	13.00	**12.95**	110485	96179	108705	92967	5	3	5	3	5.00	5.00	5.00	5.00
400	10	**15.59**	15.74	**15.59**	15.74	138221	128187	126345	124221	0	3	4	3	5.00	5.00	5.00	5.00
500	10	18.16	**18.05**	18.16	**18.05**	65331	64526	63023	64851	0	1	6	9	5.00	5.00	5.00	5.00
120	20	8.01	8.01	8.01	8.01	71	81	69	77	0	1	0	2	0.00	0.00	0.00	0.00
150	20	8.89	8.89	8.89	8.89	174	210	158	196	0	0	2	3	0.00	0.00	0.00	0.00
200	20	10.83	10.83	10.83	10.83	25775	31099	23882	28109	3	0	3	2	1.29	1.61	1.22	1.45
300	20	13.57	13.57	13.57	13.57	97753	97250	96531	96462	2	0	4	3	5.00	5.00	5.00	5.00
400	20	16.24	16.24	16.20	16.24	121161	135972	115517	138386	0	2	3	9	5.00	5.00	5.00	5.00
500	20	**18.62**	18.75	**18.62**	18.75	71229	99548	70547	99335	4	0	4	3	5.00	5.00	5.00	5.00
150	40	9.61	9.61	9.61	9.61	45	55	45	55	1	0	1	0	0.00	0.00	0.00	0.00
200	40	11.37	11.37	11.37	11.37	7	107	7	107	1	2	1	2	0.00	0.02	0.00	0.02
300	40	14.34	14.34	14.34	14.34	1622	1916	1580	1901	1	1	4	4	0.16	0.17	0.16	0.17
400	40	16.90	**16.86**	16.90	**16.86**	149513	141089	149751	138501	0	2	3	2	5.00	5.00	5.00	5.00
500	40	19.48	19.48	19.48	19.48	58269	59632	58275	62001	0	0	4	2	5.00	5.00	5.00	5.00

Tabela 10 – Resultados das variações no conjunto inicial de arestas do utilizadas com ponto de partida do algoritmo *Branch-and-Bound*.

6.2.4 Pré-processamento

O valor da função objetivo gerado pelo algoritmo HEF é usado como o primeiro limite superior em cada instância avaliada no teste $A2.2$ na Tabela (11) em que temos como ponto de partida da árvore geradora mínima. O teste $A5$ representa a união dos testes $A2.1$ e $A2.2$, onde temos como limite superior o HEF e como ponto de partida todas as arestas da árvore geradora mínima, além de todas as arestas cogitadas.

De acordo com a Tabela (11), nesse novo grupo de testes $A2.2$ e $A5$ tiveram melhor desempenho que $A2$ e $A2.1$. Avaliando os valores da função objetivo, temos que $A2.2$ obteve o melhor resultado em 4 instâncias e obteve o mesmo custo de função objetivo que os demais testes da Tabela (11) em outras 2 instâncias. Seguindo com essa avaliação, $A5$ alcançou o melhor resultado em 3 instâncias e esteve entre os melhores em outras 4. Com exceção das

colunas de número de nós prematuros, onde $A2.1$ obteve melhor resultado, $A2.2$ teve a melhor desempenho. O teste $A2.2$ encontrou bons valores de função objetivo buscando em menos nós e, como consequência, necessitando de menos tempo para encontrar a melhor solução possível a partir da árvore geradora mínima utilizada, em média, 6% menor que $A5$ e 16% menor em relação a $A2$. Sob esse ponto de vista, é correto afirmar que utilizar a função de avaliação ϕ_3 (Seção 5.3.1) e a floresta gerada pelo HEF como primeira solução viável produz os melhores resultados encontrados até agora.

		Valor da Função Objetivo				Número de Nós Avaliados				Nº de Nós Prematuros				Tempos de Preprocessamento (s)				Tempos de Execução (s)			
n	k	$A2$	$A2.1$	$A2.2$	$A5$	$A2$	$A2.1$	$A2.2$	$A5$	$A2$	$A2.1$	$A2.2$	$A5$	$A2$	$A2.1$	$A2.2$	$A5$	$A2$	$A2.1$	$A2.2$	$A5$
100	5	6.37	6.37	6.37	6.37	46524	68796	39363	47957	1	6	0	1	0.00	0.00	0.00	0.00	2.40	3.56	2.10	2.62
120	5	6.63	6.63	6.63	6.63	104095	114731	103273	110646	11	8	5	6	0.00	0.00	0.00	0.00	5.00	5.00	5.00	5.00
150	5	8.15	8.21	8.09	**8.06**	126451	147544	96903	92937	6	0	0	0	0.00	0.00	0.00	0.00	5.00	5.00	5.00	5.00
200	5	8.99	9.03	**8.91**	8.97	118425	122693	65931	89849	9	0	0	0	0.00	0.00	0.01	0.01	5.00	5.00	5.00	5.00
300	5	11.87	11.82	11.67	**11.58**	110621	122376	54909	50059	4	2	0	0	0.01	0.01	0.03	0.03	5.00	5.00	5.00	5.00
400	5	14.03	13.96	**13.58**	13.62	97701	128113	32107	42801	5	0	0	0	0.02	0.03	0.05	0.06	5.00	5.00	5.00	5.00
500	5	16.38	16.35	**15.90**	15.95	79451	91455	26307	26707	2	0	0	0	0.04	0.05	0.09	0.09	5.00	5.00	5.00	5.00
100	10	6.74	6.74	6.74	6.74	339	479	334	336	0	0	0	0	0.00	0.00	0.00	0.00	0.01	0.02	0.01	0.01
120	10	7.24	7.24	7.24	7.24	2661	3350	2627	2891	6	2	3	2	0.00	0.00	0.00	0.00	0.16	0.20	0.17	0.18
150	10	8.92	8.92	8.92	8.92	60319	31652	60299	26981	0	0	0	0	0.00	0.00	0.00	0.00	2.42	1.37	2.46	1.19
200	10	10.13	10.13	10.13	10.13	124927	134662	124047	131833	10	2	3	3	0.00	0.00	0.01	0.01	5.00	5.00	5.00	5.00
300	10	13.00	**12.95**	13.00	**12.95**	108705	96179	96605	94421	5	3	1	1	0.01	0.01	0.03	0.03	5.00	5.00	5.00	5.00
400	10	15.59	15.74	**15.46**	**15.46**	126345	128187	92801	51003	4	3	0	0	0.02	0.03	0.05	0.06	5.00	5.00	5.00	5.00
500	10	18.16	18.05	18.08	**17.98**	63023	64526	52749	34931	6	1	0	0	0.04	0.05	0.09	0.09	5.00	5.00	5.00	5.00
120	20	8.01	8.01	8.01	8.01	69	81	67	77	0	1	0	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
150	20	8.89	8.89	8.89	8.89	158	210	97	154	2	0	0	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
200	20	10.83	10.83	10.83	10.83	23882	31099	23448	28095	3	0	1	1	0.00	0.00	0.01	0.01	1.22	1.61	1.21	1.46
300	20	13.57	13.57	13.57	13.57	96531	97250	95553	98256	4	0	2	2	0.01	0.01	0.03	0.03	5.00	5.00	5.00	5.00
400	20	**16.20**	16.24	**16.20**	16.24	115517	135972	112856	124580	3	2	1	2	0.02	0.03	0.06	0.06	5.00	5.00	5.00	5.00
500	20	**18.62**	18.75	**18.62**	**18.62**	70547	99548	69409	63683	4	0	2	0	0.04	0.05	0.09	0.09	5.00	5.00	5.00	5.00
150	40	9.61	9.61	9.61	9.61	45	55	43	54	1	0	0	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
200	40	11.37	11.37	11.37	11.37	7	107	7	21	1	2	0	0	0.00	0.00	0.01	0.01	0.00	0.02	0.02	0.01
300	40	14.34	14.34	14.34	14.34	1580	1916	1459	1847	4	1	2	2	0.01	0.01	0.03	0.03	0.16	0.17	0.16	0.17
400	40	16.90	**16.86**	16.90	**16.86**	149751	141089	147621	113276	3	2	1	1	0.02	0.03	0.06	0.06	5.00	5.00	5.00	5.00
500	40	19.48	19.48	19.48	19.48	58275	59632	57441	61935	4	0	1	2	0.04	0.05	0.09	0.09	5.00	5.00	5.00	5.00

Tabela 11 – Resultados das variações no pré-processamento do algoritmo *Branch-and-Bound*.

6.2.5 Heurística de Cobertura Mínima para Árvores k-Capacitadas

Nos testes da Tabela (12), o teste $A2.2$ da Tabela (11) é comparado com a heurística de cobertura mínima para árvores k-capacitadas, representado na coluna $B1$, e, também, com união desses dois algoritmos. É importante lembrar que nesse trabalho já realizamos testes, onde o conjunto de arestas inicial eram todas as arestas da árvore geradora mínima mais todas as arestas cogitadas no algoritmo de Kruskal. No entanto, é possível existir arestas que não estavam naquele conjunto por terem um custo maior que o custo da última aresta a ser adicionada na árvore geradora mínima.

a Tabela (12), o teste $B1 + A2.2$ obteve os melhores valores de função objetivo em 7

das 25 instâncias, no entanto necessitou ser executado por um período maior de tempo que o teste $A2.2$ para conseguir finalizar sua busca, atingindo o limite máximo de tempo de execução em todas as instâncias. Diante dos resultados produzidos, $A2.2$ ainda é o teste que obteve os melhores resultados para o PAkC gerando boas soluções em baixos tempos de execução.

		Valor da Função Objetivo			Tempos de Execução (s)		
n	k	$A2.2$	$B1$	$B1 + A2.2$	$A2.2$	$B1$	$B1 + A2.2$
100	5	**6.37**	6.63	6.52	2.10	0.00	5.00
120	5	**6.63**	7.43	6.66	5.00	0.00	5.00
150	5	**8.09**	9.31	8.13	5.00	0.00	5.00
200	5	**8.91**	9.41	8.93	5.00	0.00	5.00
300	5	11.67	12.15	**11.63**	5.00	0.01	5.00
400	5	13.58	14.58	**13.56**	5.00	0.01	5.00
500	5	15.90	17.74	**15.87**	5.00	0.02	5.00
100	10	**6.74**	7.02	**6.74**	0.01	0.00	5.00
120	10	7.24	7.78	**7.22**	0.17	0.00	5.00
150	10	**8.92**	9.16	9.08	2.46	0.00	5.00
200	10	**10.13**	11.20	10.26	5.00	0.00	5.00
300	10	13.00	14.50	**12.99**	5.00	0.01	5.00
400	10	**15.46**	16.32	**15.45**	5.00	0.01	5.00
500	10	**18.08**	18.98	18.13	5.00	0.03	5.00
120	20	**8.01**	9.28	**8.01**	0.00	0.00	5.00
150	20	**8.89**	9.05	9.00	0.00	0.00	5.00
200	20	**10.83**	11.24	10.85	1.21	0.00	5.00
300	20	**13.57**	14.36	13.63	5.00	0.01	5.00
400	20	**16.20**	17.03	16.27	5.00	0.03	5.00
500	20	**18.62**	20.33	18.63	5.00	0.03	5.00
150	40	**9.61**	10.03	9.67	0.00	0.00	5.00
200	40	**11.37**	12.66	**11.37**	0.02	0.00	5.00
300	40	**14.34**	14.75	14.42	0.16	0.01	5.00
400	40	16.90	17.82	**16.85**	5.00	0.02	5.00
500	40	**19.48**	21.94	19.52	5.00	0.03	5.00

Tabela 12 – B&B x Heurística de Cobertura Mínima para Árvores k-Capacitadas.

7 CONCLUSÃO

Nesse trabalho, implementamos um algoritmo *Branch-and-Bound*, o modelo de programação matemática proposto em (BONATES *et al.*, 2011) e a heurística de Cobertura Mínima para Árvores k-Capacitadas. Melhoramos o algoritmo introduzindo novas condições na função de avaliação dos nós das árvores de *Branch-and-Bound* e utilizando o algoritmo HEF como primeiro limite superior para as soluções parciais. Realizamos, também, experimentos computacionais com a Heurística de Cobertura Mínima para Árvores k-Capacitadas proposta nesse trabalho.

Os experimentos computacionais foram realizados sobre 25 instâncias. De forma geral, os resultados mostram que a versão final do algoritmo *Branch-and-Bound* forneceu resultados próximos ao valor ótimo, em comparação aos valores produzidos pelo modelo de programação matemática ND tendo como ponto de partida a árvore geradora mínima. Concluímos, portanto, que dentre as variações propostas ao algoritmo *Branch-and-Bound* a versão aplicada no teste $A2.2$ na Seção (6.2.4) do capítulo anterior é a melhor variação do algoritmo *Branch-and-Bound*.

Observamos que a heurística de Árvores Disjuntas de Prim encontra solução parcial de boa qualidade, no entanto os vértices que permanecem isolados necessitam de um novo tratamento, além de apenas serem incluídos nas árvores constituídas.

A heurística de Cobertura Mínima para Árvores k-Capacitadas diversifica as arestas utilizadas, fazendo uso não somente das arestas da árvore geradora mínima. Contudo, esta heurística mostrou melhor desempenho quando aliada ao algoritmo *Branch-and-Bound* unindo o conjunto de arestas selecionadas pela heurística com as arestas da árvore geradora mínima.

Concluímos que as heurísticas propostas não geram bons resultados quando comparados aos resultados obtidos pelo algoritmo *Branch-and-Bound*. No entanto, em trabalhos futuros, temos a possibilidade de explorar as duas heurísticas apresentadas buscando meios de reduzir o tempo de execução. Além disso, podemos unir o conjunto de arestas gerado pelas heurísticas sugeridas nesses trabalho com o conjunto de arestas de uma árvore geradora mínima do grafo de entada e, sobre esse novo conjunto de arestas, podemos aplicar diversos algoritmos disponíveis na literatura. Dessa maneira, podemos reunir a variedade do conjunto inicial de arestas produzida pelas novas heurísticas e a objetividade e eficiência dos algoritmos existentes na literatura.

REFERÊNCIAS

BOLLOBÁS, B. **Modern graph theory**. [S.l.]: Springer Science & Business Media, 2013. v. 184.

BONATES, T. O.; SANTIAGO, C. P.; SILVA, J. B. Novas abordagens exatas para o problema de partiçao de um grafo emarvores k-capacitadas. XLIII Simpósio Brasileiro de Pesquisa Operacional, p. 2056–2068, 2011.

BONDY, J. A.; MURTY, U. S. R. *et al.* **Graph theory with applications**. [S.l.]: Citeseer, 1976. v. 290.

COUËTOUX, B. A $\frac{3}{2}$ approximation for a constrained forest problem. In: SPRINGER. **European Symposium on Algorithms**. [S.l.], 2011. p. 652–663.

DOMINGO-FERRER, J.; MATEO-SANZ, J. M. Practical data-oriented microaggregation for statistical disclosure control. **IEEE Transactions on Knowledge and data Engineering**, IEEE, v. 14, n. 1, p. 189–201, 2002.

GOEMANS, M. X.; WILLIAMSON, D. P. A general approximation technique for constrained forest problems. **SIAM Journal on Computing**, SIAM, v. 24, n. 2, p. 296–317, 1995.

IMIELIŃSKA, C.; KALANTARI, B.; KHACHIYAN, L. A greedy heuristic for a minimum-weight forest problem. **Operations Research Letters**, Elsevier, v. 14, n. 2, p. 65–71, 1993.

JI, X. Graph partition problems with minimum size constraints. **Rensselaer Polytechnic Institute, Troy, New York**, 2004.

JUNGNICKEL, D. **Graphs, networks and algorithms**. [S.l.]: Springer Science & Business Media, 2007. v. 5.

LASZLO, M.; MUKHERJEE, S. Another greedy heuristic for the constrained forest problem. **Operations Research Letters**, Elsevier, v. 33, n. 6, p. 629–633, 2005.

LASZLO, M.; MUKHERJEE, S. Minimum spanning tree partitioning algorithm for microaggregation. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 17, n. 7, p. 902–911, 2005.