ÁRVORES CAPACITADAS

Hugo de Oliveira Barbalho

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Abilio Pereira de Lucena Filho
Luidi Gelabert Simonetti

Rio de Janeiro
Dezembro de 2018

ÁRVORES CAPACITADAS

Hugo de Oliveira Barbalho

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Abilio Pereira de Lucena Filho, Ph.D.

Prof. Luidi Gelabert Simonetti, D.Sc.

Prof. Alexandre Salles da Cunha, D.Sc.

Prof. Felipe Maia Galvão França, Ph.D.

Prof. Yuri Abitbol De Menezes Frota, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2018

Barbalho, Hugo de Oliveira

Árvores Capacitadas/Hugo de Oliveira Barbalho. – Rio de Janeiro: UFRJ/COPPE, 2018.

IX, 64 p.: il.; 29, 7cm.

Orientadores: Abilio Pereira de Lucena Filho
Luidi Gelabert Simonetti

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2018.

Referências Bibliográficas: p. 61 – 64.

1. Árvore Geradora Mínima. 2. Árvore Geradora Mínima Capacitada. 3. Geração de Colunas. 4. Programação Inteira. I. Lucena Filho, Abilio Pereira de *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

ÁRVORES CAPACITADAS

Hugo de Oliveira Barbalho

Dezembro/2018

Orientadores: Abilio Pereira de Lucena Filho
Luidi Gelabert Simonetti

Programa: Engenharia de Sistemas e Computação

Nesta tese de doutorado investigamos diversos problemas relacionados a reformulações Dantzig-Wolfe sugeridas para o Problema da Árvore Geradora Mínima Capacitada (CMSTP). Para esses problemas, propomos formulações e algoritmos de soluções exatas e heurísticas para os problemas investigados.

Inicialmente, nós estudamos o problema da Arborescência Mínima (MAP). Esse problema é uma relaxação do problema de *pricing* associado as reformulações Dantzig-Wolfe para o CMSTP. Como primeira contribuição, é apresentada uma formulação matemática e um algoritmo *Branch-and-cut* para resolver o MAP. Em seguida, é proposta uma formulação para a versão do problema que é restrita a grafos direcionados acíclicos (DAGs - Directed Acyclic Graphs). Os limites encontrados pela relaxação linear dessa formulação são, em média, 77% melhores. Ainda, apresentamos um novo conjunto de instâncias DAGs, sendo essas mais dificeis que as já existentes para o problema.

O segundo problema relacionado às reformulações Dantzig-Wolfe sugeridas para o CMSTP é denominado problema da Arborescência Mínima Capacitada (CMAP). Neste trabalho, definimos formalmente esse problema e apresentamos uma formulação matemática. Adicionalmente, generalizamos as desigualdades *multistar*, originalmente propostas para o CMSTP, e elaboramos um algoritmo de separação para essas desigualdades. Os resultados computacionais mostram que o algoritmo de plano de cortes sob essa formulação, quando fortalecida pelas *multistars* generalizadas, levam à limites inferiores 50% melhores, em média. Por fim, nós apresentamos uma heurística baseada em programação dinâmica e um conjunto de instâncias para o CMAP. Essas são criadas a partir de um algoritmo de geração de colunas para o CMSTP.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

CAPACITATED TREES

Hugo de Oliveira Barbalho

December/2018

Advisors: Abilio Pereira de Lucena Filho
Luidi Gelabert Simonetti

Department: Systems Engineering and Computer Science

In this thesis we investigate some problems related to Dantzig-Wolfe reformulation suggested to the CMSTP. Indeed, we study same variants of the column generation problems relative to such reformulations. In this process, we propose formulations and algorithms for exact and heuristic solutions to the problems.

We start by studing the Minimum Arborescence Problem (MAP), which is a relaxation of the pricing problem associated to the CMSTP Dantzig-Wolfe reformulations. We propose a better formulation for the problem and, for the version restricted to Directed Acyclic Graphs (DAGs), we prove that this formulation leads to a better representation of the cover hull. Additionally, we present a set of new DAGs instances, which turns to be more difficult to solve than the existing ones.

Later, we formally define the pricing problem related to Dantzig-Wolfe reformulation suggested to the CMSTP, named Capacitated Minimum Arborescence Problem (CMAP). Except for the capacity constraint, the formulation we propose for this problem is equal to the one proposed for the MAP. Moreover, we generalize the multistar inequalities, originaly proposed for the CMSTP, and present a separation algorithm. Computational results show that the cutting plane algorithm under the formulation enforced by the generalized multistars leads to dual bounds 50% tighter, on average. Finally, we propose a dynamic programming based heuristic and benchmark instances originated from a column generation algorithm.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Problemas definidos sob estruturas de arborescências ou árvores são frequentemente encontrados em diferentes projetos de redes de telecomunicação. Dependendo da natureza do problema, exigências adicionais podem deixá-lo ainda mais complexo e interessante do ponto de vista econômico. Nesta tese, estudaremos problemas definidos sob arborescências e árvores que se diferenciam por algumas exigências particulares. São eles: o problema da Arborescência de Custo Mínimo; o problema da Arborescência Capacitada de Custo Mínimo; e o problema da Árvore Geradora Capacitada de Custo Mínimo.

No problema da Arborescência de Custo Mínimo (MAP - *Minimum Arborescence Problem*), procura-se uma arborescência, não necessariamente geradora, em que a soma dos custos dos arcos da arborescência seja a menor possível. A raiz da arborescência pode ser definida de duas maneiras: (i) um vértice é escolhido como raiz a priori, ou (ii) qualquer vértice é uma raiz em potencial. Além disso, o custo associado aos arcos do grafo pode ser positivo ou negativo. Note que quando todos os custos são positivos, as soluções contendo exatamente um vértice (necessariamente a raiz no caso (i)) representam uma solução ótima. Analogamente, quando todos os custos são negativos, uma arborescência geradora de custo mínimo corresponde à solução ótima do problema. O algoritmo [3] resolve o problema da arborescência geradora em tempo polinomial tanto no caso (i) quanto no caso (ii).

O problema da Arborescência Capacitada de Custo Mínimo (CMAP - *Capacitated Minimum Arborescence Problem*) também tem como objetivo encontrar uma arborescência, não necessariamente geradora, em que a soma dos custos dos arcos da arborescência seja a menor possível. No CMAP, porém, exige-se ainda que a soma das demandas dos vértices não exceda uma capacidade predefinida. O CMAP pode ser definido em duas versões dependendo da forma como o vértice raiz é escolhido. Assim como no MAP, teremos uma versão associada ao caso (i) e uma versão associada ao caso (ii). Além de ser um problema intrinsecamente interessante, o mesmo surge de uma decomposição *Dantzig-Wolfe* para o problema

da Árvore Geradora Capacitada de Custo Mínimo, proposta por Uchoa et al. [4].

No problema da Árvore Geradora Capacitada de Custo Mínimo (CMSTP - *Capacitated Minimum Spanning Tree Problem*), o objetivo é encontrar uma árvore geradora de custo mínimo enraizada em um vértice r definido a priori, onde para cada subárvore ligada à raiz, a soma da demanda dos seus vértices não excede uma capacidade máxima predefinida. Na literatura desse problema existem trabalhos que se limitam a resolver apenas o caso unitário, onde todos os vértices tem demandas de peso 1, e outros que consideram o caso geral.

1.1 Contribuições e Organização

Este trabalho tem como objetivo o estudo dos três problemas apresentados anteriormente, bem como o desenvolvimento de algoritmos heurísticos e exatos.

Os primeiros trabalhos da literatura do MAP estudam sua versão restrita a grafos direcionados acíclicos (DAGs - Directed Acyclic Graphs) [2, 5–7]. A versão generalizada desse problema, definida sob grafos direcionados (DGs - directed graphs), começou a ser explorada em 2006 [1, 2]. No Capítulo 2, estudaremos as duas versões do MAP. Para a versão generalizada (DGs), iremos apresentar uma formulação matemática alternativa à apresentada em Duhamel et al [1]. Nessa formulação, a conectividade é garantida através de *cutsets* generalizadas, enquanto que Duhamel et al [1] propõe uma formulação com variáveis de multifluxo. Quando a dimensão do grafo cresce, o número de variáveis envolvidas nessa segunda formulação torna o problema intratável do ponto de vista computacional. Em contraste a esse cenário, a formulação com *cutsets* generalizadas utiliza menos variáveis e através de um algoritmo de plano cortes polinomial é possível trabalhar com um número exponencial de desigualdades *cutsets* de forma eficiente. Essa formulação foi sugerida por Ljubic et al [8] como uma reformulação bidirecionada para o problema da árvore de Steiner com coleta de prêmios (PCSTP - *Prize-Collecting Steiner Tree Problem*) [9]. Para a versão restrita a DAGs, Rao et al [5, 6] propõe uma formulação onde o número de restrições e variáveis é da ordem da dimensão do grafo. Para essa versão, apresentamos uma formulação em que a árvore de enumeração do *Branch-and-bound* tende a ser menor e as *cutsets* generalizadas funcionam como cortes que fortalecem o limite da relaxação linear.

No Capítulo 3, apresentamos as principais contribuições, limitadas a métodos exatos, encontradas na literatura do CMSTP. Começamos esse capítulo definindo formalmente o problema e fazendo uma breve revisão bibliográfica. Em seguida, estudamos duas formulações e desigualdades válidas existentes para o CMSTP. A primeira formulação é definida sob variáveis de arcos e a segunda é definida sob variáveis de arcos com demandas. Uma reformulação *Dantzig-Wolfe* para essas

formulações é então descrita, seguida de uma classe de desigualdades denominada Corte de Capacidade Extendido [4], que pode ser utilizada para fazer *lifting* das primeiras desigualdades.

Para o CMAP, apresentamos uma formulação matemática que adiciona à formulação do MAP uma nova restrição referente a capacidade máxima. Em seguida, apresentamos uma nova família de desigualdades para o problema, que é baseada nas desigualdades *multistar* [10] CMSTP, e um algoritmo exato de separação. Sob essa formulação e essa nova desigualdade, um algoritmo *Branch-and-Cut* é implementado para o CMAP. Nos resultados computacionais, veremos que esses cortes melhoram significativamente o limite da relaxação linear. Como heurística, apresentamos uma programação dinâmica para encontrar soluções viáveis de boa qualidade. Por último, apresentamos um conjunto de instâncias para o CMAP criado a partir das iterações de um algoritmo de geração de colunas para o CMSTP.

Com o algoritmo apresentado nesta tese para o CMAP é possível resolver o problema de *pricing* da geração de colunas do CMSTP. Em Uchoa et al [4], o problema de *pricing* é resolvido por uma relaxação do CMAP, que é uma estrutura denominada *q-arbs*. Sendo assim, teoricamente, o limite encontrado por um algoritmo de geração de colunas que utiliza o CMAP para modelar o problema de *pricing* tem um limite dual melhor ou igual ao do algoritmo que utiliza *q-arbs* para resolver o problema de *pricing* associado ao problema mestre. Por outro lado, como veremos no Capítulo 4, o problema CMAP é NP-difícil e, na prática, resolver um algoritmo de *Branch-and-Cut* toda iteração da geração de colunas pode ser computacionalmente custoso. Por esse motivo, usamos uma heurística para tentar encontrar colunas com custo reduzido negativo e assim diminuir a quantidade de vezes que resolvemos o CMAP com o algoritmo de *Branch-and-Cut*.

A organização desta tese é feita da seguinte maneira: No Capítulo 2, apresentamos uma revisão bibliográfica do problema da Arborescência de Custo Mínimo (MAP). Então, formulações matemáticas são descritas tanto para o MAP quanto para o MAP restrito a DAGs. Para uma dessas formulações é proposto um algoritmo exato do tipo *Branch-and-cut*. Para as análises dos resultados computacionais, criamos ainda um novo conjunto de instâncias DAGs. O Capítulo 3 inicia com uma revisão bibliográfica do problema da Árvore Geradora Capacitada de Custo Mínimo (CMSTP). Ainda nesse capítulo, estudamos formulações e desigualdades existentes na literatura e apresentamos a decomposição *Dantzig-Wolfe* proposta em [4]. No Capítulo 4, uma formulação para o problema da Arborescência Capacitada de custo mínimo (CMAP) e uma generalização das desigualdades *multistar* são apresentadas. Em seguida, são propostos um procedimento de separação e uma heurística primal baseada em programação dinâmica. Ao final desse capítulo, conduzimos um experimento computacional para avaliar a performance

do algoritmo proposto e para analisar a dificuldade das instâncias provenientes do algoritmo de geração de colunas para o CMSTP. Por fim, no Capítulo 5, revisamos os principais resultados obtidos e discutimos trabalhos futuros.

Capítulo 2

Algoritmo para o Problema da Arborescência Mínima

Seja $G = (V, A)$ um grafo direcionado com um conjunto de vértices V e um conjunto de arcos A e assuma que custos $\{c_a \in \mathbb{R} : a \in A\}$ são associados aos arcos de G. Neste capítulo, vamos investigar *arborescências* de G. Uma *arborescência* de G é um subgrafo $T = (V_T, A_T)$ com $V_T \subseteq V$ e $A_T \subseteq A$ que satisfaça as seguintes condições: (i) $|A_T| = |V_T| - 1$ e (ii) existe exatamente um único vértice raiz $r \in V_T$ de onde partem caminhos direcionados em T para cada vértice de V_T. Subgrafos T são então denominados *r-arborescências* de G, onde r é o vértice raiz da arborescência. Nosso objetivo é identificar *arborescências* de G onde a soma dos custos dos arcos seja a menor possível.

Distinguimos ainda os casos onde o vértice $r \in V$ da arborescência é predefinido ou não. Se a raiz é predefinida, encontrar a *r-arborescência* de menor custo de G é denominado como o problema da Arborescência Enraizada de Custo Mínimo (Minimum Rooted Arborescence Problem - MRAP). Por outro lado, se nenhuma raiz for imposta explicitamente, o problema é denominado como o problema da Arborescência de custo mínimo (Minimum Arborescence Problem - MAP).

2.1 Definição e Revisão Bibliográfica

MRAP foi apresentado por Rao e Sridharan [6, 11] para grafos direcionados acíclicos (DAGs - Directed Acyclic Graphs). Nesses trabalhos, DAGs são conexos e seus arcos (i, j) tem uma ordenação topológica tal que $i < j$. Além disso, apenas arcos (r, i) estão associados ao vértice raiz r de G. Através de uma redução feita a partir do problema de Localização Não Capacitada (Uncapacitated Plant Location Problem) [12], foi conduzida a prova de que MRAP restrito a DAGs é NP-Difícil em [6, 11]. MAP, por sua vez, foi proposto em termos de grafos direcionados (Directed Graphs

- DGs) gerais por Duhamel et al. [1]. Nesse mesmo trabalho, os autores provam que o MAP é NP-Difícil a partir de uma redução do problema da Árvore de Steiner com Peso nos Vértices (Node Weighted Steiner Tree Problem - NWSTP) [13]. Por último, para o MRAP restrito a DAGs, [6, 11] também apresentam algumas considerações sobre o trabalho de Rao and McGinnis [5] em que os custos são atribuídos apenas aos vértices de G e não aos arcos, como em [6, 11].

Um modelo de Programação Linear Inteira para o MRAP restrito a DAGs e um framework Lagrangeano para encontrar limites primais e duais para o problema são propostos em [6, 11]. Instâncias com até 301 vértices e 327 arcos são resolvidas até a otimalidade em [6]. Em seguida, em outra contribuição para MRAP restrito a DAGs, Meteo et al. [7] investiga uma heurística e um algoritmo colônia de formigas. O algoritmo foi testado em um conjunto de instâncias aleatórias DAGs com até 5000 vértices e para essas instâncias os resultados foram melhores que os apresentados em Rao et al. [6].

Quando se pensa em modelos, deve-se notar que MRAP não é tão restritivo quanto pode parecer. Até o MAP pode ser entendido como um caso restrito do MRAP, como indica [1]. Mais especificamente, MAP é um MRAP definido sob um grafo aumentado $G = (V, A)$, obtido a partir da adição de um *vértice raiz*, r, artificial ao grafo original. Nesse caso, os únicos arcos de G saindo de r são aqueles em $\{(r, j) : j \in V \setminus \{r\}\}$, todos eles com custo zero. Além disso, soluções viáveis seriam restritas à *r-arborescências* com um único arco saindo de r. Note que, nesse caso, r estaria sendo um apontador para a *verdadeira* raiz da arborescência. Por último, devemos observar que possíveis custos nos vértices, como nos DAGs em [5], podem ser facilmente adaptados. Para isso, basta redefinir os custos dos arcos como $\{c_{ij} := c_{ij} + d_j : (i, j) \in A\}$.

Seguindo os pontos anteriores, Duhamel et al [1] propõe uma formulação com variáveis de multifluxo (Multi Commodity Flow Formulation - MCFF). Ainda, uma heurística primal e dois frameworks Lagrangeanos baseados na formulação MCFF são estudados. Para cada um deles, são apresentados limites inferiores e limites superiores heurísticos. Resultados computacionais são apresentados para instâncias de até 400 vértices, mas certificados de otimalidade são apresentados de forma consistente apenas para instâncias com até 100 vértices. Computar o limite da relaxação linear da formulação MCFF consome um tempo computacional excessivo e por esse motivo, apenas instâncias com até 100 vértices tem esse limite calculado.

Mais recentemente, Blum e Calvo [2] investigaram duas metaheurísticas. Uma delas foi projetada para DAGs e é baseada em uma formulação para o MRAP restrito a DAGs proposta em [6, 11]. A outra se aplica a grafos gerais e é baseada na formulação MCFF [1]. Dado um grafo $G = (V, A)$, a ideia principal por trás dos dois algoritmos é a mesma. Isto é, gerar um subgrafo de G com um

número reduzido de arcos com a expectativa de conseguir soluções de boa qualidade para o MRAP. Os subgrafos utilizados em [2] correspondem a um conjunto de *r-arborescências* de G geradas heuristicamente. Dessa forma, as *r-arborescências* para esses subgrafos, obtidas através do uso de um *solver* de Programação Inteira Mista (Mixed Integer Programming - MIP), são portanto viáveis para o MRAP. Os resultados computacionais para os DAGs propostos por Mateo et al [7] são apresentados em [2], assim como os resultados para novos DAGs e grafos gerais com mais de 3000 vértices.

Finalmente, concluindo essa breve visão geral da literatura do MAP, podemos ressaltar que ele está intimamente relacionado a alguns problemas de Otimização Computacional bem estudados. Dentre eles, o problema da Arborescência Geradora Mínima (Minimum Spanning Arborescence Problem) [14], o Simple Plant Location Problem [12] e o problema de Steiner para Grafos Direcionados (Steiner Problem in Directed Graphs - SPDG) [15]. Além disso, o MAP também possui aplicações interessantes no planejamento de produção multiestágio (multistage production planning) [11] e na reconstrução automatizada de estruturas de árvores consistentes a partir de imagens com ruído [16].

A primeira contribuição deste capítulo é um algoritmo *Branch-and-Cut* (BC) para o MAP. Ele é baseado em uma formulação que usa desigualdades *cutsets* generalizadas para garantir a conectividade da solução. Essa formulação está relacionada à reformulação bi-direcionada para o problema de Steiner em Grafos proposta em [17, 18] e à formulação para o problema da Árvore de Steiner com Coleta de Prêmios (Prize Collecting Steiner Tree Problem - PCSTP) sugerida em [8].

PCSTP é definido sob um grafo não direcionado e árvores são soluções viáveis para o problema. Para qualquer árvore, um prêmio é coletado por cada vértice e um custo é pago para cada aresta. Uma árvore ótima maximiza a soma dos prêmios coletados menos a soma dos custos das arestas selecionadas. Para a versão direcionada do PCSTP, o problema da Árvore de Steiner com Coleta de Prêmio Assimétrico (Asymmetric Prize Collecting Steiner Tree Problem - APCSTP), Leitner et al [19] apresenta um algoritmo Branch-and-Bound baseado em um *dual ascent*. Note que através de uma simples transferência dos prêmios dos vértices para os custos dos arcos, podemos reduzir o APCSTP ao MAP.

Ainda neste capítulo, mostramos que essa formulação define uma melhor representação da envoltória convexa do MAP restrito a DAGs. Por último, para os resultados computacionais dessa versão do MAP, apresentamos um novo conjunto de instâncias.

2.2 Formulação para o MAP

Nesta seção apresentamos duas formulações para o MAP. Ambas são definidas sob um grafo $G = (V, A)$ que aumenta o grafo de entrada $G_0 = (V_0, A_0)$. Esse grafo aumentado G é criado com a adição de um vértice raiz artificial r e arcos saindo de r. Isto é, arcos $\{(r, i) : i \in V_0\}$, todos com custo zero. A primeira formulação segue diretamente das reformulações sob grafos direcionados propostos em [17, 18] e [8], respectivamente para os problemas SPG e PCSTP. Essa formulação garante a conectividade através de desigualdades *cutset* direcionadas (Directed cutset - DCS). A segunda, denominada MCFF, é proposta por Duhamel et al. [1] e é baseada no problema de fluxo em redes.

Ao longo deste trabalho, dado um grafo $G = (V, A)$, vamos usar as seguintes notações. Para cada conjunto $S \subseteq V$ temos $A(S) \subseteq A$ o conjunto de arcos com ambos os vértices em S. Ainda, seja $\delta^-(S) = \{(i, j) \in A : i \notin S, j \in S\}$ e $\delta^+(S) = \{(i, j) \in A : i \in S, j \notin S\}$ os conjuntos de arcos apontando, respectivamente, para dentro de S e para fora de S. Se S tem apenas um vértice, digamos $S = \{i\}$, iremos usar $\delta^-(i)$ (resp. $\delta^+(i)$) ao invés de $\delta^-(\{i\})$ (resp. $\delta^+(\{i\})$) para simplificar a notação. Finalmente, a notação compacta $X(A') = \sum_{a \in A'} x_a$ será usada para representar o somatório das variáveis x para um subconjunto $A' \subseteq A$.

2.2.1 Formulação *cutset* para o MAP

Essa formulação é definida sob as variáveis $\{x_{ij} \geq 0 : (i, j) \in A\}$ e $\{y_i \geq 0 : i \in V_0\}$. Elas são responsáveis por selecionar respectivamente os arcos e os vértices da arborescência e assim a formulação é dada por

$$\min \left\{ \sum_{(i,j) \in A} c_{ij} x_{ij} : (x, y) \in \mathcal{R}_0 \cap (\mathbb{R}^{|A|}, \mathbb{Z}^{|V_0|}) \right\}, \tag{2.1}$$

onde $\mathcal{R}_0$ é a região poliedral definida pela interseção das restrições

$$\sum_{(r,j) \in A} x_{rj} = 1 \tag{2.2}$$

$$\sum_{(j,i) \in A} x_{ji} = y_i, \ \forall i \in V_0 \tag{2.3}$$

$$\sum_{(j,i) \in \delta^-(S)} x_{ji} \geq y_i, \ \forall S \subseteq V_0, \forall i \in S \tag{2.4}$$

$$X(A(S)) \leq y_i, \ \forall (i, j) \in A_0, S = \{i, j\} \tag{2.5}$$

$$0 \leq x_{ij} \leq 1, \ \forall (i, j) \in A \tag{2.6}$$

$$0 \leq y_i \leq 1,\ \forall i \in V_0. \tag{2.7}$$

Para provar a validade da formulação (2.1), considere uma solução viável $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ para a mesma. Ainda, seja $\overline{G} = (\overline{V}, \overline{A})$ um grafo suporte correspondente à $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$. Ou seja, $\overline{G}$ é um subgrafo de G definido pelos vértices $\{i \in V_0 : \overline{y}_i = 1\} \cup \{r\}$ e pelos arcos $\{a \in A : \overline{x}_a = 1\}$.

A restrição (2.2) impõe que apenas um arco esteja saindo de r. Além disso, note que, para cada vértice $i \in V_0$ na solução inteira ($y_i = 1$), existe exatamente um arco $(j, i) \in A$ associado a ele, como exigido por (2.3). As restrições (2.2), (2.3) e (2.5), juntas, garantem que $\overline{G}$ tenha exatamente $(\sum_{i \in V_0} \overline{y}_i + 1)$ vértices, sendo r um deles, e $(\sum_{i \in V_0} \overline{y}_i)$ arcos, como é de se esperar de uma r-arborescência de G. Note que as desigualdades (2.5) correspondentes ao vértice j não são necessárias, uma vez que as restrições (2.3) já pertencem ao poliedro. As desigualdades *cutset* direcionadas (Directed cutset - DCS) (2.4), por sua vez, garantem a conectividade de $\overline{G}$, fazendo com que exista um caminho direcionado partindo de r para cada um dos vértices de $\overline{G}$. Para mostrar que isto de fato acontece, assuma inicialmente que não existe um caminho direcionado em $\overline{G}$ partindo de r para qualquer um dos vértices em $S \subseteq \overline{V}$. Se $S = \emptyset$, o resultado é obviamente válido. Caso contrário, a desigualdade (2.4) está violada por S, o que contradiz nossa premissa de que $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ é viável. Portanto, deve existir um caminho orientado em $\overline{G}$ partindo de r para todo vértice de $\overline{V}$. Consequentemente, podemos concluir que (a) $|\overline{A}| = |\overline{V}| - 1$, (b) $\overline{G}$ é conexo, e (c) $\overline{G}$ tem caminhos direcionados partindo de r para todos os outros vértices de $\overline{V}$. Em suma, $\overline{G}$ define uma r-arborescência de G.

A relaxação linear correspondente à formulação (2.1) é

$$\min \{ \sum_{(i,j) \in A} c_{ij} x_{ij} : (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_0 \}. \tag{2.8}$$

2.2.2 Formulação MCFF para o MAP

Duhamel et al. [1] propõe uma formulação para o MAP baseada no problema de fluxo em redes. Nessa formulação, além das variáveis x e y descritas no modelo anterior, é utilizado um conjunto adicional de variáveis $\{z_a^k : a \in A, k \in V_0\}$, denominadas variáveis de fluxo. As variáveis z_a^k especificam se o caminho de r até k utiliza o arco a. Assim, a formulação MCFF em [1] para o MAP é dada por:

$$\min \left\{ \sum_{(i,j) \in A} c_{ij} x_{ij} : (x, y, z) \in \mathcal{R}_1 \cap (\mathbb{R}^{|A|}, \mathbb{Z}^{|V_0|}, \mathbb{R}^{|A||V_0|}) \right\}, \tag{2.9}$$

onde $\mathcal{R}_1$ é a região poliedral definida pela interseção das restrições

$$\sum_{(r,j)\in A} x_{rj} = 1 \tag{2.10}$$

$$\sum_{(j,i)\in A} x_{ji} = y_i,\ \forall i \in V_0 \tag{2.11}$$

$$\sum_{j\in V} z^k_{rj} - \sum_{j\in V} z^k_{jr} = y_k,\ \forall k \in V_0 \tag{2.12}$$

$$\sum_{j\in V} z^k_{ij} - \sum_{j\in V} z^k_{ji} = 0,\ \forall k, i \in V_0, i \neq k \tag{2.13}$$

$$\sum_{j\in V} z^k_{kj} - \sum_{j\in V} z^k_{jk} = -y_k,\ \forall k \in V_0 \tag{2.14}$$

$$z^k_{ij} \leq x_{ij},\ \forall (i,j) \in A, k \in V_0 \tag{2.15}$$

$$0 \leq x_{ij} \leq 1,\ \forall (i,j) \in A \tag{2.16}$$

$$0 \leq y_i \leq 1,\ \forall i \in V_0, \tag{2.17}$$

$$0 \leq z^k_{ij} \leq 1,\ \forall (i,j) \in A, \forall k \in V_0. \tag{2.18}$$

Nessa formulação, as duas primeiras restrições, (2.10) e (2.11), são iguais as restrições (2.3) e (2.3) respectivamente. As restrições (2.15) garantem que um caminho de r para um vértice $k \in V_0$ só passe por um arco $(i,j) \in A$ quando esse arco estiver na solução ($x_{ij} = 1$). As restrições (2.12)-(2.14) são restrições de conservação de fluxo, que garantem a conectividade da solução. Essas restrições, juntamente com as restrições (2.11) e (2.15), fazem com que exista um caminho direcionado de r para cada um dos vértices da solução. Após colocar todas essas informações juntas, temos que as soluções da formulação são r-arborescências de G.

A relaxação linear para a formulação (2.9) é dada por

$$\min \left\{ \sum_{(i,j)\in A} c_{ij} x_{ij} : (x,y,z) \in \mathcal{R}_1 \right\}. \tag{2.19}$$

2.3 Formulações para o MAP restrito a DAGs

Nesta seção, vamos apresentar duas formulações para o MAP restrito a DAGs. Essas formulações são aplicadas a um grafo direcionado acíclio (DAG) $G = (V, A)$ que aumenta o DAG de entrada $G_0 = (V_0, A_0)$, assim como os grafos de entrada das formulações na Seção 2.2. O vértice artificial $r \in V$ irá determinar a verdadeira raiz de G_0 na solução.

A primeira formulação desta seção é proposta por Rao e Sridharan [6]. A segunda, denominada *GSEC-2*, é uma relaxação da formulação (2.1) apresentada na Seção 2.2.1 para DGs.

2.3.1 Formulação Rao e Sridharan

A formulação de Rao e Sridharan [6], para MAP restrito a DAGs, envolve apenas variáveis $\{x_a \geq 0 : a \in A\}$ associadas aos arcos de G. Sabendo que estas variáveis selecionam os arcos da solução, a formulação pode ser descrita como

$$\min \left\{ \sum_{(i,j)\in A} c_{ij} x_{ij} : x \in \mathcal{R}_2 \cap \mathbb{Z}^{|A|} \right\}, \tag{2.20}$$

onde $\mathcal{R}_2$ é uma região poliedral definida pela interseção das restrições

$$\sum_{(r,j)\in A} x_{rj} = 1 \tag{2.21}$$

$$\sum_{(j,i)\in\delta^-(i)} x_{ji} \leq 1,\ \forall i \in V_0 \tag{2.22}$$

$$\sum_{(k,i)\in\delta^-(i)} x_{ki} \geq x_{ij},\ \forall (i,j) \in A_0 \tag{2.23}$$

$$0 \leq x_{ij} \leq 1,\ \forall (i,j) \in A. \tag{2.24}$$

A restrição (2.21) exige que exatamente um dos arcos saindo de r esteja presente na solução. Desigualdades (2.22) garantem que no máximo um arco irá chegar em cada vértice. As desigualdades (2.23), por sua vez, garantem a conectividade da solução, dado que o grafo de entrada não contém ciclos direcionados. Combinadas à (2.21), elas garantem que existe exatamente um caminho de r para cada vértice da solução. Além disso, como o grafo de entrada é acíclico, a solução também é. Finalmente, combinando essas informações, temos que as soluções da formulação definem r-arborescências de G.

A relaxação linear para a formulação (2.20) é dada por

$$\min \left\{ \sum_{(i,j)\in A} c_{ij} x_{ij} : x \in \mathcal{R}_2 \right\}. \tag{2.25}$$

2.3.2 Formulação *GSEC-2*

A formulação *GSEC-2* envolve as mesmas variáveis $\{x_a \geq 0 : a \in A\}$ e $\{y_i \geq 0 : i \in V_0\}$ da formulação (2.1), as quais estão respectivamente associadas aos arcos

e vértices. Sendo assim, podemos formular o MAP restrito a DAGs da seguinte maneira

$$\min \left\{ \sum_{a \in A} c_a x_a : (x, y) \in \mathcal{R}_3 \cap (\mathbb{R}^{|A|}, \mathbb{Z}^{|V_0|}) \right\}, \tag{2.26}$$

onde $\mathcal{R}_3$ é a região poliedral definida pela interseção das restrições

$$\sum_{(r,j) \in A} x_{rj} = 1 \tag{2.27}$$

$$\sum_{(j,i) \in A} x_{ji} = y_i, \ \forall i \in V_0 \tag{2.28}$$

$$x_{ij} \leq y_i, \ \forall (i,j) \in A_0, \tag{2.29}$$

$$0 \leq x_{ij} \leq 1, \ \forall (i,j) \in A \tag{2.30}$$

$$0 \leq y_i \leq 1, \ \forall i \in V_0, \tag{2.31}$$

Como o grafo G é acíclico e, a exceção das *cutset* direcionadas (2.4), $\mathcal{R}_3$ tem as mesmas restrições de $\mathcal{R}_0$, temos que $\mathcal{R}_3$ define uma r-arborescência para o DAG G.

A relaxação linear para a formulação (2.26) é dada por

$$\min \left\{ \sum_{a \in A} c_a x_a : (x, y) \in \mathcal{R}_3 \right\}. \tag{2.32}$$

Apesar de não serem necessárias para formular o MAP restrito a DAGs, as *cutset* direcionadas (2.4) definem cortes válidos para envoltória convexa de (2.26). Além disto, temos que o poliedro $\mathcal{R}_3$ fortalecido com as desigualdades de *cutset* direcionadas (2.4), ou seja o $\mathcal{R}_0$, é mais forte que o poliedro $\mathcal{R}_2$ apresentado por Rao e Sridharan [6] para pelo menos uma classe de grafos. Agora, vamos comparar esses dois poliedros e mostrar que para uma classe específica de grafos $\mathcal{R}_0 \subset \mathcal{R}_2$.

Comparando poliedros $\mathcal{R}_0$ e $\mathcal{R}_2$

Inicialmente, vamos definir uma classe particular de grafos denominada α-DAGs. Então, para essa classe de grafos, vamos mostrar que a formulação com *cutsets* direcionadas é mais forte que a formulação apresentada por Rao e Sridharan [6]. A seguir, apresentamos a definição dos α-DAG e na Figura 2.1 ilustramos um exemplo dessa classe de grafos.

Definição 1. *Um grafo direcionado acíclico $G = (V, A)$ é um α-DAG se e somente se para ao menos um par de vértices $i \in V$ e $f \in V$:*

(i) existem dois caminhos distintos direcionados de i para f e

(ii) existe ao menos um caminho direcionado de r para i.

Figura 2.1: Um grafo α-DAG à esquerda e uma solução viável para $\mathcal{R}_2$ à direita.

Seguindo [20], seja $\mathcal{R}^0_{proj} = \{\mathbf{x} \in \mathbb{R}^{|A|} : (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_0, \text{ para algum } \mathbf{y} \in \mathbb{R}^{|V_0|}\}$ a projeção de $\mathcal{R}_0$ no subespaço $\mathbb{R}^{|A|}$. Inicialmente vamos mostrar que $\mathcal{R}^0_{proj} \subseteq \mathcal{R}_2$. Com esse resultado, poderemos concluir que (2.25) não irá apresentar um limite inferior melhor que (2.8). Em seguida, vamos focar nos grafos α-DAGs. Para os α-DAGs, vamos então escolher uma solução $\overline{\mathbf{x}} \in \mathcal{R}_2$ e mostrar que não existe um $\overline{\mathbf{y}} \in \mathbb{R}^{|V_0|}$ onde $(\overline{\mathbf{x}}, \overline{\mathbf{y}}) \in \mathcal{R}_0$. Combinando os dois resultados, concluiremos que $\mathcal{R}^0_{proj} \subset \mathcal{R}_2$ para os α-DAGs. Ou seja, (2.1) é uma melhor formulação que (2.25) para esta classe de grafos.

Seja $(\overline{\mathbf{x}}, \overline{\mathbf{y}}) \in \mathcal{R}_0$ e assuma que $\overline{G} = (\overline{V}, \overline{A})$ seja seu grafo suporte correspondente. Então, seguindo os mesmos argumentos apresentados anteriormente para mostrar a validade de (2.1), será possível concluir que $\overline{G}$ é conexo e contém um ou mais caminhos direcionados de r até cada vértice em $\overline{V}$.

Claramente, DAGs correspondentes à arborescências não satisfazem as características de um α-DAG. Isso, porém, não é uma grande limitação, já que um algoritmo de Programação Dinâmica (PD) poderia resolver o MRAP sob um desses grafos em tempo polinomial (recursões da PD partindo das folhas de G e subindo até a raiz r).

Lema 2.3.1. *$\mathcal{R}^0_{proj} \subset \mathcal{R}_2$ para α-DAGs.*

Demonstração. Para mostrar que uma solução $\overline{\mathbf{x}} \in \mathcal{R}^0_{proj}$ de fato pertence à $\mathcal{R}_2$, note que $\overline{\mathbf{x}}$ satisfaz (2.21), já que ela satisfaz (2.2). Ainda, $\overline{\mathbf{x}}$ também satisfaz (2.22) uma vez que, de (2.3) e (2.7), $\sum_{(j,i)\in\delta^-(i)} x_{ji} = \overline{y}_i \leq 1$ é válido para todo $i \in V_0$. Além disto, dado que $\overline{G}$ contém pelo menos um caminho direcionado partindo de r até cada vértice de $\overline{V}$ e (2.3) é satisfeita, (2.23) então também deve ser satisfeita. Sendo assim, temos que $\overline{\mathbf{x}} \in \mathcal{R}_2$ e, consequentemente, $\mathcal{R}^0_{proj} \subseteq \mathcal{R}_2$ se aplica.

Para mostrar que $\mathcal{R}^0_{proj} \subset \mathcal{R}_2$ para os α-DAGs, assuma, por simplicidade, que $G = (V, A)$ tem os seguintes arcos: (r, i), (i, j), (j, k), (j, f) e (k, f) (Figura 2.1). Ainda, G satisfaz as condições impostas e os dois caminhos orientados j-para-f tem o menor número possível de arcos. Estes caminhos podem ser maiores sem afetar o argumento que será utilizado. O mesmo também se aplica ao caminho r-para-i (r, i), que também pode ser maior.

Agora, assuma que na solução $(\overline{\mathbf{x}}, \overline{\mathbf{y}}) \in \mathcal{R}_0$ esses arcos possuem os seguintes valores: $\overline{\mathbf{x}}_{r,i} = 1$ e $\overline{\mathbf{x}}_{i,j} = \overline{\mathbf{x}}_{j,k} = \overline{\mathbf{x}}_{j,f} = \overline{\mathbf{x}}_{k,f} = 0.5$ (Figura 2.1). Observe que $\overline{\mathbf{x}}$ é uma

solução viável para $\mathcal{R}_2$, ou seja, $\overline{\mathbf{x}} \in \mathcal{R}_2$. Deixando de lado, por esse momento, as desigualdades *cutset* (2.4), note que $\overline{\mathbf{y}}_f = 1$ e $\overline{\mathbf{y}}_j = \overline{\mathbf{y}}_k = 0.5$ satisfaz todas as outras desigualdades de $\mathcal{R}_0$. Note, porém, que a desigualdade *cutset* (2.4) correspondente ao conjunto $S = \{j, k, f\}$ impõe que $\overline{\mathbf{x}}_{i,j} \geq \overline{\mathbf{y}}_f$ e consequentemente que $\overline{\mathbf{y}}_f \leq 0.5$.

Portanto, nenhum $\overline{\mathbf{y}} \in \mathbb{R}^{|V_0|}$ existe de forma que $(\overline{\mathbf{x}}, \overline{\mathbf{y}}) \in \mathcal{R}_0$ seja válido. Sendo assim, podemos concluir que $\mathcal{R}^0_{proj}$ está estritamente contido em $\mathcal{R}_2$ para os α-DAGs. Com isto, $\mathcal{R}_0$ é uma formulação mais forte que $\mathcal{R}_2$ para essa classe de grafos. □

2.4 Algoritmo Branch-and-Cut

O algoritmo Branch-and-Cut (BC) proposto nesta seção utiliza o pacote de otimização XPRESS versão 8.4.5 para resolver os problemas de programação linear e gerenciar a árvore de enumeração. No entanto, o gerador de cortes automático, testes de pré-processamento e heurísticas primais são mantidas desligadas.

Para esse algoritmo, apresentamos um procedimento de separação para as desigualdades DCS (2.4) na Subseção 2.4.1. Dado um grafo $G = (V, A)$, dois vértices distintos $s, t \in V$ e um subconjunto $S \subset V$ onde $s \in V \setminus S$ e $t \in S$, o corte $s - t$ é definido pelo subconjunto de arcos $\{(i, j) \in A : i \in V \setminus S, j \in S\}$.

2.4.1 Separação para as desigualdades *cutset* (DCS)

Seja $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ uma solução ótima do programa linear

$$\min \left\{ \sum_{a \in A} c_a x_a : (x, y) \in \overline{\mathcal{R}}_0 \right\}, \tag{2.33}$$

onde $\overline{\mathcal{R}}_0$ é uma região poliedral definida pela interseção das restrições (2.2)-(2.3), (2.5)-(2.7) e por eventuais DCS anteriormente separadas. Além dessas, quando associada a algum nó não raiz da árvore de enumeração, $\overline{\mathcal{R}}_0$ contém as restrições de *branching*.

Se $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ é uma solução inteira e seu grafo suporte $\overline{G} = (\overline{V}, \overline{A})$ é acíclico, $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ é uma solução viável de (2.1). Caso contrário, procuramos por desigualdades para o $\mathcal{R}_0$ que violem $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$.

Em nosso algoritmo *Branch-and-cut* para o MAP, as desigualdades potencialmente violadas são as DCS (2.4). Desta forma, o algoritmo de separação para estas desigualdades resolve múltiplas vezes um problema de fluxo máximo (Maximum Flow Problem - MFP). Em cada um deles tem-se a raiz r como vértice origem s, um vértice de $\overline{V}$ como destino t e arcos com pesos $\{\bar{x}_{ij} : (i, j) \in \overline{A}\}$. Sendo assim, as desigualdades (2.4) podem ser separadas em $O(|\overline{V}|^3)$.

Algoritmo 1: Algoritmo de separação para as restrições (2.4).

Entrada: Uma solução $(\bar{x}, \bar{y})$ de LP
Criar um grafo direcionado induzido $\overline{G}(\overline{V}, \overline{A})$ com valores não nulos de $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, onde $\overline{V} \subseteq V$ e $\overline{A} \subseteq A$;
$\overline{V}_0 \leftarrow \overline{V} \setminus \{r\}$;
para *cada arco* $a \in \overline{A}$ **faça**
    $\bar{c}_a \leftarrow \bar{x}_a$;
fim
$ativado\_ortogonal \leftarrow falso$;
para *cada vértice* $i \in \overline{V}_0$ **faça**
    se *i não foi separado* **então**
        Rodar algoritmo de fluxo máximo [21] da raiz até i ;
        se $maximum\_flow < y_i$ **então**
            Construir o corte $(V \setminus S_i, S_i)$, onde $raiz \in V \setminus S_i$ e $i \in S_i$;
            Adicionar restrição (2.4) associada a S_i e i ;
            Marcar i como separado ;
            para *cada vértice* $j \in S_i$ **faça**
                se $y_j < y_i$ **então**
                    Marcar j como separado ;
                fim
            fim
            se $y_i - maximum\_flow > limiar$ **então**
                $ativado\_ortogonal \leftarrow verdadeiro$;
            fim
            se *ativado_ortogonal* **então**
                para *cada arco* $a \in (V \setminus S_i, S_i)$ **faça**
                    $\bar{c}_a \leftarrow 1$;
                fim
            fim
        fim
    fim
fim

Ao resolver os MFPs definidos acima, assuma que $S_i \subset \overline{V}$ é o conjunto de vértices que define o corte de peso mínimo que separa r de i em $\overline{G}$. Adicionalmente, considere a soma dos pesos dos arcos de $\overline{G}$ que definem esse corte. Sempre que esse valor for inferior a $\overline{y}_k$, para qualquer $k \in S_i$, uma desigualdade DCS violada é identificada.

Os MFPs são resolvidos pelo algoritmo seguindo a ordem crescente dos índices de seus respectivos vértices de origem $s \in \overline{V}$. Quando uma DCS violada é identificada, digamos para o vértice $i \in V_0$, pode-se deixar de resolver o MFP subsequente correspondente a um vértice $j \in V_0$, $j > i$, dado que $j \in S_i$ e $\overline{y}_j < \overline{y}_i$ sejam satisfeitos. Além do mais, o grau de violação do plano de corte e a ortogonalidade também são consideradas como critérios de aceitação. Suponha que uma desigualdade seja aceita para ser incluída e seu grau de violação é superior ao dado *limiar* (em nossos

experimentos computacionais, nós usamos *limiar* = 0.7). Assim, a partir desse ponto, desigualdades violadas subsequentes só são consideradas se forem ortogonais às anteriores. Mais especificamente, quando uma desigualdade DCS violada é adicionada, as iterações subsequentes não consideram uma eventual desigualdade DCS violada associada com algum dos arcos desse corte. Para isso, o peso desses arcos são alterados para 1 no grafo suporte. Uma descrição detalhada da separação é feita no Algoritmo 1.

2.5 Novas Instâncias DAGs

Como veremos na seção de resultados computacionais, as instâncias DAGs resultantes do procedimento apresentado por Blum et al [2] são todas resolvidas pelo nosso algoritmo *Branch-and-bound* implementado sob a formulação (2.26).

Por esse motivo, vamos propor um novo conjunto de instâncias aleatórias de grafos direcionados acíclicos (DAGs). A topologia dessas instâncias são baseadas na topologia dos DAGs resultantes do algoritmo de redução do problema de Localização de Facilidades Não capacitado [12] (Uncapacitated Plant Location Problem) apresentado por Venkata e Rao [11].

Os DAGs resultantes do gerador de instâncias desta seção têm seus vértices divididos em níveis (N_0, N_1, ..., N_k). No nível N_0 temos apenas a raiz do grafo ($|N_0| = 1$) e o número de vértices dos níveis subsequentes está no intervalo $[\alpha_{min}, \alpha_{max}]$, onde α_{min} e α_{max} respectivamente representam o número mínimo e máximo de vértices por nível. Como ilustrado na Figura 2.2, existe um arco entre o vértice de N_0 e cada um dos vértices em N_1.

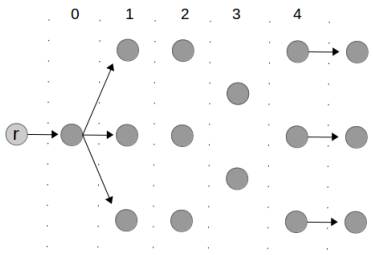

Figura 2.2: Níveis dos vértices do DAG aumentado resultante do gerador de instâncias com $k = 4$, $\alpha_{min} = 2$ e $\alpha_{max} = 3$.

Além desses níveis, existe um nível adicional N_{k+1}, onde $|N_{k+1}| = |N_k|$. Este nível adicional tem como objetivo impor que alguns dos vértices em N_k estejam obrigatoriamente na solução. Para isso, cada vértice em N_k é conectado ao seu vértice correspondente em N_{k+1}, como ilustra a Figura 2.2, por um arco de custo negativo -100. Note que, para resolver essas instâncias usando as formulações da

Seção 2.3, uma raiz artificial r deve ser conectada à raiz do grafo por um arco de custo zero (ver Figura 2.2).

As duas últimas etapas consistem em criar os arcos interníveis restantes e determinar seus respectivos custos. Com uma probabilidade $p_{arc} = 25\%$, criamos os arcos (i, j), para todo $i \in N_l$, $j \in N_t$ e $1 \leq l < t \leq k$. Por fim, custos não negativos no intervalo $[0, 100)$ são atribuídos a estes arcos e aos arcos saindo do nível N_0.

As Tabelas 2.1 e 2.2 apresentam 30 instâncias para o MAP restrito a DAGs que foram geradas a partir do procedimento descrito acima. As instâncias da Tabela 2.1 têm como característica grafos mais *finos*, ou seja, onde $k \geq \alpha_{max}$, e as instâncias da Tabela 2.2 têm como característica grafos mais *largos*, ou seja, onde $k < \alpha_{min}$.

k	α_{min}	α_{max}	$\|V\|$	$\|A\|$
20	11	16	306	985
20	12	17	314	1023
20	13	18	320	1062
20	14	19	356	1338
20	15	20	379	1509
25	16	21	494	2152
25	17	22	517	2365
25	18	23	539	2577
25	19	24	565	2843
25	20	25	581	2968
30	21	26	730	4028
30	22	27	759	4341
30	23	28	791	4717
30	24	29	821	5108
30	25	30	858	5603

Tabela 2.1: Instâncias DAGs com $p_{arc} = 25\%$ e $k \geq \alpha_{max}$ (*finos*).

k	α_{min}	α_{max}	$\|V\|$	$\|A\|$
20	21	26	500	2686
20	22	27	519	2854
20	23	28	537	3064
20	24	29	560	3384
20	25	30	587	3634
25	26	31	743	4935
25	27	32	770	5272
25	28	33	800	5656
25	29	34	809	5827
25	30	35	845	6280
30	31	36	1041	8156
30	32	37	1067	8635
30	33	38	1094	9042
30	34	39	1117	9471
30	35	40	1160	10199

Tabela 2.2: Instâncias DAGs com $p_{arc} = 25\%$ e $k < \alpha_{min}$ (*largos*).

2.6 Resultados Computacionais

Os experimentos deste capítulo foram conduzidos em um computador Intel Core i7 com 16 GiB de RAM e rodando a 2,70GHz. Apenas um processador foi utilizado e não foi permitido o uso de *multi-threading*. Para cada instância testada, foi dado um limite de duas horas.

O primeiro resultado reportado nesta seção é referente ao nosso algoritmo *Branch-and-cut* (BC) sob a formulação *cutset* (2.1) (BC-CSF - *BC Cutset Formulation*) e o algoritmo *Branch-and-bound* sob a formulação MCFF (2.9) (XPRESS-MCFF). Para esse resultado, consideramos as instâncias propostas por Duhamel et al. [1] definidas sob grafos direcionados gerais.

Em seguida, apresentamos os resultados do algoritmo BC sob as formulações

restritas a DAGs da Seção 2.3. Para esses resultados, estudamos o comportamento do nosso algoritmo para dois conjuntos de instâncias: (i) primeiro para as instâncias resultantes de uma reprodução do algoritmo de geração de instâncias proposto por Blum et al. [2] e, em seguida, (ii) para as instâncias apresentadas na Seção 2.5.

Instância				Limite Superior		Tempo CPU (s)		Speed-Up		#CS
Tipo	$\|V\|$	$\|A\|$	k	Duhamel	BC-CSF	MCFF	BC-CSF	MCFF	BC-CSF	
TA	50	250	03	-573	-573	0,77	0,01	1,00	144,14	26
TA	50	250	06	-800	-800	0,61	0,00	1,00	136,60	2
TA	50	500	03	-872	-872	2,13	0,01	1,00	343,33	0
TA	50	500	06	-1135	-1135	2,42	0,01	1,00	230,92	72
TA	100	500	03	-1524	-1524	5,09	0,01	1,00	676,06	0
TA	100	500	06	-4583	-4583	8,38	0,02	1,00	348,43	32
TA	100	500	09	-2654	-2654	5,87	0,01	1,00	749,92	0
TA	100	1000	03	-1708	-1708	46,19	0,02	1,00	2217,30	7
TA	100	1000	06	-780	-780	57,80	0,03	1,00	1993,06	19
TA	100	1000	09	-3085	-3085	61,47	0,02	1,00	3456,02	60
TA	200	1000	03	-6359	-6359	225,85	0,04	1,00	5197,21	45
TA	200	1000	06	-3909	-3929	160,13	0,04	1,00	4235,77	246
TA	200	1000	09	-8867	**-8878**	165,24	0,02	1,00	9865,93	0
TA	200	1000	12	-218	-218	245,96	0,22	1,00	1139,68	454
TA	200	2500	03	-5642	**-5648**	2192,94	0,26	1,00	8513,88	580
TA	200	2500	06	-9029	-9029	2498,47	0,05	1,00	46990,65	7
TA	200	2500	09	-7310	-7310	3398,52	0,09	1,00	36409,71	233
TA	200	2500	12	-1842	**-1883**	1981,47	0,05	1,00	43251,06	53
TA	400	1000	03	-14482	-14482	310,04	0,31	1,00	986,95	488
TA	400	1000	06	-12348	-12348	186,48	0,15	1,00	1220,90	289
TA	400	1000	09	-14764	-14764	159,92	0,10	1,00	1575,30	12
TA	400	1000	12	-11321	**-11336**	243,17	0,13	1,00	1863,52	34
TA	400	1000	15	-17426	-17426	173,01	0,24	1,00	717,26	38
TA	400	2000	03	-6628	-6628	6576,87	0,23	1,00	28048,30	59
TA	400	2000	06	-217	**-234**	4095,00	0,05	1,00	89670,56	3
TA	400	2000	09	-11448	-11448	3367,61	0,05	1,00	74222,08	2
TA	400	2000	12	-8697	**-8750**	5065,14	0,17	1,00	29469,87	40
TA	400	2000	15	-5949	-5949	4167,30	0,06	1,00	69173,51	7
TA	400	4000	03	-4628	-4628	249,71	0,18	1,00	1386,33	29
TA	400	4000	06	-7532	**-7540**	416,70	0,09	1,00	4894,21	1
TA	400	4000	09	-5697	**-5714**	165,61	0,10	1,00	1600,42	13
TA	400	4000	12	-12601	-12601	231,96	0,09	1,00	2601,53	1
TA	400	4000	15	-2245	**-2250**	267,10	0,29	1,00	916,50	169
Média Total						1113,18	0,10	1,00	14371,12	91,55

Tabela 2.3: Resultados para instâncias TA: DGs Duhamel et al. [1].

2.6.1 Resultados para DGs Duhamel et al. [1]

Os resultados computacionais desta seção têm como objetivo comparar os algoritmos XPRESS-MCFF e BC-CSF para o MAP sob as instâncias propostas em [1]. Essas

instâncias são separadas em dois grupos: TA (Truly Asymmetric) e SA (Symmetric with Asymmetric cost). As instâncias TAs são aquelas que, se existir o arco (i, j) não necessariamente existe o arco (j, i). Enquanto que para as instâncias SAs, se existir o arco (i, j) necessariamente existe o arco (j, i).

As Tabelas 2.3 e 2.4 comparam o limite superior do BC-CSF com o melhor limite superior reportado em Duhamel et al. [1] e o tempo de CPU dos algoritmos XPRESS-MCFF e BC-CSF. Os dois algoritmos foram executados no mesmo computador. As quatro primeiras colunas identificam a instância pelo tipo, número de vértices, número de arcos e o número de componentes negativas k. Em seguida,

Instância				Limite Superior		Tempo CPU (s)		Speed-Up		#CS
Tipo	$\|V\|$	$\|A\|$	k	Duhamel	BC-CSF	MCFF	BC-CSF	MCFF	BC-CSF	
SA	50	250	03	-208	-208	0,49	0,00	1,00	128,67	0
SA	50	250	06	-2169	-2169	0,52	0,00	1,00	108,24	4
SA	50	500	03	-2464	**-2470**	1,99	0,02	1,00	131,22	136
SA	50	500	06	-591	-591	1,39	0,01	1,00	136,75	0
SA	100	500	03	-1351	**-1370**	4,09	0,04	1,00	115,25	552
SA	100	500	06	-2264	**-2267**	5,50	0,02	1,00	307,42	35
SA	100	500	09	-858	**-862**	5,85	0,01	1,00	449,23	106
SA	100	1000	03	-972	-972	23,75	0,03	1,00	825,01	8
SA	100	1000	06	-3921	**-3929**	27,83	0,03	1,00	801,78	82
SA	100	1000	09	-4160	-4160	23,78	0,03	1,00	864,02	2
SA	200	1000	03	-765	**-781**	54,59	0,02	1,00	2428,02	2
SA	200	1000	06	-737	**-749**	47,90	0,07	1,00	685,62	181
SA	200	1000	09	-2932	**-2945**	44,78	0,03	1,00	1519,95	10
SA	200	1000	12	-7236	**-7258**	53,04	0,03	1,00	2001,18	3
SA	200	2500	03	-6635	**-6637**	1243,76	0,18	1,00	6984,95	9
SA	200	2500	06	-3705	**-3719**	1245,73	0,16	1,00	7558,94	69
SA	200	2500	09	-1817	**-1830**	1091,39	0,20	1,00	5596,58	103
SA	200	2500	12	-6392	**-6425**	889,30	0,19	1,00	4605,34	22
SA	400	1000	03	-15641	**-15668**	288,13	0,02	1,00	16918,66	0
SA	400	1000	06	-6162	**-6205**	174,10	0,05	1,00	3507,96	160
SA	400	1000	09	-3692	**-3722**	134,05	0,03	1,00	5269,88	96
SA	400	1000	12	-7286	**-7339**	144,18	0,09	1,00	1568,84	45
SA	400	1000	15	-9763	**-9796**	135,83	0,04	1,00	3478,84	9
SA	400	2000	03	-9272	**-9288**	801,46	0,07	1,00	11760,49	0
SA	400	2000	06	-11977	**-11999**	762,80	0,35	1,00	2193,09	81
SA	400	2000	09	-4426	**-4493**	735,73	0,08	1,00	9758,03	4
SA	400	2000	12	-1909	**-1950**	421,90	0,10	1,00	4349,99	186
SA	400	2000	15	-5305	**-5351**	771,89	0,18	1,00	4234,61	675
SA	400	4000	03	-9173	**-9200**	TL*	0,49	-	-	1
SA	400	4000	06	-16476	**-16520**	278,34	0,97	1,00	286,12	69
SA	400	4000	09	-13781	**-13808**	TL*	1,02	-	-	87
SA	400	4000	12	-8803	**-8869**	TL*	1,24	-	-	323
SA	400	4000	15	-2944	**-3017**	289,41	0,52	1,00	553,80	17
Média Total						323,45	0,19	1,00	3304,28	93,24

Tabela 2.4: Resultados para instâncias SA: DGs Duhamel et al. [1].

as duas colunas seguintes comparam o melhor limite superior conhecido (Duhamel) e o limite superior encontrado pelo BC-CSF. As duas colunas subsequentes reportam o tempo de CPU, em segundos: uma coluna para o algoritmo XPRESS-MCFF (abreviado como MCFF) e outra para o BC-CSF. Nas duas colunas que se seguem, é apresentado o *speed-up* de cada um dos dois algoritmos. Para cada instância de teste, seu *speed-up* corresponde à razão entre o tempo de CPU do algoritmo mais lento e o tempo de CPU necessário para o algoritmo resolver essa mesma instância. Sendo assim, *speed-up* é sempre igual a 1,00 para o algoritmo mais lento e maior que 1,00 para os algoritmos mais rápidos que ele. Por fim, a última coluna apresenta o número total de *cutsets generalizadas* (2.4) adicionadas pelo algoritmo BC-CSF.

Podemos notar nas Tabelas 2.3 e 2.4, que o algoritmo BC-CSF provou a otimilidade de todas as instâncias, melhorando assim o melhor limite superior, e ainda foi mais rápido que o XPRESS-MCFF em todas as instâncias. Diferentemente do XPRESS-MCFF que provou ter encontrado a solução ótima em apenas 63 instâncias. Além disso, o tempo médio e maior tempo de processamento do BC-CSF são respectivamente 0,15 e 1,24, enquanto que para o XPRESS-MCFF esses valores são respectivamente 718,32 e 6576,87, quando não execedeu o limite de tempo (TL).

2.6.2 Resultados para DAGs Blum et al. [2]

Os resultados computacionais desta seção focam em comparar duas versões do algoritmo *Branch-and-bound* para o MRAP restrito a DAGs. Os DAGs considerados para esses experimentos foram propostos por Blum et al. [2]. A primeira delas é implementada sob a formulação (2.20), denominada BB_2, proposta por Rao e Sridharan [6]. A segunda implementação é realizada sob a formulação *GSEC-2* (2.26), denominada BB_3, apresentada na Seção 2.3.2.

A Tabela 2.5 compara o tempo de CPU dos dois algoritmos para os DAGs propostos em Blum et al. [2]. As três primeiras colunas identificam a instância de teste pelo número de vértices $|V|$, probabilidade p_{arc} de existir um dado arco e probabilidade p_{neg} do custo associado a um arco ser negativo. O tempo de CPU, em segundos, é então reportado: uma coluna para o algoritmo BB_2 e outra para o BB_3. Nas duas colunas subsequentes, são apresentados os *speed-ups* de cada um dos dois algoritmos. As duas últimas colunas apresentam o número de nós da árvore de enumeração de cada versão do algoritmo *Branch-and-bound*.

Como podemos observar na Tabela 2.5, em todas as comparações conduzidas, BB_3 é mais rápido que BB_2. O *speed-up* médio e máximo correspondente ao BB_3 é respectivamente 265,02 e 2032,05. Além disso, para as instâncias com $|V| = 3000$ e $p_{arc} \in \{30, 50\}$, nota-se que o algoritmo BB_3 provou ter encontrado a solução ótima em um tempo médio de 103,88 segundos, enquanto que o algoritmo BB_2 atingiu o

Instância			Tempo CPU (s)		Speed-Up		Nós	
$\|V\|$	p_{arc}	p_{neg}	BB_2	BB_3	BB_2	BB_3	BB_2	BB_3
100	10	25	0,01	0,01	1,00	1,14	1	1
100	10	50	0,01	0,01	1,00	1,26	1	1
100	10	75	0,01	0,01	1,00	1,28	1	1
100	30	25	0,13	0,02	1,00	6,54	1	1
100	30	50	0,13	0,02	1,00	6,45	1	1
100	30	75	0,13	0,02	1,00	6,42	1	1
100	50	25	0,61	0,03	1,00	19,03	1	1
100	50	50	0,62	0,03	1,00	18,95	1	1
100	50	75	0,61	0,03	1,00	18,88	1	1
1000	10	25	3416,23	1,68	1,00	2032,05	1	1
1000	10	50	3322,10	1,88	1,00	1769,52	1	1
1000	10	75	3350,01	2,07	1,00	1620,60	1	1
1000	30	25	54,93	5,04	1,00	10,90	1	1
1000	30	50	54,90	5,45	1,00	10,08	1	1
1000	30	75	54,83	5,42	1,00	10,11	1	1
1000	50	25	8,11	7,49	1,00	1,08	1	1
1000	50	50	8,24	7,69	1,00	1,07	1	1
1000	50	75	8,28	7,61	1,00	1,09	1	1
3000	10	25	417,51	40,20	1,00	10,39	1	1
3000	10	50	417,22	43,96	1,00	9,49	1	1
3000	10	75	417,74	46,00	1,00	9,08	1	1
3000	30	25	LM^*	116,48	-	-	-	1
3000	30	50	LM^*	119,37	-	-	-	1
3000	30	75	LM^*	121,11	-	-	-	1
3000	50	25	LM^*	94,74	-	-	-	1
3000	50	50	LM^*	87,63	-	-	-	1
3000	50	75	LM^*	83,96	-	-	-	1
Média Total			549,16	29,55	1,00	265,02	1,00	1,00

Tabela 2.5: *Branch-and-bound*: DAGs Blum [2].

limite de memória (LM) de 12GiB estabelecido para esses resultados.

2.6.3 Resultados para DAGs da Seção 2.5

Nessa seção serão apresentados os resultados computacionais para as instâncias propostas nesse trabalho (ver Seção 2.5). As Tabelas 2.6 e 2.7 comparam os limites inferior encontrados pelas relaxações: $\mathcal{R}_2$ (2.25), $\mathcal{R}_3$ (2.32) e da $\mathcal{R}_3$ (2.32) fortalecida com as *cutsets* direcionadas DCS (2.4), denominada $\mathcal{R}_3^+$. Os resultados dessa seção são conduzidos com os DAGs apresentados na Seção 2.5. As primeiras três colunas dessas tabelas identificam a instância pela tripla k, α_{min} e α_{max}. Em seguida, a coluna *Opt* apresenta o custo da solução ótima de cada uma das instâncias. As três colunas subsequentes, apresentam o GAP percentual entre o limite inferior e o custo da solução ótima para cada uma das formulações. O limite encontrado pela formulação $\mathcal{R}_3^+$ não é necessariamente o melhor limite da relaxação linear, uma

vez que o número de cortes encontrado durante o algoritmo de plano de cortes foi limitado em 100. Ou seja, para os experimentos dessa seção incluimos uma condição de parada adicional ao Algoritmo 1. Por fim, as três últimas colunas apresentam o tempo de CPU, em segundos, necessários para cada formulação chegar ao seu respectivo limite inferior.

Nas Tabelas 2.6 e 2.7, podemos ver o impacto das desigualdades *cutset* direcionadas sob o limite da relaxação linear referente à formulação $\mathcal{R}_3$ para DAGs. Enquanto que para a formulação $\mathcal{R}_3^+$ o menor GAP e a média dos GAP são, respectivamente, 0,21% e 1,98%, para a formulação $\mathcal{R}_3$ esses valores são respectivamente 3,05% e 9,31%.

Instâncias			Opt	GAP (%)			Tempo (s)		
k	α_{min}	α_{max}		$\mathcal{R}_2$	$\mathcal{R}_3$	$\mathcal{R}_3^+$	$\mathcal{R}_2$	$\mathcal{R}_3$	$\mathcal{R}_3^+$
20	11	16	-417	25,07	25,07	1,10	0,02	0,02	0,22
20	12	17	-620	21,10	21,10	4,52	0,02	0,02	0,27
20	13	18	-497	27,13	27,13	4,31	0,02	0,02	0,33
20	14	19	-774	16,18	16,18	2,11	0,03	0,03	0,60
20	15	20	-1030	9,83	9,83	1,10	0,04	0,03	0,29
25	16	21	-744	21,41	21,41	6,67	0,06	0,06	1,21
25	17	22	-999	13,19	13,19	1,12	0,06	0,05	1,15
25	18	23	-1018	9,77	9,77	0,88	0,07	0,05	1,49
25	19	24	-1131	11,17	11,17	3,25	0,10	0,07	1,15
25	20	25	-1184	11,03	11,03	2,82	0,09	0,06	1,53
30	21	26	-1179	11,03	11,03	2,23	0,14	0,09	1,55
30	22	27	-1318	11,32	11,32	3,87	0,18	0,13	2,22
30	23	28	-1723	7,04	7,04	1,37	0,20	0,11	2,50
30	24	29	-1852	7,03	7,03	2,15	0,22	0,11	2,56
30	25	30	-1745	7,41	7,41	1,92	0,28	0,13	3,83
Média Total				13,98	13,98	2,63	0,10	0,07	1,39

Tabela 2.6: Nó raiz da árvore de enumeração: DAGs *finos*.

O próximo conjunto de resultados compara duas versões do algoritmo *Branch-and-bound* e um algoritmo *Branch-and-cut* para o MAP restrito a DAGs. Os dois algoritmos *Branch-and-bound* são respectivamente implementados sob a formulação Rao (2.25), denominado BB_2, e sob a formulação GSEC-2 (2.32), denominado BB_3. O algoritmo *Branch-and-cut*, denominado BC_3^+, é implementado sob a formulação GSEC-2 (2.32) fortalecido com as desigualdades *cutset* direcionadas (2.4). Vale ressaltar que em cada nó da árvore de enumeração do BC_3^+, o algoritmo de plano de cortes realiza a separação das *cutsets* somente se o limite máximo de 100 cortes para cada nó não tiver sido excedido.

As Tabelas 2.8 e 2.9 comparam o tempo de execução e o número de nós explorados pela árvore de enumeração de cada um dos três algoritmos. Novamente, as primeiras três colunas destas tabelas identificam a instância pela tripla k, α_{min} e α_{max}. O tempo de CPU, em segundos, é então reportado: uma coluna para o algoritmo BB_2,

Instâncias			Opt	GAP (%)			Tempo (s)		
k	α_{min}	α_{max}		$\mathcal{R}_2$	$\mathcal{R}_3$	$\mathcal{R}_3^+$	$\mathcal{R}_2$	$\mathcal{R}_3$	$\mathcal{R}_3^+$
20	21	26	-1227	7,66	7,66	2,13	0,09	0,05	1,15
20	22	27	-1637	4,70	4,70	0,71	0,10	0,06	0,85
20	23	28	-1708	3,36	3,36	0,21	0,12	0,06	0,48
20	24	29	-1484	6,22	6,22	1,51	0,14	0,07	1,72
20	25	30	-1914	3,71	3,71	0,64	0,18	0,08	0,98
25	26	31	-1926	5,18	5,18	1,41	0,29	0,14	2,53
25	27	32	-2080	5,25	5,25	1,26	0,31	0,12	3,50
25	28	33	-2145	4,15	4,15	1,09	0,41	0,12	1,98
25	29	34	-2171	4,93	4,93	2,09	0,37	0,15	2,61
25	30	35	-2290	4,79	4,79	2,21	0,51	0,20	2,60
30	31	36	-2518	4,80	4,80	1,54	0,72	0,22	5,31
30	32	37	-2554	4,01	4,01	1,33	0,80	0,19	3,30
30	33	38	-2803	3,77	3,77	1,34	0,90	0,30	5,09
30	34	39	-2660	3,99	3,99	1,51	1,01	0,28	5,35
30	35	40	-3031	3,05	3,05	0,94	1,47	0,27	5,36
Média Total				4,64	4,64	1,33	0,49	0,15	2,85

Tabela 2.7: Relaxação Linear: DAGs *largos*.

outra para o algoritmo BB_3 e uma terceira para o algoritmo BC_3^+. Nas três colunas subsequentes, são apresentados os *speed-ups* de cada algoritmo. O número de nós explorados por cada algoritmo é então reportado nas últimas três colunas.

Como podemos observar nas Tabelas 2.8 e 2.9, para a maioria das comparações, BC_3^+ foi mais rápido que BB_2 e BB_3, embora o BB_3 já tenha sido mais rápido que BB_2. Mais especificamente, BC_3^+ foi mais rápido em 27 instâncias, enquanto que em apenas 3 instâncias o algoritmo BB_3 foi mais rápido. Além disso, o *speed-up* médio e máximo correspondente ao BC_3^+ é de respectivamente 6,92 e 28,62. Em constraste a esse cenário, para o BB_3, estes valores são respectivamente 1,51 e 3,68 e para o BB_2 estes valores são respectivamente 1,06 e 1,79. Para cada algoritmo, desconsiderando aquelas instâncias em que o algoritmo foi mais lento, o *speed-up* médio para o BC_3^+ é de 6,35, enquanto que para o BB_3 e o BB_2 este valor é de, respectivamente, 1,58 e 1,27. Comparando a média do número de nós da árvore de enumeração, podemos notar que para o BC_3^+ esse valor foi de 42,77, enquanto que para o BB_3 foi de 37.605,50 e para o BB_2 foi de 61.269,63. Por último, note que para a instância $k = 30$, $\alpha_{min} = 34$ e $\alpha_{max} = 39$ os algoritmos BB_2 e BB_3 excederam o limite de tempo de 2h, contudo, o melhor limite inferior encontrado ao final do algoritmo BB_3 foi de -2666,47 e ao final do algoritmo BB_2 foi de -2673,00. Para essa mesma instância, o algoritmo BC_3^+ conseguiu provar a otimalidade da solução encontrada em 587,36 segundos.

Instância			Tempo CPU (s)			Speed-Up			Nós		
k	α_{min}	α_{max}	BB_2	BB_3	BC_3^+	BB_2	BB_3	BC_3^+	BB_2	BB_3	BC_3^+
20	11	16	1,60	1,19	1,11	1,00	1,34	1,45	1203	783	5
20	12	17	5,69	5,31	1,80	1,00	1,07	3,16	4441	2907	7
20	13	18	5,06	2,51	3,57	1,00	2,02	1,42	3565	1283	19
20	14	19	7,53	5,02	4,79	1,00	1,50	1,57	4547	2333	13
20	15	20	3,70	1,88	1,32	1,00	1,97	2,81	1103	345	3
25	16	21	655,00	343,10	44,27	1,00	1,91	14,79	145197	52328	135
25	17	22	74,30	56,05	4,32	1,00	1,33	17,20	18899	10617	7
25	18	23	13,62	9,85	5,16	1,00	1,38	2,64	3529	2073	7
25	19	24	93,13	49,62	25,14	1,00	1,88	3,70	27863	11450	45
25	20	25	80,03	61,78	29,19	1,00	1,30	2,74	30977	17994	35
30	21	26	230,85	140,82	15,14	1,00	1,64	15,24	52379	18164	23
30	22	27	500,96	545,86	46,60	1,09	1,00	11,71	103063	80576	31
30	23	28	108,29	193,36	49,64	1,79	1,00	3,90	14529	20383	39
30	24	29	280,05	180,78	120,12	1,00	1,55	2,33	56459	26966	65
30	25	30	394,60	399,51	50,38	1,01	1,00	7,93	48823	40603	25
Média Total			163,63	133,11	26,84	1,06	1,46	6,17	34438,47	19253,67	30,60

Tabela 2.8: *Branch-and-cut*: DAGs *finos*.

Instância			Tempo CPU (s)			Speed-Up			Nós		
k	α_{min}	α_{max}	BB_2	BB_3	BC_3^+	BB_2	BB_3	BC_3^+	BB_2	BB_3	BC_3^+
20	21	26	7,66	7,09	6,62	1,00	1,08	1,16	1805	1346	11
20	22	27	8,15	5,63	4,52	1,00	1,45	1,80	1479	717	11
20	23	28	4,57	4,60	2,85	1,01	1,00	1,61	761	691	7
20	24	29	18,49	15,60	29,01	1,57	1,86	1,00	4569	2445	39
20	25	30	11,90	7,56	8,97	1,00	1,57	1,33	2401	1085	23
25	26	31	121,47	85,00	81,29	1,00	1,43	1,49	15889	7148	69
25	27	32	795,87	270,19	82,34	1,00	2,95	9,67	93857	19560	65
25	28	33	124,06	82,62	27,16	1,00	1,50	4,57	18021	8153	17
25	29	34	602,64	163,86	76,17	1,00	3,68	7,91	72403	14915	57
25	30	35	1109,65	889,20	249,33	1,00	1,25	4,45	106671	64489	134
30	31	36	3645,07	3042,25	192,54	1,00	1,20	18,93	276749	164120	37
30	32	37	1247,69	787,72	157,48	1,00	1,58	7,92	140237	63720	69
30	33	38	2600,05	3602,71	125,88	1,39	1,00	28,62	148799	163939	33
30	34	39	TL*	TL*	587,36	1,00	1,00	12,26	282286	213373	173
30	35	40	2772,07	2954,75	242,95	1,07	1,00	12,16	155585	113599	79
Média Total			1351,29	1274,58	124,96	1,07	1,57	7,66	88100,80	55953,33	54,93

Tabela 2.9: *Branch-and-cut*: DAGs *largos*.

Capítulo 3

Problema da Árvore Geradora Capacitada de Custo Mínimo

3.1 Definição do problema

Seja $G = (V, E)$ um grafo não direcionado com um conjunto de vértices V e um conjunto de arestas E, e assuma que *demandas* $\{d_i \in \mathbb{N} : i \in V\}$ e *custos* $\{c_e \in \mathbb{R} : e \in E\}$ estão respectivamente associados aos vértices e às arestas. Um dos vértices de G, denotado por $r \in V$, é a raiz de qualquer árvore viável e possui demanda zero ($d_r = 0$). Uma *árvore geradora capacitada* de G é um subgrafo $T = (V, E')$ com $E' \subseteq E$ que satisfaça as seguintes condições: (i) $|E'| = |V| - 1$, (ii) existe um único caminho simples de r até cada um dos vértices em $V \setminus \{r\}$ e (iii) para cada subárvore conectada a r, a soma das demandas dos seus vértices não deve exceder uma capacidade $C \in \mathbb{N}$. O objetivo é encontrar uma *arvore geradora capacitada* T com menor custo possível, onde o custo de T é dado por $\sum_{e \in E'} c_e$.

Dada uma raiz r, o problema da Árvore Geradora Capacitada de Custo Mínimo (CMSTP - *Capacitated Minimum Spanning Tree Problem*) consiste em encontrar a *árvore geradora capacitada* de menor custo em G. Esse problema NP-difícil tem importantes aplicações em projetos de rede e é bem estudado na literatura. Parte dela é dedicada apenas à versão com demandas unitárias (UD - *unitary demands*), onde todas demandas $d_i = 1$. Neste capítulo, vamos estudar a literatura do problema das *árvores geradoras capacitadas* de G.

3.2 Revisão Bibliográfica

Diversas soluções exatas podem ser encontradas na literatura do CMSTP [4, 22–30]. Gouveia e Martins apresentaram um algoritmo *branch-and-cut* sob uma formulação *hop-indexed* limitado apenas à versão UD [26, 27, 29]. Essa formulação apresenta

bons limites inferiores, mas o número de variáveis é $O(|E|C)$. Para o caso com demandas gerais (non-UD), [25] apresentou um algoritmo *branch-and-cut* sob uma formulação com arcos e [28] desenvolveu uma relaxação Lagrangeana. Esses algoritmos fazem uso de cortes poliedrais investigados em [10, 31–33]. Apesar disso, os limites inferiores encontrados não eram muito apertados. Uchoa et al. [4] notou que essas formulações, mesmo utilizando cortes, não conseguem capturar a estrutura do problema da mochila associado ao CMSTP. Com isso, torna-se atraente formular o CMSTP como um problema de partição. Essa formulação é uma reformulação *Dantzig-Wolfe* para o problema sob um grafo direcionado, onde as colunas são estruturas *q-arbs*. O problema de *pricing* dessa reformulação com *q-arbs* é uma relaxação do problema de *pricing* de uma reformulação similar, onde as colunas são arborescências não necessariamente geradora capacitadas. Como veremos no próximo capítulo, o problema de encontrar essa arborescência capacitada é NP-difícil. Apesar de ser uma relaxação, as *q-arbs* ainda conseguem capturar parte da estrutura do problema da mochila associado ao CMSTP e o problema de *pricing* associado pode ser resolvido por um algoritmo pseudo-polinomial. As instâncias da *OR-library* foram utilizadas como *benchmark* e com o algoritmo de *branch-cut-and-price*, [4] conseguiu resolver todas as instâncias com tamanho até 100 vértices e muitas das instâncias com até 160 vértices.

3.3 Formulações e Desigualdades

Nesta seção, vamos apresentar as formulações existentes na literatura do CMSTP. A primeira delas é definida com variáveis de arco e a segunda usando variáveis indexadas por capacidade. Por último, uma reformulação *Dantzig-Wolfe* é descrita.

3.3.1 Formulação baseada em arcos

A primeira formulação é definida sob um grafo direcionado $D = (V, A)$, onde A tem um par de arcos (i, j) e (j, i) para cada aresta $\{i, j\} \in E$ de G, exceto as arestas $\{r, i\}$ adjacentes à raiz, que são transformados em apenas um arco (r, i). Os custos de cada arco são os mesmos da sua aresta correspondente no grafo original. Assim, o objetivo é encontrar a arborescência geradora capacitada de custo mínimo enraizada em r no grafo direcionado D. Vamos denotar por V_0 o conjunto de vértices não raiz, ou seja, $V_0 = V \setminus \{r\}$.

A formulação para o CMSTP baseada em arcos usa variáveis x_a para indicar se um arco $a \in A$ pertence à solução. Dado um conjunto $S \subseteq V$ temos $d(S) = \sum_{i \in S} d_i$ e $k(S) = \lceil d(S)/C \rceil$. A função $k(S)$ é uma aproximação do número de subárvores

necessárias para atender o subconjunto S. Assim, a formulação é dada por:

$$\min \left\{ \sum_{a \in A} c_a x_a : x \in \Pi_1 \cap \mathbb{Z}^{|A|} \right\}, \tag{3.1}$$

onde Π_1 é a região poliedral definida por:

$$\sum_{(j,i) \in A} x_{ji} = 1, \ \forall i \in V_0 \tag{3.2}$$

$$\sum_{(j,i) \in \delta^-(S)} x_{ji} \geq k(S), \ \forall S \subseteq V_0 \tag{3.3}$$

$$0 \leq x_a \leq 1, \ \forall a \in A \tag{3.4}$$

Note que, com exceção de r, todo vértice tem exatamente um arco chegando nele, como exigido por (3.2). Além disso, as desigualdades (3.3) são *Capacity Cuts* que garantem a conectividade e a capacidade das soluções viáveis. Mais especificamente, o número de arcos entrando em S deve ser superior ou igual ao limite inferior do número mínimo de subárvores $k(S)$ necessárias para atender S. Em conjunto com (3.2), elas também previnem a formação de circuitos e garantem a existência de um caminho direcionado de r para cada um dos vértices de $V \setminus \{r\}$.

3.3.2 Desigualdade *Multistar*

As desigualdades *multistar* foram introduzidas por Araque, Hall e Magnanti [31] para o CMSTP e o problema de roteamento de veículos capacitados (Capacitated Vehicle Routing Problem - CVRP). Essas desigualdades são restrições de conectividade capacitadas, e têm diferentes expressões dependendo do tipo da demanda considerada (UD ou non-UD). Para o caso unitário (UD), as seguintes desigualdades são facetas sob condições identificadas em [25]:

$$\sum_{i \in S} \sum_{j \in S} C x_{ij} + \sum_{i \in S} \sum_{\substack{j \in V \setminus S \\ j \neq r}} (x_{ij} + x_{ji}) \leq |S|(C-1), \qquad \forall S \subseteq V_0 \tag{3.5}$$

Para mostrar que a desigualdade (3.5) é válida para o problema, vamos partir do caso extremo em que nenhum dos arcos em $A(S) = \{(i,j) : i \in S, j \in S\}$ pertencem à solução. Em seguida, por construção, iremos adicionar arcos de $A(S)$ à solução e assim perceber que o número de arcos adjacentes a S diminui em C unidades. Para essa prova, não consideramos os arcos da raiz quando nos referimos aos arcos adjacentes a um vértice de S ou aos adjacentes ao conjunto S.

No caso extremo descrito acima, cada vértice de S será uma componente conexa

do subgrafo induzido por S ($G[S]$). Nesse cenário, pode-se inferir que o número máximo de arcos adjacentes a um vértice de $i \in S$ será $C - 1$, que é o número máximo de vértices na mesma subárvore que o vértice i pertence. Assim temos que a desigualdade é válida para esse caso.

Suponha agora que um único arco $(i, j) \in A(S)$ seja adicionado à solução. Com esse novo arco, temos que i e j formam uma componente conexa em $G[S]$ e assim a contribuição desses vértices para o limite máximo de arcos adjacentes a S que era de $(C-1)+(C-1)$ passa a ser de $C-2$, que é o número máximo de vértices que podem estar na mesma subárvore que os vértices i e j pertencem. Ou seja, diminuímos C unidades do limite de arcos que são adjacentes a S. Note que o primeiro termo da desigualdade é responsável por subtrair C unidades do lado direito para cada arco em $A(S)$ na solução e sendo assim, temos que a desigualdade continua válida com esse novo arco na solução. Esse processo pode ser facilmente generalizado, uma vez que um novo arco sempre irá unir duas componentes conexas em $G[S]$. Seja a e b os respectivos tamanhos das componentes. Nesse caso, essas componentes contribuem respectivamente com $C - a$ e $C - b$ arcos e, ao serem conectadas por um novo arco, irá contribuir com $C - (a + b)$, ou seja, menos C unidades. Logo, pode-se concluir que a desigualdade (3.5) é válida.

Em seguida, Hall [25] propõe uma versão da desigualdade multistar (3.5) que se aplica também ao CMSTP com demandas unitárias e não unitárias. Seguindo essa mesma linha, Gouveia e Lopes [10] apresentam uma outra versão para essas desigualdades:

$$\sum_{i \in V \setminus S} \sum_{j \in S} (C - d_i) x_{ij} - \sum_{j \in S} \sum_{i \in V \setminus S} d_i x_{ji} \geq d(S), \qquad \forall S \subseteq V_0 \tag{3.6}$$

Essa desigualdade garante que a capacidade disponível ao entrar em S, subtraída da demanda consumida ao sair de S, seja suficiente para suprir a demanda de S. Mais especificamente, elas são desigualdades de fluxo em redes que tem como objetivo equalizar a capacidade disponível com a demanda dos vértices no conjunto $S \subseteq V_0$. Letchford e Salazar-González [34] descrevem um algoritmo exato sua separação.

3.4 Desigualdade Root-Cutset

A desigualdade *Root-Cutset* foi apresentada inicialmente por Hall [25] somente para o caso UD do CMSTP. Zhang [35], por sua vez, apresenta uma versão dessa desigualdade para o caso Non-UD. Essa desigualdade tem como objetivo fortalecer a conectividade com a raiz.

Para o caso UD, considere $S \subseteq V_0$ e $k(S)$ como definido anteriormente. Agora assuma que S seja distribuído entre as $k(S)$ sub-arborescências, de modo que a

quantidade de sub-arborescências contendo C vértices de S seja minimizada, e $b = \max\{0, |S| - k(S)(C-1)\}$. Nesse caso, temos a seguinte desigualdade válida:

$$\sum_{a \in A(S)} bx_a \leq b(|S| - k(S) - 1) + \sum_{v \in S} x_{rv}, \qquad \forall S \subseteq V_0 \tag{3.7}$$

Araque [31] mostrou que essas desigualdades definem facetas em certos casos.

Para o caso Non-UD, essa desigualdade não é tão interessante. Sendo assim, Hall [25] apresenta uma versão generalizada para (3.7) e uma heurística para fazer a separação. Essa generalização é dada por:

$$\frac{k(S)+1}{k(S)} \sum_{i \in \bar{B}} \sum_{j \in S} x_{ij} + \sum_{i \in B} \sum_{j \in S} x_{ij} \geq k(S) + 1, \qquad \forall S \subseteq V_0 \tag{3.8}$$

onde $B = \{i \notin S : k(S \cup \{i\}) > k(S)\}$ é o conjunto de vértices que não estão em S e que aumentaria o número de sub-arborescências necessárias caso algum deles fossem adicionados em S, e $\bar{B} = V_0 \setminus \{S \cup B\}$ são os vértices que não aumentariam.

3.5 Formulação com variáveis capacitadas

Gouveia [36] apresenta uma formulação que usa um conjunto de variáveis binárias x_a^d. Tal conjunto de variáveis indica se um arco $a = (i, j)$ pertence à solução e se a demanda total de todos os vértices da sub-arborescência enraizada em j é exatamente d. Essa formulação é então dada por:

$$\min \left\{ \sum_{a \in A} \sum_{d=1}^{C} c_a x_a^d : x \in \Pi_2 \cap \mathbb{Z}^{|A|.|C|} \right\}, \tag{3.9}$$

onde Π_2 é a região poliedral definida por:

$$\sum_{a \in \delta^-(i)} \sum_{d=1}^{C} x_a^d = 1 \qquad , \forall i \in V_0 \tag{3.10}$$

$$\sum_{a \in \delta^-(i)} \sum_{d=1}^{C} d x_a^d - \sum_{a \in \delta^+(i)} \sum_{d=1}^{C} d x_a^d = d_i \qquad , \forall i \in V_0 \tag{3.11}$$

$$0 \leq x_a^d \leq 1 \qquad , \forall a \in A, d = 1, ..., C. \tag{3.12}$$

A restrição de grau (3.10) impõe que cada vértice, exceto a raiz, tenha exatamente um arco chegando. A restrição de conservação de fluxo de capacidade (3.11) previne ciclos e sub-arborescências com capacidade maior que C.

3.6 Reformulação *Dantzig-Wolfe*

O CMSTP pode ser reformulado usando exponencialmente muitas variáveis e restrições. Nesse caso, as variáveis podem ser arborescências, não necessariamente geradoras, com exatamente um arco saindo da raiz e não excedendo a capacidade C. Ou ainda, podem ser uma estrutura apresentada por Uchoa et al. [4], chamada de *q-arbs*. Vale ressaltar que essa última é uma relaxação da primeira.

Para ambas estruturas, podemos formular o CMSTP usando um conjunto de variáveis λ_j que indica se a j-ésima, de um total de p, arborescências (ou *q-arbs*) está presente na solução. Seja q_a^j o número de vezes que o arco a aparece na arborescência j. Podemos reformular o problema da seguinte maneira:

$$\min \left\{ \sum_{a \in A} c_a x_a : (x, \lambda) \in \Pi_3 \cap \mathbb{Z}^{|V_0|+p} \right\}, \tag{3.13}$$

onde Π_3 é a região poliedral definida por:

$$\sum_{j=1}^{p} q_a^j \lambda_j - x_a = 0, \ \forall a \in A \tag{3.14}$$

$$\sum_{(j,i) \in A} x_{ji} = 1, \ \forall i \in V_0 \tag{3.15}$$

$$\sum_{(j,i) \in \delta^-(S)} x_{ji} \geq k(S), \ \forall S \subseteq V_0 \tag{3.16}$$

$$0 \leq x_a \leq 1, \ \forall a \in A \tag{3.17}$$

$$0 \leq \lambda_j \leq 1, \ j = 1, ..., p \tag{3.18}$$

Essa formulação possui todas as variáveis e restrições da formulação com arcos (3.1), mais as novas colunas (variáveis) λ e restrições (3.14) que definem x como sendo uma soma ponderada das colunas λ.

Substituindo todas as ocorrências de x_a na formulação (3.13) por sua equivalência dada pela restrição (3.14), podemos obter um LP mais compacto para resolver a relaxação linear usando geração de linhas e colunas. Para cada coluna, temos um valor q_a^j associado a cada arco $a = (i, j)$. Esse valor define o número de vezes em que o arco a aparece na coluna j. Note que, quando o problema de *pricing* for uma *arborescência capacitada*, $0 \leq q_a^j \leq 1$ são os possíveis valores de q_a^j. Como resultado, temos o seguinte LP denotado por *Dantzig-Wolfe Master* (DWM):

$$\min \left\{ \sum_{j=1}^{p} (\sum_{a \in A} c_a q_a^j) \lambda_j : \lambda \in \Pi_4 \cap \mathbb{Z}^p \right\}, \tag{3.19}$$

onde Π_4 é a região poliedral definida por:

$$\sum_{j=1}^{p} (\sum_{a \in \delta^-(i)} q_a^j) \lambda_j = 1, \ \forall i \in V_0 \tag{3.20}$$

$$\sum_{j=1}^{p} (\sum_{a \in \delta^-(S)} q_a^j) \lambda_j \geq k(S), \ \forall S \subseteq V_0 \tag{3.21}$$

$$0 \leq \lambda_j \leq 1, \ j = 1, ..., p \tag{3.22}$$

O custo reduzido da variável λ_j é a soma dos custos reduzidos dos arcos correspondentes na arborescência ou *q-arb*. Sejam ω e π variáveis duais associadas respectivamente às restrições (3.20) e (3.21). O custo reduzido $\bar{c}_a$ de um arco a é dado por:

$$\bar{c}_a = c_a - \omega_j - \sum_{S: a \in \delta^-(S)} \pi_S \qquad , \forall a = (i, j) \in A \tag{3.23}$$

Outras famílias de desigualdades também podem ser adicionadas ao problema *Master*. Dado um corte genérico $\sum_{a \in A} \alpha_a x_a \geq b$, podemos incluir $\sum_{j=1}^{p} (\sum_{a \in A} \alpha_a q_a^j) \lambda_j \geq b$. Esse novo corte contribui com $-\alpha_a \beta$ no cálculo do custo reduzido $\bar{c}_a$, onde β é a nova variável dual. Com os novos cortes, apenas os custos dos subproblemas são alterados, sua estrutura permanece a mesma.

Uchoa et al. [4] sugere uma reformulação DW, usando as variáveis de (3.9) no modelo (3.1) e considerando ainda as restrições (3.28). Dessa vez, vamos definir q_a^{dj} como o número de arcos a carregando exatamente d unidades de capacidade na j-ésima arborescência ou *q-arb*. Como resultado, temos o seguinte LP *Dantzig-Wolfe Master* (DWM):

$$\min \left\{ \sum_{d=1}^{C} \sum_{j=1}^{p} (\sum_{a \in A} c_a q_a^{dj}) \lambda_j : \lambda \in \Pi_5 \cap \mathbb{Z}^p \right\}, \tag{3.24}$$

onde Π_5 é a região poliedral definida por:

$$\sum_{j=1}^{p}(\sum_{d=1}^{C}\sum_{a\in\delta^-(i)} q_a^{dj})\lambda_j = 1,\ \forall i \in V_0 \tag{3.25}$$

$$\sum_{j=1}^{p}(\sum_{d=1}^{C}\sum_{a\in\delta^-(S)} q_a^{dj})\lambda_j \geq k(S),\ \forall S \subseteq V_0 \tag{3.26}$$

$$0 \leq \lambda_j \leq 1,\ j = 1, .., p \tag{3.27}$$

Das reformulações DWM (3.13) e (3.24) surge o problema de *pricing* da geração de colunas chamado Arborescência Capacitada de Custo Mínimo. Esse problema é estudado no capítulo seguinte e tem como aplicação encontrar limites duais mais apertados para CMSTP que as *q-arbs*.

3.7 Corte Estendido de Capacidade

Uchoa et al. [4] apresenta uma nova família de cortes válidos para a formulação (3.9). Cada corte é resultante da soma de algumas das restrições em (3.11) excluindo-se os arcos internos a S. Destes somatórios, correspondente a um conjunto de vértices $S \subseteq V_0$, tem-se as seguintes igualdades:

$$\sum_{a\in\delta^-(S)}\sum_{d=1}^{C} dx_a^d - \sum_{a\in\delta^+(S)}\sum_{d=1}^{C} dx_a^d = d(S) \qquad ,\forall S \subseteq V_0 \tag{3.28}$$

Como definido em [4], um Corte de Capacidade Extendido (ECC - *Extended Capacity Cut*) de S é qualquer desigualdade valida para o *o poliedro dado pela envoltória convexa das soluções 0-1 de (3.1).*

As desigualdades de Capacidade (3.3) e *Root Cutsets* podem ser derivadas a partir das igualdades ECC (3.28). Ao invés de separar as desigualdades ECC, os autores sugerem utilizar os algoritmos de separação específicos das desigualdades de Capacidade (3.3) e *Root Cutsets* para encontrar cortes violados e então fazer um *lifting* dos coeficientes (no espaço extendido).

3.8 Conclusão

Este capítulo investigou o problema da Árvore Geradora Mínima Capacitada (Capacitated Minimum Spanning Tree Problem - CMSTP). Inicialmente, fizemos uma revisão bibliográfica dos métodos exatos existentes para o problema. Em seguida, estudamos decomposições Dantzig-Wolfe para o CMSTP. Isso porque, num complemento futuro a esta pesquisa de doutorado, pretendemos introduzir um novo algoritmo de geração de colunas para o CMSTP. Tal algoritmo, em boa medida, vai

se basear no material desenvolvido nesta tese e terá como a referência a bater, o algoritmo proposto em [4].

Em sua versão (incompleta) atual, nosso algoritmo de geração de colunas para o CMSTP inicialmente resolve o problema de *pricing*, de forma heurística. Caso uma solução viável de custo reduzido negativo não venha a ser obtida, o problema é então resolvido de forma exata, através de um algoritmo do tipo *branch-and-cut* (ambos os algoritmos são descritos no Capítulo 4). Em complemento a isso pretendemos, futuramente, introduzir (no problema mestre associado ao algoritmo) os planos-de-corte descritos neste capítulo. Pelos resultados do Capítulo 4, é possível observar que nosso algoritmo, em sua versão atual, sem a introdução de planos-de-corte no problema mestre, é competitivo com uma versão similar do algoritmo em [4]. Em função desses resultados, é de se esperar que sua versão completa, com planos-de-corte, ele também venha a ser competitivo com o algoritmo em [4].

Capítulo 4

Problema da Arborescência Capacitada de custo mínimo

4.1 Definição

Seja $G = (V, A)$ um grafo direcionado, onde V é um conjunto de vértices e A um conjunto de arcos e assuma que demandas $\{d_i \in \mathbb{N} : i \in V\}$ e custos $\{c_a \in \mathbb{R} : a \in A\}$ estão respectivamente associados aos vértices e arcos de G. Nesse capítulo vamos investigar *arborescências capacitadas* de G. Isto é, subgrafos $T = (V_T, A_T)$ com $V_T \subseteq V$ e $A_T \subseteq A$, que satisfazem as condições: (i) $|A_T| = |V_T| - 1$, (ii) existe um (único) vértice $r \in V_T$ do qual partem caminhos direcionados em T com destino a cada um dos outros vértices em V_T e (iii) a soma das demandas dos vértices de T não deve exceder uma dada capacidade $C \in \mathbb{N}$. Subgrafos T são então denominados *r-arborescências-capacitadas* de G, onde r é o vértice raiz da arborescência. Nosso objetivo é identificar *arborescências capacitadas* de G onde a soma dos custos dos arcos seja a menor possível.

Ainda, distinguimos os casos onde o vértice $r \in V$ da arborescência é predefinida ou não. Se a raiz é predefinida, encontrar a *r-arborescência-capacitada* de menor custo de G é denominado como o problema da Arborescência Enraizada Capacitada de custo mínimo (Capacitated Minimum Rooted Arborescence Problem - CMRAP). Por outro lado, se nenhuma raiz for imposta explicitamente, o problema é denominado como o problema da Arborescência Capacitada de custo mínimo (Capacitated Minimum Arborescence Problem - CMAP). Assim como no capítulo anterior, para o problema CMSTP, existe uma distinção quanto ao tipo da demanda: UD ou Non-UD . Além de ser um subproblema da decomposição *Dantzig-Wolfe* (3.13) e (3.19), esse problema é intrinsecamente interessante por suas aplicações em redes de telecomunicações.

Como visto no capítulo anterior, Uchoa et al. [4] descreve esse problema ao

apresentar a decomposição *Dantzig-Wolfe* para o CMSTP. Excluindo essa descrição, até onde nós sabemos, esse problema ainda não foi estudado pela literatura. Os autores em [4] propõem uma estrutura denominada *q-arbs* para resolver o problema de *pricing* da geração de colunas. Essa estrutura é uma relaxação para o CMRAP. O CMRAP com $d_i = 1$, $\forall i \in V$ e $C = |V|$, é exatamente o problema de encontrar a arborescência, não necessariamente geradora, de custo mínimo (Minimum Rooted Arborescence Problem - MRAP) estudado no Capítulo 2. Como o MRAP foi classificado como NP-Difícil por Duhamel et al. [1], podemos concluir que CMRAP também é NP-Díficil.

As principais contribuições deste capítulo são: um algoritmo *Branch-and-cut* (BC) para o CMAP, uma família de desigualdades válidas, um algoritmo de separação para estas desigualdades e uma heurística primal para o CMAP. O algoritmo BC é baseado em uma formulação similar à proposta no Capítulo 2, onde a conectividade da solução é imposta através de *cutset* direcionadas (*Directed Cutsets* - DCS). Durante o BC, essa formulação é reforçada por desigualdades *multistar* generalizadas, obtidas através das propostas por Gouveia e Lopes [10] para o problema CMSTP. Nossos resultados computacionais mostram que, em média, as novas desigualdades melhoraram o limite da Relaxação Linear em 56,76% nas instâncias UD e em 48,19% nas instâncias Non-UD.

Neste capítulo também contribuímos com um novo conjunto de instâncias para CMRAP a partir de um algoritmo de geração de colunas sob a formulação (3.13), para o CMSTP. As instâncias utilizadas por Uchoa et al. [4] foram dadas como entrada para um algoritmo de geração de colunas e, em cada iteração, os custos reduzidos associados aos arcos induzem uma instância para o problema CMRAP.

Este capítulo está organizado da seguinte maneira. A Seção 4.2 apresenta uma formulação para o CMAP baseada em *cutset* direcionadas. Em seguida, uma família de desigualdades válidas é apresenta na Seção 4.3. Na Seção 4.4, um algoritmo de separação para essas desigualdades é proposto. Na seção subsequente, uma heurística primal é apresentada. Na Seção 4.6 apresentamos como foi conduzida a geração das instâncias e suas características. Por último, a Seção 4.7 apresenta os resultados do algoritmo *Branch-and-cut* proposto.

4.2 Formulação

Nesta seção, vamos apresentar uma formulação para o problema CMAP sob um grafo direcionado $G = (V, A)$ que aumenta o grafo de entrada $G_0 = (V_0, A_0)$ com a adição de um vértice raiz artificial r e arcos saindo de r. Isto é, arcos $\{(r, i) : i \in V_0\}$, todos com custo zero. Com exceção da restrição do problema da mochila, essa formulação está totalmente relacionada àquela apresentada no Capítulo 2, para o problema

MAP. Novamente, as variáveis responsáveis por selecionar os vértices e arcos da arborescência são respectivamente $\{y_i \geq 0 : i \in V_0\}$ e $\{x_{ij} \geq 0 : (i,j) \in A\}$. Usando as mesmas notações apresentadas anteriormente, temos a seguinte formulação:

$$\min \left\{ \sum_{(i,j)\in A} c_{ij}x_{ij} : (x,y) \in \mathcal{R}_0 \cap (\mathbb{R}^{|A|}, \mathbb{Z}^{|V_0|}) \right\}, \tag{4.1}$$

onde $\mathcal{R}_0$ é a região poliedral definida pela interseção das restrições

$$\sum_{(r,j)\in A} x_{rj} = 1 \tag{4.2}$$

$$\sum_{(j,i)\in A} x_{ji} = y_i, \ \forall i \in V_0 \tag{4.3}$$

$$\sum_{(j,i)\in \delta^-(S)} x_{ji} \geq y_i, \ \forall S \subseteq V_0, \forall i \in S \tag{4.4}$$

$$X(A(S)) \leq y_i, \ \forall (i,j) \in A_0, S = \{i,j\} \tag{4.5}$$

$$\sum_{i\in V_0} d_i y_i \leq C \tag{4.6}$$

$$0 \leq x_{ij} \leq 1, \ \forall (i,j) \in A \tag{4.7}$$

$$0 \leq y_i \leq 1, \ \forall i \in V_0, \tag{4.8}$$

e a sua relaxação linear correspondente é

$$\min \{ \sum_{(i,j)\in A} c_{ij}x_{ij} : (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_0 \}. \tag{4.9}$$

Note que as restrições (4.2)-(4.5) também definem a região poliedral referente à formulação (2.1) em que o MAP é definido, no Capítulo 2. Sendo assim, as soluções inteiras viáveis do poliedro da formulação (4.1) são *r-arborescências* de G. Além disso, ao considerar a restrição de capacidade (4.6) associada ao *knapsack problem*, temos que essas soluções viáveis são *r-arborescências-capacitadas*.

4.3 Desigualdade Válida: *Multistar* Generalizada

Nesta seção apresentamos uma versão generalizada da desigualdade (3.6) para o problema CMAP, denominada Multistar Generalizada (Generalized Multistar - GM). Para esse problema, a arborescência não precisa ser geradora e sendo assim o lado direito da desigualdade (3.6) deve ser reescrito em função das variáveis y.

Dessa forma, temos a seguinte desigualdade:

$$\sum_{i \in V \setminus S} \sum_{j \in S} (C - d_i) x_{ij} - \sum_{j \in S} \sum_{i \in V \setminus S} d_i x_{ji} \geq \sum_{i \in S} d_i y_i, \qquad \forall S \subset V_0 \tag{4.10}$$

Assim como as desigualdades (3.6), as desigualdades GMs são restrições que garantem a conservação de fluxo de capacidade. Nas GMs, porém, a demanda de S depende dos vértices que são selecionados para entrar na solução. Os resultados computacionais mostram que fortalecendo a formulação (4.9) com as desigualdades GMs (4.10) é possível encontrar limites duais melhores. Agora vamos mostrar que essas desigualdades são de fato válidas para a formulação apresentada.

Proposição 1. *As desigualdades GMs (4.10) são válidas para a formulação (4.1) do CMRAP.*

Demonstração. Reescreva (4.10) da seguinte maneira:

$$\sum_{i \in V \setminus S} \sum_{j \in S} (C - d_i) x_{ij} \geq \sum_{i \in S} d_i y_i + \sum_{j \in S} \sum_{i \in V \setminus S} d_i x_{ji}, \qquad \forall S \subset V_0. \tag{4.11}$$

e assuma que $(\overline{x}, \overline{y})$ seja uma solução viável inteira para a formulação (4.1). Adicionalmente, considere $\overline{G} = (\overline{V}, \overline{A})$ seu grafo suporte correspondente.

Para a prova que se segue, a GM (4.10) será derivada apenas a partir de argumentos provenientes da formulação (4.1). Para isso, alguns resultados intermediários devem ser obtidos. Ainda, assuma que um conjunto de vértices $\mathcal{S} \subseteq V_0$ seja dado e que sejam definidos os seguintes conjuntos associados a ele:

- $\mathcal{V}^-(\mathcal{S}) = \{i \in \overline{V} \setminus \mathcal{S} : (i, j) \in \overline{A}, j \in \mathcal{S}\}$
- $\mathcal{V}(\mathcal{S}) = \{j \in \mathcal{S} \cap \overline{V}\}$
- $\mathcal{V}^+(\mathcal{S}) = \{i \in \overline{V} \setminus \mathcal{S} : (j, i) \in \overline{A}, j \in \mathcal{S}\}$

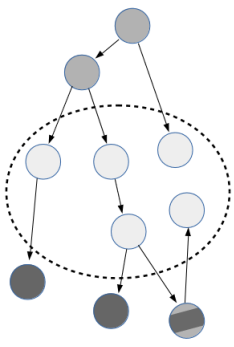

Figura 4.1: Conjutos associados a $\mathcal{S} \subseteq V_0$: $\mathcal{V}^-(\mathcal{S})$; $\mathcal{V}(\mathcal{S})$; $\mathcal{V}^+(\mathcal{S})$

Dado que (4.10) é trivialmente satisfeita quando $\mathcal{V}(\mathcal{S}) = \emptyset$, somente o caso onde $\mathcal{V}(\mathcal{S}) \neq \emptyset$ será considerado ao longo dessa prova. Das definições acima note que

$$\sum_{i \in V \setminus \mathcal{S}} \sum_{j \in \mathcal{S}} (C - d_i) x_{ij} \geq \sum_{i \in \mathcal{V}^-(\mathcal{S})} (C - d_i) \geq \sum_{i \in \mathcal{V}^-(\mathcal{S}) \setminus \mathcal{V}^+(\mathcal{S})} (C - d_i) \tag{4.12}$$

são válidas. Além disso, $\mathcal{V}^-(\mathcal{S}) \setminus \mathcal{V}^+(\mathcal{S}) \neq \emptyset$ também se aplica uma vez que $\overline{G}$ é conexo e não contém ciclos. Ainda, pela restrição (4.3) temos que

$$\sum_{j \in \mathcal{S}} \sum_{i \in V \setminus \mathcal{S}} d_i x_{ji} = \sum_{k \in \mathcal{V}^+(\mathcal{S})} d_k \tag{4.13}$$

e pela definição de $\mathcal{V}(\mathcal{S})$ temos também

$$\sum_{j \in \mathcal{S}} d_j y_j = \sum_{j \in \mathcal{V}(\mathcal{S})} d_j. \tag{4.14}$$

Considere agora a seguinte desigualdade trivial

$$\sum_{i \in \mathcal{V}^-(\mathcal{S}) \setminus \mathcal{V}^+(\mathcal{S})} (C - d_i) \geq 0. \tag{4.15}$$

Note que podemos aumentar seu lado direito para

$$\sum_{i \in \mathcal{V}^-(\mathcal{S}) \setminus \mathcal{V}^+(\mathcal{S})} (C - d_i) \geq \sum_{j \in \mathcal{V}(\mathcal{S})} d_j + \sum_{k \in \mathcal{V}^+(\mathcal{S})} d_k \tag{4.16}$$

por conta da restrição de capacidade (4.6). Além disso, substituindo o lado direito de (4.16) por suas equivalencias em (4.13) e (4.14), temos que

$$\sum_{i \in \mathcal{V}^-(\mathcal{S}) \setminus \mathcal{V}^+(\mathcal{S})} (C - d_i) \geq \sum_{j \in \mathcal{S}} d_j y_j + \sum_{i \in V \setminus \mathcal{S}} \sum_{j \in \mathcal{S}} d_i x_{ji}. \tag{4.17}$$

Combinando as desigualdades (4.17) e (4.12) pode-se derivar as desigualdades GM (4.10). Sendo assim, temos que a desigualdade GM (4.10) é válida para formulação (4.1). □

4.4 Algoritmo *Branch-and-cut* para o CMRA

Nesta seção, vamos descrever alguns detalhes do nosso algoritmo *Branch-and-cut* para o CMAP. Novamente, utilizamos o resolvedor XPRESS versão 8.4.4 e com os mesmos parâmetros utilizados no algoritmo *Branch-and-cut* do Capítulo 2. Nesse algoritmo *Branch-and-cut*, porém, além das desigualdades *cutset* direcionadas (DCS), vamos procurar por desigualdades GMs (4.10) que estejam violadas. O

algoritmo de separação para as desigualdades DCS é o mesmo apresentado na Seção 2.4.1 e o algoritmo de separação referente às GMs é apresentado na Seção 4.4.1.

Seja $(\overline{x}, \overline{y})$ uma solução ótima para

$$\min \left\{ \sum_{a \in A} c_a x_a : (x, y) \in \mathcal{R}_1 \right\}, \tag{4.18}$$

onde $\mathcal{R}_1$ é definido por (4.2)-(4.3), (4.5)-(4.8) e eventualmente por algumas desigualdades DCS e GMs separadas anteriormente. Ainda, se (2.33) corresponde a um nó não raiz da árvore de enumeração, $\mathcal{R}_1$ também pode conter desigualdades impostas pela regra de *branching*.

4.4.1 Separação *multistar* generalizada

Apesar de não serem necessárias para formular o problema, as desigualdades GMs (4.10) conseguem melhorar consideravelmente o limite inferior para o problema CMAP, como veremos nos resultados computacionais. Para adicionar esse novo conjunto de desigualdades de forma eficiente, é necessário um algoritmo de separação para encontrar, dentre as exponencialmente muitas desigualdades GMs, algumas que sejam violadas em $(\overline{x}, \overline{y})$, caso existam.

Letchford e Salazar-González [34] apresentam um algoritmo exato para fazer a separação das desigualdades *Multistar* (3.6) referentes ao problema CMSTP. Esse algoritmo associa algumas capacidades especificas para $\overline{G}' = (\overline{V}', \overline{A}')$, um grafo que aumenta o grafo suporte $\overline{G}$ associado a $(\overline{x}, \overline{y})$. Para isso, adicionam-se dois vértices artificiais *source* e *sink*, s e t, respectivamente e arcos (s, i) e (t, i) para todo $i \in \overline{V}$. O algoritmo de separação então resolve o problema de fluxo máximo (Maximum Flow Problem - MFP) sob o grafo $\overline{G}'$, onde s e t é o par *source-sink* correspondente. O algoritmo de separação que vamos propor para as GMs aplica a mesma ideia descrita acima. Aqui, porém, as capacidades associadas aos arcos de $\overline{G}'$ são diferentes das utilizadas em [34].

Seja $S \subseteq V_0$ um subconjunto de vértices de G. Adicionalmente, considere ainda que $\overline{q}_{ij}$ e $\underline{q}_{ij}$ sejam, respectivamente, limites superior e inferior para a capacidade que pode passar pelo arco (i, j). Para as desigualdades (3.6), [34] utiliza $\overline{q}_{ij} = (C - d_i)$ e $\underline{q}_{ij} = d_j$ para esses limites. Aqui vamos aplicar esses mesmos limites à desigualdade (4.10) e reescreve-la como:

$$\sum_{i \in V \setminus S} \sum_{j \in S} \overline{q}_{ij} x_{ij} - \sum_{j \in S} \sum_{i \in V \setminus S} \underline{q}_{ji} x_{ji} \geq \sum_{i \in S} d_i y_i \tag{4.19}$$

Agora, vamos reescrever (4.19) através de dois passos. Primeiro isolando o vértice raiz r e em seguida subtraindo e adicionando $\underline{q}_{ij}$ do primeiro e do segundo termo do

lado esquerdo, respectivamente:

$$\sum_{j\in S}\overline{q}_{rj}x_{rj}+\sum_{i\in V_0\setminus S}\sum_{j\in S}(\overline{q}_{ij}-\underline{q}_{ij})x_{ij}+\sum_{j\in S}\sum_{i\in V_0\setminus S}(\underline{q}_{ij}x_{ij}-\underline{q}_{ji}x_{ji})\geq\sum_{i\in S}d_iy_i \tag{4.20}$$

Em seguida, reescrevemos (4.20) de forma que o lado esquerdo tenha apenas termos positivos e o lado direito seja independente de S:

$$\begin{aligned}\sum_{j\in S}\overline{q}_{rj}x_{rj}+\sum_{i\in V_0\setminus S}\sum_{j\in S}(\overline{q}_{ij}-\underline{q}_{ij})x_{ij}+\sum_{i\in V_0}\sum_{j\in S}\underline{q}_{ij}x_{ij}+\sum_{j\in V_0\setminus S}\sum_{i\in V_0}\underline{q}_{ji}x_{ji}+\sum_{i\in V_0\setminus S}d_iy_i\\ \geq\sum_{i\in V_0}d_iy_i+\sum_{i\in V_0}\sum_{j\in V_0}\underline{q}_{ij}x_{ij}\end{aligned} \tag{4.21}$$

A separação das desigualdades GM é então conduzida resolvendo um problema de fluxo máximo (MFP) formulado sob $\overline{G}'=(\overline{V}',\overline{A}')$. As capacidades dos arcos em $\{(i,j)\in\overline{A}':i\neq r\}$ são definidos por

$$(\overline{q}_{ij}-\underline{q}_{ij})\overline{x}_{ij}$$

então atribuímos as capacidades dos arcos (s,i):

$$\overline{q}_{ri}\overline{x}_{ri}+\sum_{j\in V_0}\underline{q}_{ji}\overline{x}_{ji}$$

por último, as capacidades dos arcos (i,t)

$$d_i\overline{y}_i+\sum_{j\in V_0}\underline{q}_{ij}\overline{x}_{ij}.$$

Note que um corte mínimo $(s-t)$ de $\overline{G}'$ com valor

$$L<\sum_{i\in V_0}d_i\overline{y}_i+\sum_{i\in V_0}\sum_{j\in V_0}\underline{q}_{ij}\overline{x}_{ij} \tag{4.22}$$

identifica a desigualdade GM mais violada em $(\overline{x},\overline{y})$. Sendo assim, a desigualdade GM pode ser separada em tempo polinomial.

4.4.2 Algoritmo de planos de corte

Nesse algoritmo separamos as desigualdades *cutset* direcionadas (4.4) e as desigualdades *multistar* generalizada (4.10) proposta na Seção 4.3.

As primeiras desigualdades violadas a serem separadas são as *cutset* direcionadas (4.4), e para isso, utilizamos o mesmo algoritmo apresentado na Seção 2.4.1. Em

seguida, somente para o primeiro nó da árvore de *Branch-and-cut*, procuramos por desigualdades *multistar* generalizadas violadas. Limitamos esta separação ao primeiro nó pois testes empíricos mostraram que logo nos primeiros níveis da árvore de enumeração não existem mais desigualdades *multistar* generalizadas violadas. Para encontrar uma desigualdade *multistar* generalizada violada, primeiro construímos um grafo suporte como descrito na Seção 4.3 e então executamos o algoritmo de fluxo máximo [21] de s para t. Com o valor L associado ao corte $(s-t)$ podemos determinar se o corte representa uma desigualdade *multistar* generalizada violada fazendo a comparação (4.22).

4.5 Heurística para o CMAP

Nesta seção, vamos apresentar uma heurística que utiliza programação dinâmica (DPh - *Dynamic Programming Heuristic*) para o problema CMAP sob o grafo aumentado $G = (V, A)$ do grafo de entrada $G_0 = (V_0, A_0)$. Esse grafo aumentado G é o mesmo utilizado pela formulação (4.1).

A heurística DPh parte de uma arborescência geradora $T_{span} = (V, A_{span})$, onde $A_{span} \subseteq A$, e de uma capacidade C. Essa heurística retorna uma arborescência (não necessariamente geradora) enraizada em $r' \in V_0$ com capacidade menor ou igual a C. A raiz r' é a subraiz da solução para o CMRAP, e para obter uma solução viável para o CMRAP basta adicionar o arco (r, r') à solução encontrada pela heurística.

Quando o grafo aumentado G não é uma arborescência, primeiro computamos uma arborescência de G e então passamos essa arborescência como entrada para a heurística DPh. Em seguida, temos uma fase para tentar melhorar a solução encontrada pela DPh. Essa fase tem como objetivo encontrar a arborescência geradora de custo mínimo para o grafo direcionado induzido pelos vértices da solução encontrada em DPh. Note que a heurística DPh é responsável por escolher os vértices mais atraentes (fase de seleção) e a fase de melhoria por procurar a melhor maneira de conectar esses vértices considerando o custo do arco (r, r') que será adicionado a posteriori.

4.5.1 Fase de Seleção

Dada uma arborescência geradora $T_{span} = (V_0, A_{span})$, para cada sub-arborescência enraizada em $v \in V_0$ é calculada uma solução primal viável considerando cada capacidade $c \in \{d_v, d_v + 1, ..., C\}$. Essa solução é denotada por $T_v^c = (V_v^c, A_v^c)$, onde $V_v^c \subseteq V_0$ e $A_v^c \subseteq A_{span}$. Note que c é a capacidade disponível mas não necessariamente a utilizada pela solução. O custo e a demanda utilizada por uma solução T_v^c são respectivamente dados por $DPh(v, c) = \sum_{a \in A_v^c} c_a$ e $d(v, c) = \sum_{s \in V_v^c} d_s \leq c$.

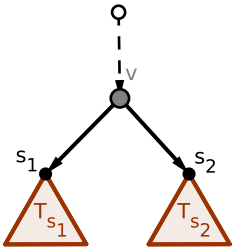

Figura 4.2: Subarborescência enraizada em v com dois filhos.

Antes de resolver o subproblema $DPh(v, c)$ para todo vértice v, é necessário resolver os subproblemas de cada filho de v em T_{span}. A Figura 4.2 ilustra a sub-arborescência T_v com os filhos s_1 e s_2. Nesse caso, resolver $DPh(v, c)$ depende de já ter resolvido $DPh(s_1, c)$ e $DPh(s_2, c)$ para todas as capacidades c.

Para todo subproblema $DPh(v, c)$ temos que escolher a combinação de sub-arborescências filhas de v que minimize o custo da solução e não viole a restrição de capacidade. Essa combinação é dada por $subDP(v, s, c')$, onde s é um filho de v e c' é a capacidade disponível. Assumindo que $DPh(v, c) = \infty$ sempre que $0 \leq c < d_v$, a formula de recursão indireta $DPh(v, c)$ é dada por

$$DPh(v, c) = \min\{0, subDP(v, s_1, c - d_v)\} \tag{4.23}$$

onde s_1 é o primeiro filho de v em T_{span}.

A equação (4.24) mostra a recursão $subDP(v, s_i, c)$. Novamente, s_i é o i-ésimo filho de v em T_{span}.

$$\begin{aligned} subDP(v, s_i, c) = \min\{&subDP(v, s_{i+1}, c), \\ &\min_{k \in \{d_{s_i}, .., c\}} \{c_{v,s_i} + DPh(s_i, k) \quad + \\ &\qquad subDP(v, s_{i+1}, c - d(s_i, k)) \\ &\qquad \} \\ \} \end{aligned} \tag{4.24}$$

Nessa recursão, avaliamos se a sub-arborescência enraizada em s_i com capacidade menor ou igual a c irá pertencer a solução da sub-arborescência $T_v^{c+d_v}$. O caso mais simples (primeiro argumento do primeiro min) é quando essa sub-arborescência não é escolhida e nesse caso basta resolver o $subDP$ para o filho seguinte sem alterar a capacidade disponível. Quando a sub-arborescência é escolhida, temos que decidir qual a melhor capacidade, ou seja, qual das sub-arborescências $\{T_{s_i}^{d_s}, T_{s_i}^{d_s+1}, ..., T_{s_i}^{c}\}$ irá levar à melhor solução. Isto é feito no segundo argumento do primeiro min. Neste caso, a capacidade disponível para avaliar os próximos filhos é subtraída de $d(s_i, k)$.

Note que $subDP(v, s, c) \leq subDP(v, s, c-1)$, uma vez que uma solução para $subDP(v, s, c-1)$ também é válida para $subDP(v, s, c)$. Sendo assim, podemos obter uma solução sub-ótima para o $subDP(v, s_i, c)$ ao reescrever a recorrência da seguinte maneira:

$$\begin{aligned} subDP(v, s_i, c) = \min\{&subDP(v, s_{i+1}, c), \\ &c_{v,s_i} + DPh(s_i, c) + subDP(v, s_{i+1}, c - d(s_i, c))\} \end{aligned} \tag{4.25}$$

Consequentemente, parte do nosso subproblema é um problema da mochila onde o peso dos itens depende da capacidade disponível, ou seja, das escolhas anteriores. Esse problema é uma generalização do problema da mochila, por isso vamos usar uma heurística gulosa para encontrar uma solução de boa qualidade. Essa heurística ordena crescentemente as sub-arborescências de v por $c_{v,s} + DPh(s, c)$. Então, seguindo esta ordenação, adicionamos $T^c_{s_i}$ à T^c_v se a demanda $d(s_i, c)$ do i-ésimo filho não violar a restrição de capacidade.

O custo da solução para o CMRAP é dado por:

$$\begin{aligned} &DPh\_Sol(T_{span}, C) = \\ &\qquad \min_{v \in V_0}\{c_{r,v} + DPh(v, C)\}. \end{aligned} \tag{4.26}$$

4.5.2 Fase de melhoria

A fase de melhoria recebe como entrada o grafo aumentado G e a solução $T_{DPh} = (V_{DPh}, A_{DPh})$, onde $V_{DPh} \subseteq V_0$ e $A_{DPh} \subseteq A_{span}$, encontrada em (4.26). Essa fase tem como objetivo escolher a melhor arborescência geradora para o grafo induzido pelos vértices de $V_{DPh} \setminus \{r\}$ ($G' = G[V_{DPh} \setminus \{r\}]$). Denotamos por $min\_arb(G', v)$ o procedimento que calcula a arborescência geradora de custo mínimo enraizada no vértice v [3] em G'. Assim sendo, essa fase é dada por:

$$\min_{v \in V_{DPh}} \{c_{r,v} + min\_arb(G', v)\}, \tag{4.27}$$

ou seja, para cada vértice v escolhido pela fase de seleção, calculamos a arborescência geradora de custo mínimo enraizada em v e adicionamos o arco (r, v) à essa solução.

4.5.3 Randomized Iterative version

Quando o grafo de entrada não é uma arborescência, escolher apenas uma arborescência geradora de G como entrada para a heurística pode não levar a soluções de boa qualidade. Com o objetivo de atenuar esse problema, desenvolvemos uma versão randomizada e iterativa da heurística DPh, chamada de $RandDPh$. Esse processo consiste em gerar aleatoriamente uma arborescência geradora de G

usando uma adaptação do algoritmo de Prim [37], onde usamos uma Lista Restrita de Candidatos (RCL - *Restricted Candidate List*) e um vértice inicial aleatório. A adaptação é feita na escolha do vértice que irá expandir a arborescência em construção, ao invés de fazer uma escolha gulosa, escolhemos um vértice da RCL.

Com isso, podemos implementar uma meta-heurística GRASP [38] para encontrar soluções viáveis. Cada iteração inicia selecionando um vértice aleatório e criando uma arborescência geradora como descrito anteriormente. Em seguida, executamos a heurística DPh para essa arborescência. No final da iteração, executamos a fase de melhoria e avaliamos se encontramos uma solução melhor que as encontradas em outras iterações.

			Limite LP		Tempo CPU (seg.)		# de *cutsets* (2.4)		# de cortes
Instâncias	C	Opt	$\mathcal{R}_0$	$\mathcal{R}_0^+$	$\mathcal{R}_0$	$\mathcal{R}_0^+$	$\mathcal{R}_0$	$\mathcal{R}_0^+$	(4.10)
tc80-1	5	-3,70	-21,11	-8,69	12,93	16,65	2055,67	721,67	52,33
tc80-1	10	-0,53	-7,67	-3,60	12,98	11,26	2663,80	792,40	33,00
tc80-1	20	-1,65	-3,27	-1,88	6,04	3,97	1688,20	906,00	39,80
tc80-2	5	-1,46	-18,25	-6,66	8,37	12,20	1468,00	641,33	48,33
tc80-2	10	-1,15	-11,50	-4,98	12,84	10,85	2611,20	904,40	47,00
tc80-2	20	-3,63	-5,83	-4,70	9,47	7,84	2704,60	902,40	47,80
tc80-3	5	-0,37	-15,21	-7,17	8,61	9,01	1743,20	465,00	40,20
tc80-3	10	-0,18	-8,52	-4,94	13,10	13,20	2959,20	382,80	24,80
tc80-3	20	-0,33	-0,37	-0,37	0,85	1,18	476,67	311,00	16,00
tc80-4	5	-0,08	-17,68	-7,31	8,39	11,22	1676,33	530,67	43,00
tc80-4	10	-1,04	-7,04	-5,47	22,08	23,21	5968,67	338,33	18,00
tc80-4	20	-0,25	-0,51	-0,48	9,87	11,10	3280,20	1394,20	50,60
tc80-5	5	-0,09	-23,98	-7,49	16,08	15,70	2509,00	658,67	47,33
tc80-5	10	-2,81	-16,18	-7,96	10,68	12,44	2250,80	829,40	33,80
tc80-5	20	-4,31	-9,03	-6,24	6,51	4,25	1648,40	768,20	40,80
tc120-1	5	-7,02	-11,60	-7,54	31,35	11,42	3109,40	668,40	29,80
tc120-1	10	-6,07	-11,63	-7,20	54,54	29,14	5222,20	1367,00	66,20
tc120-1	20	-3,12	-6,95	-4,42	73,78	50,06	7119,20	1724,00	66,80
tc160-1	5	-9,21	-11,64	-10,20	48,47	57,82	3180,20	418,60	26,80
tc160-1	10	-4,36	-8,26	-5,52	132,28	132,17	7231,60	1963,40	73,00
tc160-1	20	-6,42	-9,67	-7,59	196,53	152,51	9600,00	2340,20	57,20
Média Total		-2,96	-10,24	-5,70	4,18	5,26	3521,76	954,16	43,69

Tabela 4.1: Relaxação Linear: médias dos limites do nó raiz (instâncias do grupo **tc**).

4.6 Instâncias

As instâncias para o problema CMAP foram geradas usando um algoritmo de geração de colunas sob a formulação (3.13) para o CMSTP. Em cada iteração do algoritmo, uma instância para o CMAP pode ser gerada associando a cada arco (i, j) do grafo direcionado $G = (V, A)$ seu respectivo custo reduzido calculado por

Instâncias	C	Opt	Limite LP		Tempo CPU (seg.)		# de *cutsets* (2.4)		# de cortes
			$\mathcal{R}_0$	$\mathcal{R}_0^+$	$\mathcal{R}_0$	$\mathcal{R}_0^+$	$\mathcal{R}_0$	$\mathcal{R}_0^+$	(4.10)
te80-1	5	-1,88	-91,94	-15,08	47,85	39,91	3424,40	1585,60	169,40
te80-1	10	-5,29	-71,49	-24,93	25,50	26,69	3569,80	1211,00	92,40
te80-1	20	-18,77	-53,76	-30,38	17,11	10,82	3706,40	1016,40	71,40
te80-2	5	-1,83	-85,24	-16,77	44,55	55,74	3141,33	1336,67	150,67
te80-2	10	-5,03	-63,71	-29,00	26,08	48,56	3725,20	1261,40	68,00
te80-2	20	-26,53	-61,82	-36,89	16,43	12,08	4084,80	902,00	63,00
te80-3	5	-1,08	-78,41	-14,82	57,64	52,15	4029,40	1451,00	192,20
te80-3	10	-7,56	-72,43	-27,55	25,63	32,55	3780,80	1514,00	95,20
te80-3	20	-29,32	-63,06	-39,42	13,28	8,33	3437,00	878,20	55,40
te80-4	5	-0,58	-82,74	-17,93	59,54	50,14	4115,33	1407,67	135,33
te80-4	10	-9,67	-82,08	-28,92	38,78	39,90	5155,20	1534,00	81,40
te80-4	20	-41,14	-70,39	-48,18	12,23	9,08	3313,20	983,20	68,80
te80-5	5	-2,46	-86,20	-17,47	47,57	41,83	3626,20	1386,20	137,00
te80-5	10	-5,75	-69,78	-28,01	33,95	40,21	4694,60	1345,00	75,20
te80-5	20	-24,95	-49,01	-30,97	13,08	7,51	3284,20	923,60	62,60
te120-1	5	-12,21	-25,69	-12,56	145,73	13,88	6396,40	1759,20	55,00
te120-1	10	-6,37	-14,46	-7,54	105,52	50,83	7817,80	2339,20	90,00
te120-1	20	-9,73	-13,15	-10,17	40,38	18,71	5227,60	1800,00	59,20
te160-1	5	-6,89	-11,45	-7,14	106,39	34,37	4925,80	1375,80	44,00
te160-1	10	-8,44	-11,04	-9,16	114,10	188,03	6124,80	1819,20	41,40
te160-1	20	-17,01	-20,12	-17,64	105,94	103,53	6469,00	2445,20	61,20
Total Average		-11,96	-54,99	-22,61	7,39	8,80	4512,21	1444,40	86,85

Tabela 4.2: Relaxação Linear: médias dos limites do nó raiz (instâncias do grupo **te**).

(3.23).

As instâncias do CMSTP são identificadas por **tc**N**r**I**-**C, **te**N**r**I**-**C e **cm**N**r**I**-**C, onde N representa o número de vértices, I um contador e C a capacidade máxima. Por exemplo, **cm50r1-200** é a primeira instância de uma série de 5 instâncias com 50 vértices e capacidade máxima 200. Os grupos **tc** e **te** representam instâncias com demandas unitárias e o grupo **cm** instâncias com demandas quaisquer. Para os grupos **tc** e **te** temos grafos com 41, 81, 121 or 161 vértices e capacidades máximas 5, 10 ou 20. Para o grupo **cm** temos grafos com 50, 100 ou 200 vértices e capacidades máximas 200, 400 ou 800.

Para cada instância do CMSTP, escolhemos 5 iterações do algoritmo de geração de colunas para criar as instâncias do problema CMAP. Essas são separadas em 3 grupos: (i) duas iterações em que foi possível encontrar uma solução de custo negativo usando apenas a heurística apresentada na Seção (4.5); (ii) duas iterações em que a heurística não encontrou uma solução de custo negativo e foi necessário executar o *branch-and-cut*; (iii) a iteração onde o custo da solução ótima é não negativa. Quando o grupo (iii) é vazio, ou seja, o algoritmo parou por exceder o tempo limite, criamos uma instância com essa última iteração.

Note que para os grupos (i) e (ii) é muito provável que mais de duas iterações sejam candidatas a serem selecionadas. Nesse caso, para o grupo (i) selecionamos a iteração em que a heurística foi mais lenta e a iteração em que o valor da função objetivo foi mais próximo de zero. Para o grupo (ii) selecionamos a iteração em que o *branch-and-cut* precisou de mais tempo para provar otimalidade e a iteração com maior GAP de dualidade. Se coincidir de escolher a mesma iteração no grupo (i), a com segundo maior custo é selecionada. Para o grupo (ii), a iteração com segundo maior GAP é selecionada.

Seguindo essa mesma linha de identificação das instâncias do CMSTP, as instâncias para o problema CMAP têm a seguinte identificação: tc$N$rI-C-J, te$N$rI-C-J e cm$N$rI-C-J, onde N representa o número de vértices, I um contador, C a capacidade que foi utilizada durante o algoritmo de geração de colunas para criar a instância e J um segundo nível de contador. As configurações de vértices e capacidades máximas continuam as mesmas. Por exemplo, ao resolver a geração de colunas do CMSTP para a instância *cm50r1* com capacidade máxima $C = 200$, criamos as instâncias para o problema CMAP *cm50r1-200-1*, *cm50r1-200-2*, *cm50r1-200-3*, *cm50r1-200-4* e *cm50r1-200-5*.

4.7 Resultados computacionais

Os códigos-fonte deste capítulo foram implementados em *C++11* e compilados com *g++ 5.4.0* em modo *RELEASE*. Os experimentos foram executados em um computador Intel core i7-4770S, a 3.10 GHz e com 12 GB de RAM. Apenas um processador foi utilizado e não foi permitido o uso *multi-threading*. Para cada instância, foi estabelecido um tempo limite de execução de 2h.

O pacote de otimização XPRESS, versão 8.4.4, foi utilizado para resolver os problemas de programação linear e gerir a árvore de enumeração do *Branch-and-cut*. Assim como em todos os outros resultados computacionais desta tese, a geração de plano cortes automático, testes pré-processamento e heurísticas primais são mantidas desligadas.

Os primeiros resultados desta seção focam em mostrar o benefício do uso das desigualdades (4.10). Em particular, eles mostram a melhora que essas desigualdades trazem ao limite inferior de $\mathcal{R}_0$. Portanto, vamos denotar por $\mathcal{R}_0^+$ a formulação $\mathcal{R}_0$ fortalecida com essas novas desigualdades.

As Tabelas 4.1, 4.2 e 4.3 apresentam, respectivamente, a média dos resultados computacionais para as instâncias do grupo *tc*, *te* e *cm*. Cada linha dessas tabelas representa o resultado médio de um subconjunto de instâncias. Por exemplo, o subconjunto *tc80-1* com capacidade $C = 10$, representa o resultado médio das instâncias *tc80-1-10-1*, *tc80-1-10-2*, *tc80-1-10-3*, *tc80-1-10-4* e *tc80-1-10-5*.

			Limite LP		Tempo CPU (seg.)		# de *cutsets* (2.4)		# de cortes
Instâncias	C	MLS	$\mathcal{R}_0$	$\mathcal{R}_0^+$	$\mathcal{R}_0$	$\mathcal{R}_0^+$	$\mathcal{R}_0$	$\mathcal{R}_0^+$	(4.10)
cm50-1	200	-1,84	-47,67	-11,59	0,29	0,74	539,60	409,60	64,00
cm50-1	400	-3,23	-41,46	-13,97	0,28	0,37	693,20	451,80	33,40
cm50-1	800	-6,76	-15,18	-10,78	0,18	0,22	254,80	247,60	26,00
cm50-2	200	-2,09	-41,36	-9,87	0,25	0,53	497,00	389,80	49,60
cm50-2	400	-0,80	-31,99	-10,31	0,21	0,32	540,80	338,80	33,40
cm50-2	800	-5,10	-14,08	-8,72	0,17	0,30	428,60	438,20	34,40
cm50-3	200	-1,37	-37,83	-13,55	0,18	0,28	418,40	332,80	23,20
cm50-3	400	-5,04	-26,94	-14,16	0,24	0,24	611,60	346,00	24,80
cm50-3	800	-8,47	-12,01	-10,70	0,19	0,20	331,00	207,40	25,20
cm50-4	200	-1,87	-22,98	-13,34	0,09	0,10	215,67	134,33	12,67
cm50-4	400	-1,61	-19,27	-10,42	0,20	0,20	545,60	302,40	20,40
cm50-4	800	-3,37	-9,24	-6,65	0,30	0,36	464,80	409,20	42,60
cm50-5	200	-2,35	-29,07	-11,19	0,09	0,22	147,20	251,60	21,60
cm50-5	400	-2,01	-27,84	-11,57	0,14	0,31	322,20	371,20	31,80
cm50-5	800	-7,12	-22,07	-15,42	0,17	0,25	298,80	274,60	30,60
cm100-1	200	-4,61	-12,71	-8,02	1,79	3,71	661,60	488,60	58,00
cm100-1	400	-2,40	-5,10	-3,48	0,91	1,98	405,40	442,00	38,60
cm100-1	800	-3,37	-4,28	-3,73	0,87	1,16	447,20	413,40	21,60
cm100-2	200	-4,73	-10,73	-7,39	0,88	4,74	363,80	505,80	72,00
cm100-2	400	-2,46	-4,28	-3,46	0,31	0,87	159,20	252,20	19,00
cm100-2	800	-3,66	-4,11	-4,01	0,59	0,67	275,80	244,40	14,20
cm100-3	200	-2,42	-8,56	-7,68	0,33	0,30	139,80	81,40	6,20
cm100-3	400	-2,46	-4,19	-3,33	0,65	1,23	315,40	343,80	27,20
cm100-3	800	-3,25	-4,11	-3,54	1,22	1,34	875,60	606,40	23,80
cm100-4	200	-2,95	-10,24	-7,96	1,13	0,96	545,00	274,80	19,00
cm100-4	400	-0,92	-3,51	-2,16	0,66	3,01	392,00	634,60	54,00
cm100-4	800	-2,89	-3,46	-3,09	0,77	1,14	456,00	522,60	19,60
cm100-5	200	-4,82	-11,03	-9,36	0,70	0,79	272,80	169,60	15,40
cm100-5	400	-1,48	-4,28	-2,53	2,50	3,86	1467,00	1125,6	49,00
cm100-5	800	-2,57	-3,85	-2,99	2,50	1,89	1698,00	828,00	30,60
cm200-1	200	-2,93	-8,13	-6,02	31,74	40,41	1269,60	1384,4	64,80
cm200-1	400	-3,75	-5,40	-3,95	50,48	97,05	2770,60	2727,2	89,20
cm200-1	800	-5,93	-6,43	-6,30	25,40	27,48	1499,40	1136,8	38,40
cm200-2	200	-6,21	-9,10	-8,44	1,77	3,25	143,00	182,20	10,40
cm200-2	400	-3,91	-5,62	-4,50	103,42	58,02	4484,00	2377,2	56,80
Total Average		-3,47	-15,00	-7.77	6,69	7,47	718,62	566,26	34,58

Tabela 4.3: Relaxação Linear: médias dos limites do nó raiz das instâncias Non-UD.

A primeira e a segunda colunas das Tabelas 4.1, 4.2 e 4.3 indicam, respectivamente, o nome da instância e a capacidade C correspondente. A próxima coluna representa o melhor limite superior encontrado (MLS). Para as Tabelas 4.1 and 4.2, essa coluna representa os valores ótimos das instâncias. As duas colunas subsequentes apresentam respectivamente o limite inferior médio das formulações $\mathcal{R}_0$ e $\mathcal{R}_0^+$. As médias dos tempos de CPU para encontrar o limite inferior encontrado por cada uma das formulações são reportados nas duas próximas colunas. O número médio de desigualdades *cutsets* separadas na raiz da árvore de *Branch-and-cut* de

		Tempo CPU (s)		Fator Speed-up		# de Nos		Tempo Menor	
Instâncias	C	BC	BC^+	BC	BC^+	BC	BC^+	BC	BC^+
tc80-1	5	12,93	16,65	1,47	1,85	156,33	255,00	1	2
tc80-1	10	12,98	11,26	1,00	1,21	229,40	215,00	0	5
tc80-1	20	6,04	3,97	1,00	1,57	34,20	20,80	0	5
tc80-2	5	8,37	12,20	1,54	1,04	141,67	195,00	2	1
tc80-2	10	12,84	10,85	1,02	1,32	119,80	127,80	1	4
tc80-2	20	9,47	7,84	1,15	1,63	64,60	36,40	3	2
tc80-3	5	8,61	9,01	1,20	1,14	240,60	184,20	2	3
tc80-3	10	13,10	13,20	1,09	1,07	360,20	367,00	3	2
tc80-3	20	0,85	1,18	1,35	1,00	4,33	9,00	3	0
tc80-4	5	8,39	11,22	1,35	1,00	195,00	211,00	3	0
tc80-4	10	22,08	23,21	1,21	1,01	300,33	347,00	2	1
tc80-4	20	9,87	11,10	1,31	1,21	33,00	30,60	3	2
tc80-5	5	16,08	15,70	1,03	1,07	245,00	247,00	2	1
tc80-5	10	10,68	12,44	1,18	1,11	194,60	243,00	3	2
tc80-5	20	6,51	4,25	1,00	1,55	44,20	25,40	0	5
tc120-1	5	31,35	11,42	1,08	6,37	78,20	28,20	1	4
tc120-1	10	54,54	29,14	1,00	2,38	97,80	57,80	0	5
tc120-1	20	73,78	50,06	1,02	1,56	103,00	72,60	1	4
tc160-1	5	48,47	57,82	1,29	1,03	84,60	74,60	4	1
tc160-1	10	132,28	132,17	1,23	1,31	85,40	67,00	2	3
tc160-1	20	196,53	152,51	1,00	1,31	106,60	88,60	0	5
Total Average		35,93	30,98	1,15	1,61	134,51	128,89	36	57

Tabela 4.4: Comparação do tempo de CPU entre os algoritmos *Branch-and-cut* para as instâncias UD do grupo tc.

cada algoritmo são apresentados das duas colunas que se seguem. Por último, a última coluna informa a média do número de cortes (4.10) adicionados em $\mathcal{R}_0^+$ para resolver a relaxação linear.

A partir dos resultados das Tabelas 4.1, 4.2 e 4.3, podemos observar que as desigualdades *multistar* generalizadas induzem uma melhora significativa dos limites inferiores de $\mathcal{R}_0$, quando aplicadas às instâncias UD e Non-UD. Ainda, além dos limites inferiores de $\mathcal{R}_0^+$ serem muito mais apertados que os limites correspondentes em $\mathcal{R}_0$, eles ainda implicam em soluções inteiras ótimas para 9 instâncias UD. Além disso, podemos observar que, quando as desigualdades *multistar* generalizadas (4.10) são separadas, o número de *cutsets* necessárias para encontrar o limite inferior diminui em 70% para as instância UD e em 21% para as instâncias Non-UD.

O próximo conjunto de resultados compara duas versões do algoritmo *Branch-and-cut* proposto neste trabalho, para o CMAP. A primeira é implementada sob a formulação (4.1), denominada BC e a segunda versão é implementada sob a formulação (4.1) reforçada com as desigualdades *multistar generalizadas* (4.10) apresentadas na Seção 4.4.1, denominada BC^+.

As Tabelas 4.4, 4.5, 4.6, 4.7 e 4.8 comparam o tempo de CPU dos dois algoritmos.

		Tempo CPU (s)		Fator Speed-up		# de Nos		Tempo Menor	
Instâncias	C	BC	BC^+	BC	BC^+	BC	BC^+	BC	BC^+
te80-1	5	47,85	39,91	1,00	1,21	313,00	275,40	0	5
te80-1	10	25,50	26,69	1,11	1,06	92,20	81,40	4	1
te80-1	20	17,11	10,82	1,00	1,62	53,40	33,00	0	5
te80-2	5	44,55	55,74	1,25	1,00	354,33	519,00	2	1
te80-2	10	26,08	48,56	1,80	1,00	100,20	222,60	5	0
te80-2	20	16,43	12,08	1,02	1,54	55,00	35,80	1	4
te80-3	5	57,64	52,15	1,17	1,31	404,60	344,20	2	3
te80-3	10	25,63	32,55	1,28	1,00	111,40	125,40	5	0
te80-3	20	13,28	8,33	1,00	1,59	52,20	27,00	0	5
te80-4	5	59,54	50,14	1,03	1,25	347,00	343,00	1	2
te80-4	10	38,78	39,90	1,22	1,17	144,20	119,00	3	2
te80-4	20	12,23	9,08	1,03	1,43	52,60	26,60	1	4
te80-5	5	47,57	41,83	1,09	1,37	304,60	320,20	2	3
te80-5	10	33,95	40,21	1,31	1,06	102,60	95,40	4	1
te80-5	20	13,08	7,51	1,01	1,89	27,40	19,80	1	4
te120-1	5	145,73	13,88	1,00	10,89	137,40	3,80	0	5
te120-1	10	105,52	50,83	1,00	2,15	133,00	37,40	0	5
te120-1	20	40,38	18,71	1,00	2,33	35,00	10,20	0	5
te160-1	5	106,39	34,37	1,00	4,86	88,60	21,80	0	5
te160-1	10	114,10	188,03	1,43	1,79	69,00	58,60	2	3
te160-1	20	105,94	103,53	1,19	1,64	37,40	35,00	2	3
Total Average		52,26	41,71	1,14	2,09	135,38	119,30	35	66

Tabela 4.5: Comparação do tempo de CPU entre os algoritmos *Branch-and-cut* para as instâncias UD grupo te.

As primeiras três tabelas apresentam detalhes dos resultados computacionais médios para as instâncias do grupo *tc*, *te* e *cm*. Em seguida, para ressaltar mais tendências gerais, Tabelas 4.7 e 4.8 reportam resultados agregados médios e máximos. A primeira consolida resultados para as instâncias UD e a segunda para instâncias Non-UD.

A primeira coluna das Tabelas 4.4, 4.5 e 4.6 identifica as instâncias de teste e a segunda coluna apresenta sua capacidade C correspondente. Tempo médio de CPU, em segundos, é então reportado: uma coluna para o algoritmo BC e outra para o algoritmo BC^+. Quando um algoritmo *Branch-and-cut* não consegue provar otimalidade em 2h para uma instância, sua contribuição para o tempo médio é de 10.000,00 segundos. Nas duas colunas seguintes, é apresentado o *speed up* médio de cada algoritmo. Para cada instância de teste, assim como no Capítulo 2, seu *speed-up* corresponde à razão entre o tempo de CPU do algoritmo mais lento e o tempo de CPU necessário para o algoritmo resolver essa mesma instância. Sendo assim, *speed-up* é sempre igual a 1,00 para o algoritmo mais lento e maior que 1,00 para os algoritmos mais rápidos que ele. As próximas duas colunas apresentam o número médio de nós explorados por cada algoritmo. Essas colunas são seguidas

		Tempo CPU (s)		Fator Speed-up		# de Nos		Tempo Menor		# Opt	
Inst.	C	BC	BC^+	BC	BC^+	BC	BC^+	BC	BC^+	BC	BC^+
cm50-1	200	26,69	32,30	1,19	1,03	2675,40	2606,20	2	3	5	5
cm50-1	400	8,60	10,18	1,22	1,00	1190,60	1299,40	5	0	5	5
cm50-1	800	1,91	1,23	1,00	1,57	227,80	171,60	0	5	5	5
cm50-2	200	16,01	22,76	1,58	1,00	2130,20	2455,00	5	0	5	5
cm50-2	400	16,90	17,01	1,06	1,04	2747,40	3021,20	3	2	5	5
cm50-2	800	2,22	2,31	1,18	1,16	363,00	288,60	2	3	5	5
cm50-3	200	20,08	27,87	1,44	1,00	2629,80	2907,00	5	0	5	5
cm50-3	800	1,30	1,37	1,20	1,16	145,00	200,20	2	3	5	5
cm50-3	400	8,48	8,55	1,15	1,12	1324,20	1426,00	3	2	5	5
cm50-4	200	7,85	10,73	1,49	1,00	1456,33	1688,33	3	0	3	3
cm50-4	400	6,57	6,77	1,18	1,12	999,00	1133,40	2	3	5	5
cm50-4	800	1,81	1,88	1,12	1,11	280,60	274,20	3	2	5	5
cm50-5	200	13,10	19,39	1,55	1,00	2732,60	3432,20	5	0	5	5
cm50-5	400	6,37	8,57	1,35	1,00	1337,80	1432,00	5	0	5	5
cm50-5	800	2,10	2,59	1,41	1,11	303,00	365,00	4	1	5	5
cm100-1	200	227,33	349,27	1,49	1,02	2844,60	3621,00	3	2	5	5
cm100-1	400	56,90	47,67	1,16	1,87	1218,20	920,80	2	3	5	5
cm100-1	800	18,36	17,59	1,30	1,30	380,80	357,40	2	3	5	5
cm100-2	200	184,31	245,23	1,38	1,14	2796,60	2172,20	4	1	5	5
cm100-2	400	30,67	47,92	1,79	1,30	861,80	1184,60	4	1	5	5
cm100-2	800	18,03	20,95	1,32	1,06	764,20	1024,60	3	2	5	5
cm100-3	200	192,11	228,72	1,18	1,00	3748,40	4264,00	5	0	5	5
cm100-3	400	32,77	35,41	1,26	1,15	903,00	886,20	3	2	5	5
cm100-3	800	23,41	22,35	1,15	1,59	527,20	434,20	1	4	5	5
cm100-4	200	279,40	329,21	1,48	1,08	3944,60	3259,00	4	1	5	5
cm100-4	400	49,46	77,07	1,76	1,00	855,00	1351,40	5	0	5	5
cm100-4	800	24,43	27,09	1,33	1,43	428,40	516,20	4	1	5	5
cm100-5	200	161,59	193,00	1,19	1,08	2499,80	2160,60	3	2	5	5
cm100-5	400	125,84	126,99	1,14	1,03	1651,40	1660,40	2	3	5	5
cm100-5	800	42,44	25,67	1,13	1,86	674,20	552,20	1	4	5	5
cm200-1	200	8030,76	8301,85	1,12	1,33	3530,20	2461,20	1	1	2	1
cm200-1	400	2464,02	2960,94	1,75	1,95	1652,20	1456,60	4	1	5	5
cm200-1	800	1349,39	1404,73	1,21	1,15	1364,80	1174,00	3	2	5	5
cm200-2	200	4414,79	4818,24	2,06	1,69	3586,40	2609,20	2	3	4	4
cm200-2	400	5995,88	4871,24	1,17	1,52	1491,80	1434,00	1	4	3	4
Média Tot.		689,56	702,90	1,33	1,23	1609,36	1604,76	106	64	167	167

Tabela 4.6: Comparação do tempo de CPU entre os algoritmos *Branch-and-cut* para as instâncias non-UD.

por outras duas, intituladas *Tempo Menor*: uma coluna com o número de vezes que BC foi mais rápido que BC^+ e outra com o número de vezes que BC^+ foi mais rápido que BC. Na Tabela 4.6, temos duas colunas adicionais, intituladas *# Opt*, que apresentam, respectivamente, o número de vezes em que BC e BC^+ provou ter encontrado a solução ótima.

Como podemos observar nas Tabelas 4.4, 4.5 e 4.7, para a maioria das comparações realizadas, BC^+ é mais rápido que BC. Mais especificamente, BC^+ foi

# de Instâncias em que:	
$\mathcal{R}_0^+$ foi mais lento que $\mathcal{R}_0$	71
$\mathcal{R}_0^+$ foi mais rápido que $\mathcal{R}_0$	123
Empates	0
Média fator speed up para $\mathcal{R}_0^+$	1,86
Maior fator speed up para $\mathcal{R}_0^+$	18,45
Média fator speed up para $\mathcal{R}_0$	1,14
Maior fator speed up para $\mathcal{R}_0$	3,09

Tabela 4.7: Comparação do tempo de CPU entre os algoritmos *Branch-and-cut* para as instâncias UD: resultados agregados.

# of Instances for which:	
$\mathcal{R}_0^+$ foi mais lento que $\mathcal{R}_0$	106
$\mathcal{R}_0^+$ foi mais rápido que $\mathcal{R}_0$	64
Empates	3
Média fator speed up para $\mathcal{R}_0^+$	1,23
Maior fator speed up para $\mathcal{R}_0^+$	5,73
Média fator speed up para $\mathcal{R}_0$	1,33
Maior fator speed up para $\mathcal{R}_0$	4,20

Tabela 4.8: Comparação do tempo de CPU entre os algoritmos *Branch-and-cut* para as instâncias non-UD: resultados agregados.

mais rápido em 123 instâncias, enquanto que em apenas 71 instâncias o algoritmo BC foi mais rápido. Além disso, o *speed-up* médio e máximo correspondente ao BC^+ é respectivamente 1,86 e 18,45. Em contraste a este cenário, para o BC, estes valores são respectivamente 1,14 e 3,09. Para cada algoritmo, desconsiderando aquelas instâncias em que o algoritmo foi mais lento, o *speed-up* médio para o BC^+ é de 2,36 enquanto que para o BC é de 1,39. Vale lembrar que o limite inferior de BC é baseado em $\mathcal{R}_0$, e a razão para tal diferença de performance deve ser atribuída ao uso das desigualdades (4.10).

Para as instâncias Non-UD, as Tabelas 4.6 e 4.8 mostram que o algoritmo BC^+ não conseguiu se beneficiar do limite inferior mais apertado obtido pelas desigualdades (4.10). Mais especificamente, em apenas 64 instâncias o BC^+ conseguiu ser mais rápido que o BC, enquanto que o BC foi mais rápido em 106 instâncias. Para cada algoritmo, desconsiderando aquelas instâncias em que o algoritmo foi mais lento, o *speed-up* médio para o BC^+ é de 1,65 enquanto para o BC é de 1,54.

4.7.1 Análise das Instâncias

Nesta seção conduzimos análises da dificuldade das instâncias sugeridas para o CMAP. Na primeira análise é feita uma comparação entre as dificuldades dos casos

(i), (ii) e (iii)[1] e para isto as instâncias do CMAP são separadas em dois novos grupos: *GC-Tempo* e *GC-Convergiu*. O grupo *GC-Tempo* contém todas as instâncias do CMAP cujos respectivos algoritmos de Geração de Colunas (GC) pararam por terem excedido o limite de tempo, e o grupo *GC-Convergiu* contém as instâncias cujos seus respectivos algoritmos de Geração de Colunas encontraram uma solução ótima de custo não-negativo no último subproblema. Por exemplo, ao executar o algoritmo de Geração de Colunas para a instância do CMSTP *tc80-1* com capacidade *C=20*, o custo de uma solução ótima para o subproblema associado à última iteração é negativo, e por isso as instâncias para o CMAP *tc80-1-20-1*, *tc80-1-20-2*, *tc80-1-20-3*, *tc80-1-20-4* e *tc80-1-20-5* pertencem ao grupo *GC-Tempo*. Por outro lado, para a instância *tc80-1* com capacidade *C=10*, esse custo é não-negativo, e por isso as instâncias para o CMAP *tc80-1-10-1*, *tc80-1-10-2*, *tc80-1-10-3*, *tc80-1-10-4* e *tc80-1-10-5* pertencem ao grupo *GC-Convergiu*.

Nas Tabelas 4.9 e 4.10, a primeira coluna identifica os dois novos grupos. As três colunas subsequentes, denominadas *GAP %*, *# de Nós* e *Tempo CPU*, têm associado valores médios para os casos (i), (ii) e (iii), respectivamente.

Podemos observar nas Tabelas 4.9 e 4.10 que as instâncias do caso (i) são, em média, mais fáceis que as instâncias dos casos (ii) e (iii). Essa conclusão decorre do fato de que os maiores GAPs de dualidades se concentram nas instâncias do caso (ii) e (iii). Além disso, as colunas subsequentes dão suporte a essa tendência, mostrando que o número de nós e o tempo de CPU também são maiores nesses casos. Note que, para o grupo *CG-Timeout*, os valores médios para os casos (ii) e (iii) são próximos uns dos outros e não existe uma relação entre suas dificuldades.

	GAP %			# de Nós			Tempo CPU (seg.)		
Grupo	i	ii	iii	i	ii	iii	i	ii	iii
GC-Tempo	45,23	60,88	59,38	64,19	82,00	88,63	45,81	54,08	62,27
GC-Convergiu	89,50	97,25	100,00	193,00	293,38	381,50	23,15	26,22	33,31

Tabela 4.9: Dificuldade instâncias UD.

	GAP %			# de Nós			Tempo CPU (seg.)		
Grupo	i	ii	iii	i	ii	iii	i	ii	iii
GC-Tempo	47,81	51,99	64,90	1088,73	1753,85	1667,46	729,14	883,47	693,98
GC-Convergiu	88,28	95,76	100,00	1285,00	2171,25	3447,75	9,36	15,01	22,62

Tabela 4.10: Dificuldade instâncias Non-UD.

Nas Figuras 4.3 e 4.4 são realizadas análises da dificuldade de cada subproblema associado as iterações do algoritmo de Geração de Colunas. Para essa análise, todas as iterações do algoritmo de Geração de Colunas são resolvidas pelo algoritmo *BC* e

[1]ver definição desses casos na Seção 4.6

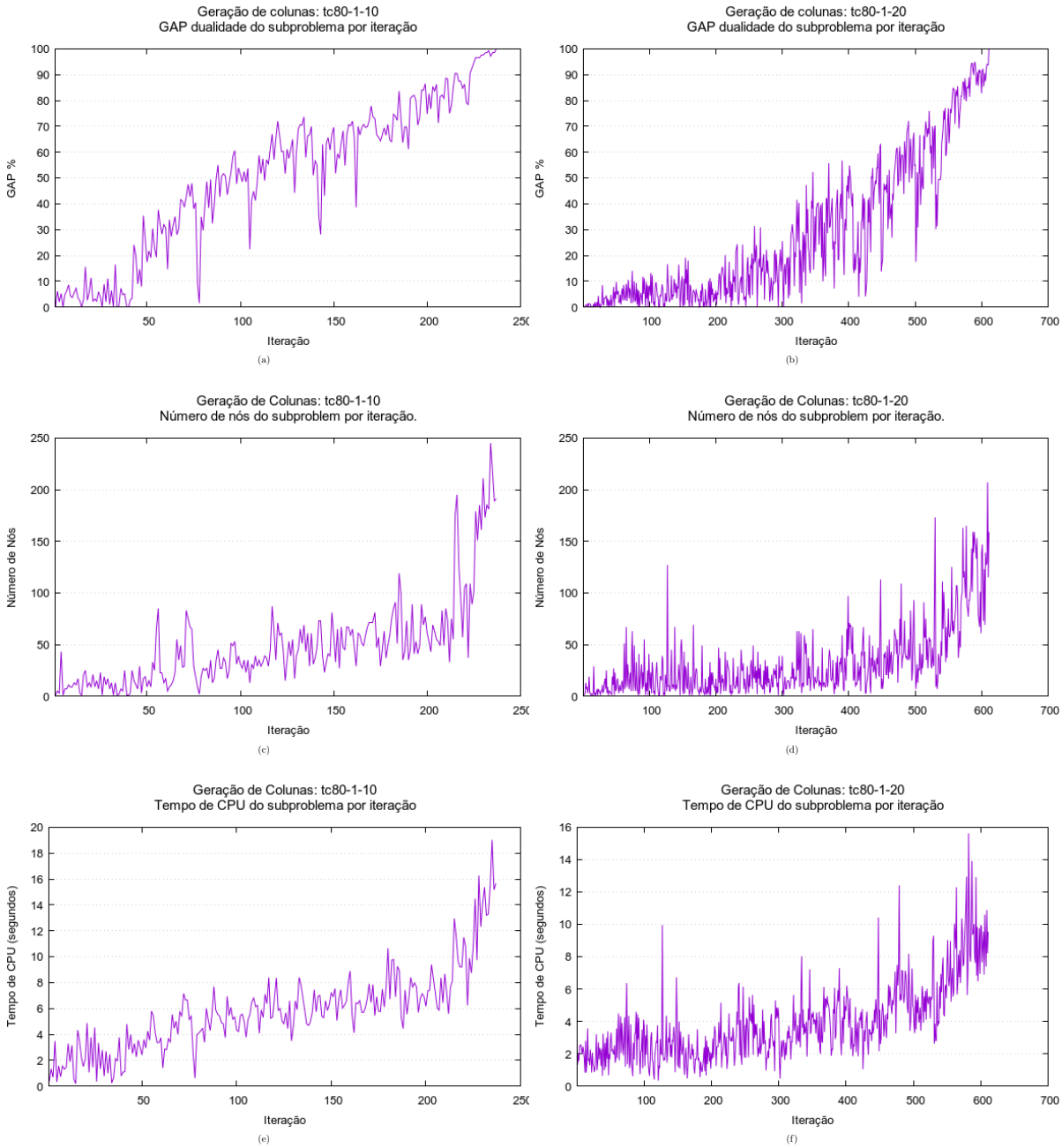

Figura 4.3: Dificuldade dos subproblemas associados à duas instâncias UD: as Figuras a, c e e estão associadas à instância *tc80-1-10* e as Figuras b, d e f estão associadas à instância *tc80-1-20*.

nenhum corte é adicionado ao problema mestre. Quatro instâncias do CMSTP, duas UD e duas non-UD, foram selecionadas para gerar os dados das figuras: *tc80-1-10*, *tc80-1-20*, *cm50r1-200* e *cm50r1-800*. O GAP de dualidade, o número de nós e tempo de CPU por iteração da Geração de colunas são ilustrados na Figura 4.3 para as instâncias UD e na Figura 4.4 para as instâncias non-UD.

Podemos observar nessas figuras que os subproblemas da Geração de colunas mais próximos das últimas iterações tendem a ser os mais difíceis. Note que, para

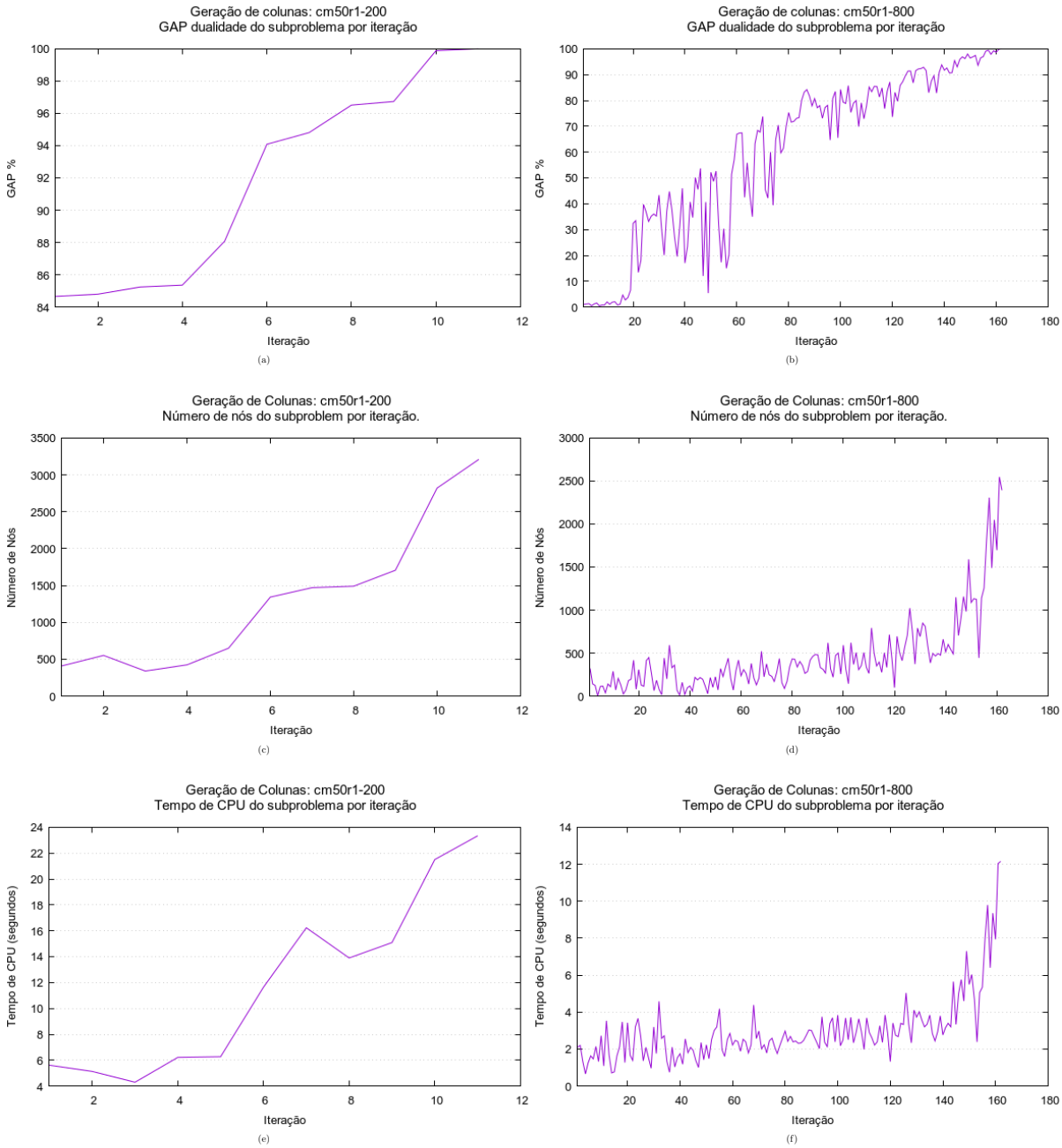

Figura 4.4: Dificuldade dos subproblemas associados à duas instâncias non-UD: as Figuras a, c e e estão associadas à instância *cm50r1-200* e as Figuras b, d e f estão associadas à instância *cm50r1-800*.

todas as quatro instâncias, o GAP de dualidade, o número de nós e o tempo de CPU crescem quando o valor da solução ótima do subproblema aproxima-se de zero.

4.7.2 Limites da geração de colunas para o CMSTP

Nesta seção vamos apresentar resultados do nosso algoritmo de geração de colunas implementado sob a formulação (3.1). Para esses resultados, porém, as desigualdades de Capacidade (3.3) não são consideradas no problema mestre. Essa implementação

		Limites					
Instâncias	C	L2	$\gamma_{bc} = 1$	$\gamma_{bc} = 2$	$\gamma_{bc} = 3$	$\gamma_{bc} = 4$	$\gamma_{bc} = 5$
te80-1	5	2524,42	2528,71	2528,71	2529,58	**2529,71**	2529,71
te80-2		2516,37	2525,30	2525,90	2526,70	**2527,80**	2527,80
te80-3		2575,24	2589,79	2589,79	2591,36	2591,36	2591,47
te80-4		2529,03	2538,17	2540,29	2540,29	2540,29	2540,29
te80-5		2450,25	2458,28	2459,38	2459,63	2460,40	**2461,40**
Tempo médio		1,77	56,98	111,51	164,28	235,84	272,70
te80-1	10	1618,22	1643,57	1646,87	1646,87	1646,87	1646,87
te80-2		1589,57	1620,01	1620,05	1620,05	1620,40	1620,40
te80-3		1643,41	1672,43	1678,36	1678,36	1678,36	1678,36
te80-4		1601,98	1618,18	1620,44	1623,96	1624,54	1625,42
te80-5		1577,63	1594,55	1594,55	1594,55	1594,55	1594,55
Tempo médio		7,30	46,07	73,77	98,14	126,88	158,24
te80-1	20	1214,99	1248,83	1252,35	1253,16	1253,16	1253,16
te80-2		1172,97	1206,90	1214,59	1214,59	1214,59	1214,59
te80-3		1218,18	1258,33	1261,03	1261,03	1261,03	1261,03
te80-4		1196,79	1260,95	1261,58	1261,58	1261,58	1264,55
te80-5		1178,67	1029,95	1029,95	1029,95	1072,30	1072,30
Tempo médio		31,50	35,68	47,29	65,59	71,54	82,42

Tabela 4.11: Geração de Colunas para instâncias te80.

tem, então, um limite comparável ao denominado *L2* em Uchoa et al. [4], onde cortes não são adicionados ao problema mestre e as colunas da formulação são definidas por *q-arbs*.

		Limites					
Instâncias	C	L2	$\gamma_{bc} = 1$	$\gamma_{bc} = 2$	$\gamma_{bc} = 3$	$\gamma_{bc} = 4$	$\gamma_{bc} = 5$
cm50r1	200	1073,29	1077,35	1079,59	**1081,30**	1081,30	1081,30
cm50r2		937,70	948,32	950,78	952,18	**952,34**	952,34
cm50r3		1154,12	1175,48	1175,79	1176,08	1176,08	1176,08
cm50r4		758,82	781,87	781,87	781,87	781,87	781,87
cm50r5		894,94	909,77	909,77	909,77	909,77	909,77
Tempo médio		7,38	12,26	24,40	38,89	49,48	56,53
cm50r1	400	652,41	670,59	671,35	671,35	671,35	671,54
cm50r2		592,70	627,64	627,64	628,38	628,38	628,38
cm50r3		678,10	702,75	703,55	703,55	703,55	703,55
cm50r4		489,96	549,55	551,67	**551,73**	551,73	551,73
cm50r5		565,05	598,53	598,79	599,84	601,97	602,31
Tempo médio		74,90	21,95	33,32	43,54	50,01	58,54
cm50r1	800	456,61	477,71	482,51	484,00	484,00	484,00
cm50r2		444,37	502,31	504,58	504,58	504,58	504,58
cm50r3		470,23	441,74	459,04	459,04	459,04	471,45
cm50r4		373,98	464,59	465,18	465,33	467,00	467,00
cm50r5		429,48	405,75	485,02	487,13	487,13	487,60
Tempo médio		817,00	31,38	56,78	70,08	78,62	91,49

Tabela 4.12: Geração de Colunas para instâncias cm50.

Diferentemente das *q-arbs* utilizadas por Uchoa et al. [4], nossas colunas são representadas por soluções inteiras viáveis para o CMRAP. Sendo assim, nosso problema de *pricing* consiste em encontrar uma *r-arborescência-capacitada* de custo mínimo, onde o custo reduzido (3.23) determina o custo associado a cada arco de G.

Neste capítulo, vimos que esse problema é NP-Difícil e, sendo assim, procurar uma solução inteira ótima para cada iteração pode ser impraticável. Por esse motivo, em cada iteração tentamos, inicialmente, encontrar uma solução com custo negativo a partir da heurística proposta na Seção (4.5). Quando a heurística não consegue encontrar uma solução de custo negativo, o nosso algoritmo de geração de colunas então procura uma coluna com custo reduzido negativo a partir do algoritmo Branch-and-cut proposto na Seção (4.4).

Nas Tabelas (4.11) e (4.12), comparamos o limite encontrado pelo algoritmo de geração de colunas descrito acima com o limite *L2* apresentado em Uchoa et al. [4]. Na Tabela (4.11) são consideradas todas as 5 instâncias UD de *te80* alternando a capacidade em {*5*, *10*, *20*}. Enquanto que para a Tabela (4.12) são consideradas as 5 instâncias Non-UD de *cm50r* alternando a capacidade em {*200*, *400*, *800*}.

Para cada execução do nosso algoritmo passamos como parâmetro um valor γ_{bc}, que indica o número máximo de vezes que permitiremos resolver o problema de *pricing* usando o Branch-and-cut. As duas primeiras colunas das Tabelas (4.11) e (4.12) apresentam respectivamente o nome da instância e a capacidade utilizada. A coluna seguinte, *L2*, apresenta o limite reportado por Uchoa et al. [4]. Os limites encontrados pelo nosso algoritmo de geração de colunas são apresentados nas 5 colunas seguintes. Para alguns deles, nosso algoritmo terminou por que, no último problema de *pricing*, o custo da solução ótima foi não negativo. Nesses casos, colocamos o valor do limite em negrito. Para cada uma das 5 colunas usamos um valor $\gamma_{bc} \in \{1, 2, 3, 4, 5\}$ diferente. Em cada tabela, a média do tempo de CPU é calculada para instâncias agrupadas por capacidade. Os tempos médios referentes a coluna *L2* são reportados em Uchoa et al. [4] em um Pentium 3.0GHz.

A Tabela (4.11) mostra que, já na primeira vez que resolvemos o problema de *pricing* com o Branch-and-cut ($\gamma_{bc} = 1$), é possível encontrar limites melhores que o *L2* para quase todas as instâncias UD testadas. Além disso, para as instâncias *te80-1* e *te80-2* com capacidade 5, foi possível provar que não existe outra coluna com custo reduzido negativo com $\gamma_{bc} = 4$. A instância *te80-5* com capacidade 5, por sua vez, teve que resolver Branch-and-cut cinco vezes para obter esse resultado. Na Tabela (4.12) podemos perceber que nosso algoritmo também conseguiu limites melhores que o *L2* para quase todas as instâncias Non-UD testadas já na primeira vez que resolvemos o Branch-and-cut.

Embora o algoritmo tenha parado pelo limite γ_{bc} imposto, nota-se que resolver o

problema de *pricing* usando a heurística DPh e rodar o Branch-and-cut uma única vez já é suficiente para encontrar bons limites para essa reformulação.

Análise da performance da heurística

Nos resultados a seguir, vamos analisar a influência da heurística proposta na Seção 4.5 sob a performance do nosso algoritmo de geração de colunas. Com esse objetivo, executamos nosso algoritmo de geração de colunas para algumas das instâncias Non-UD em que o último subproblema tem solução ótima não negativa. Para medir o impacto de cada nova coluna sob o melhor limite conhecido até a iteração corrente, comparamos nosso algoritmo de geração de colunas com uma versão onde mantemos a heurística desligada. Ou seja, nessa nova versão, em toda iteração, procuramos por uma nova coluna usando o algoritmo *Branch-and-cut*. As Figuras (4.5) comparam, para essas duas versões, o tempo de CPU necessário para encontrar seus respectivos limites.

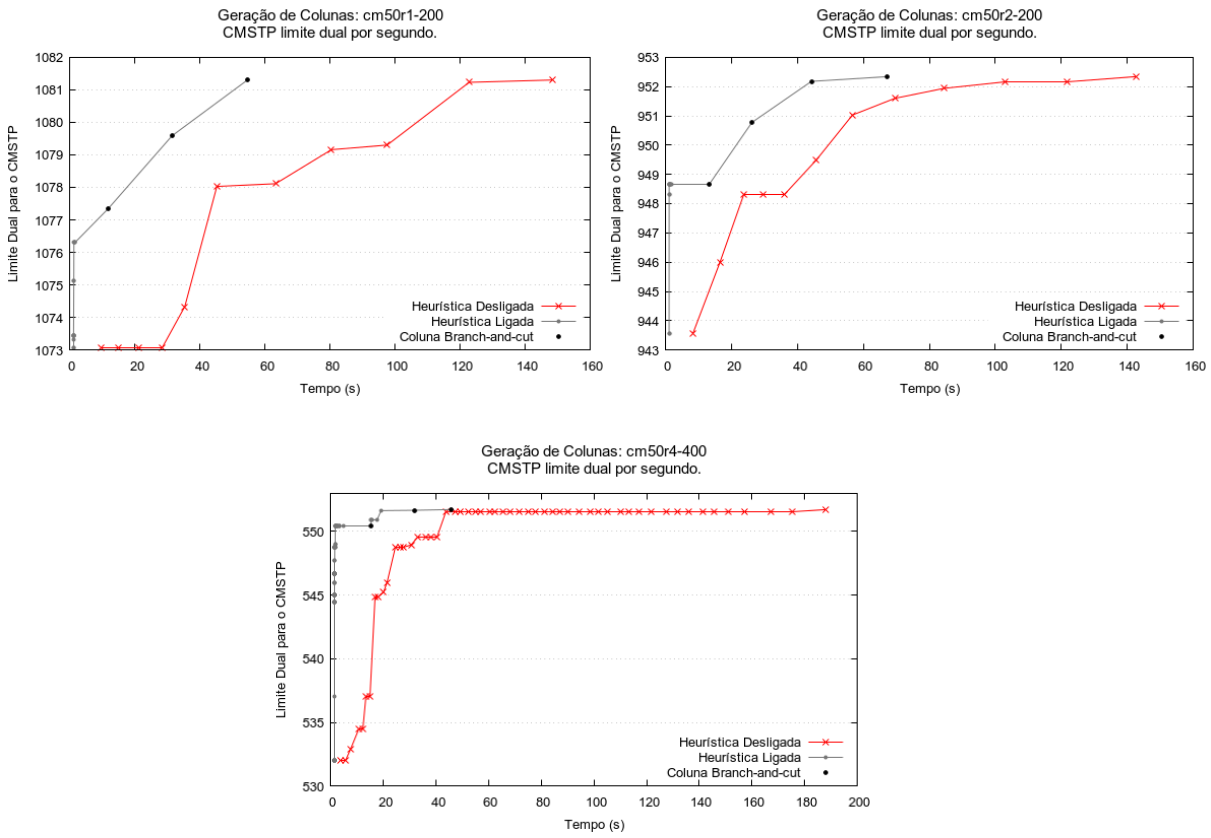

Figura 4.5: Comparação entre as versões do algoritmo de geração de colunas com heurística ligada e desligada.

Para obter os gráficos da Figura 4.5, adaptamos a versão do algoritmo de geração de colunas com heurística ligada para resolver o *Branch-and-cut* em todas as iterações, ou seja, independentemente da solução encontrada pela heurística.

Todavia, somente quando o custo da solução heurística for não negativo que a coluna associada ao *Branch-and-cut* é incorporada ao problema mestre e seu tempo contabilizado para o tempo de CPU. Note que essa adaptação é necessária apenas para computar o limite dual válido de cada iteração encontrado ao longo do nosso algoritmo de geração de colunas original. Nessas figuras, o eixo x e y representam respectivamente o tempo de CPU em segundos e o melhor limite dual conhecido até então. Adicionalmente, para a versão com a heurística ligada, indicamos com um *círculo* preto os momentos em que foram necessários chamar o *Branch-and-cut* para procurar por uma nova coluna.

Para todas as 3 instâncias analisadas, nota-se que a heurística proposta nesse trabalho foi fundamental para acelerar o processo de convergência. Mais especificamente, enquanto que a versão com a heurística ligada leva respectivamente 54, 67 e 45 segundos, a versão com a heurística desligada leva respectivamente 148, 142 e 188 segundos.

4.8 Conclusão

Este capítulo investigou o problema da Arborescência Enraizada Capacitada de custo mínimo (Capacitated Minimum Rooted Arborescence Problem - CMRAP). Inicialmente, apresentamos a definição formal do problema e suas aplicações. Em seguida, uma formulação e uma família de desigualdades são propostas. Para essas desigualdades, ainda apresentamos um algoritmo polinomial de separação. A partir dessas contribuições foi proposto um algoritmo do tipo *branch-and-cut* para o CMRAP. Por último, para os testes computacionais, criamos instâncias para o CMRAP a partir de um algoritmo de geração de colunas para o CMSTP.

Os resultados computacionais mostram que a família de desigualdades proposta neste capítulo melhoraram o limite da relaxação linear em 56,76% nas instâncias UD e em 48,19% nas instâncias Non-UD. Em seguida, conduzimos alguns experimentos sob o algoritmo de geração de colunas para o CMSTP. Os resultados obtidos nesses experimentos indicam que podemos ter um algoritmo competitivo ao apresentado por [4] a partir da introdução de planos-de-corte no problema mestre.

Capítulo 5

Conclusão

Nesta Tese, apresentamos dois trabalhos que tratam da solução de problemas que envolvem a determinação de arborescências ótimas em grafos.

Para o primeiro deles, denominado Problema da Arborescência Mínima (MAP - Minimum Arborescence Problem), apresentamos um algoritmo *Branch-and-cut* sob uma formulação *cutset* computacionalmente mais eficiente que as utilizadas na literatura para resolver o problema. Inicialmente, estudamos o MAP para grafos direcionados (DGs - Directed Graphs) e os resultados computacionais mostraram que, com essa formulação, conseguimos resolver todas as 66 instâncias propostas por Duhamel et al [1] em um tempo médio de 0,19 segundos. Em contraste a esse cenário, o algoritmo *Branch-and-bound* sob a formulação proposta em [1] resolveu 63 dessas instâncias em um tempo médio de 323,45 segundos.

Em seguida, estudamos o MAP restrito a grafos direcionados acíclicos (DAGs - Directed Acyclic Graphs). A primeira contribuição para essa versão é uma formulação com variáveis para os arcos e vértices, denominada *GSEC2*. A segunda contribuição é um novo conjunto de instâncias DAGs para o problema. Essas instâncias se mostraram mais difíceis que as propostas por Blum et al. [2]. Os resultados computacionais mostraram que, apesar de ter o mesmo limite inferior da formulação de Rao et al [6], o algoritmo *Branch-and-Bound* sob a formulação apresentada neste trabalho enumerou menos nós na árvore de *Branch-and-Bound* e foi mais rápido na maioria das instâncias que o algoritmo *Branch-and-Bound* sob a formulação de Rao et al [6]. Mais especificamente, para as instâncias do Blum et al [2], nosso algoritmo resolve todas as 28 instâncias em um tempo médio de 29,55 segundos. Já o *Branch-and-bound* sob a formulação Rao et al [6] resolve apenas 22 dessas instâncias em um tempo médio de 549,16 segundos. Comparando o número de nós explorados, a eficiência da formulação *GSEC-2* fica mais evidente nos testes computacionais para as instâncias propostas na Seção 2.5. Nessas, o *Branch-and-bound* sob a formulação *GSEC-2* foi 1,07 vezes mais rápido e explora, em média, 61% dos nós do *Branch-and-bound* sob a formulação Rao et al [6].

Além disso, para a formulação *GSEC2*, as desigualdades *cutsets* funcionam como cortes que fortalecem seu limite inferior, como foi provado no Capítulo 2. Nos resultados computacionais, vimos que o algoritmo *Branch-and-cut* sob a formulação sugerida neste trabalho fortalecida com as *cutsets* reportaram *speed-up* de até 28,62 e encontrou uma solução ótima para instâncias nas quais a melhor solução conhecida não tinha prova de otimalidade.

No segundo trabalho desta Tese estudamos o Problema da Arborescência Mínima Capacitada (CMAP - Capacitated Minimum Arborescence Problem). Embora esse problema já tenha sido mencionado em Uchoa et al [4], no Capítulo 4 apresentamos uma definição formal do problema sob um grafo direcionado e uma formulação também baseada em *cutsets*. Em seguida, foi proposto um conjunto de cortes que generaliza os cortes *multistar* [10] para o CMSTP. Para esses cortes, descrevemos um algoritmo de separação para encontrar desigualdades violadas em soluções da relaxação linear. Ainda nesse capítulo, apresentamos uma heurística baseada em programação dinâmica, e com isto criamos um conjunto de instâncias para o CMAP a partir de um algoritmo de geração de colunas para o CMSTP. Nos resultados computacionais, conseguimos limites inferiores mais apertados quando utilizamos as *multistars* generalizadas. Mais especificamente, o limite inferior dado pela relaxação linear da formulação *cutset* (4.9) foi 50% melhor, em média, quando a mesma foi fortalecida com os cortes *multistars* generalizadas. Além disso, para as instâncias UD, o algoritmo *Branch-and-Cut* com esses cortes conseguiu um *speed-up* de até 18,45. Por último, apresentamos resultados para nosso algoritmo de geração de colunas cujo problema de *pricing* associado é o CMAP. Esse algoritmo é comparado ao seu equivalente apresentado em [4] e os resultados mostram que nessa versão conseguimos encontrar limites mais apertados. Esses últimos resultados indicam que, com a introdução de planos-de-corte no problema mestre do nosso algoritmo de geração de colunas, será possível encontrar limites duais melhores para o CMSTP do que os apresentados por [4].

Referências Bibliográficas

[1] DUHAMEL, C., GOUVEIA, L., MOURA, P., et al. "Models and heuristics for a minimum arborescence problem", *Networks*, v. 51, n. 1, pp. 34–47, 2008.

[2] BLUM, C., CALVO, B. "A matheuristic for the minimum weight rooted arborescence problem", *Journal of Heuristics*, v. 21, n. 4, pp. 479–499, 2015.

[3] TARJAN, R. E. "Finding optimum branchings", *Networks*, v. 7, n. 1, pp. 25–35, 1977.

[4] UCHOA, E., FUKASAWA, R., LYSGAARD, J., et al. "Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation", *Mathematical Programming*, v. 112, n. 2, pp. 443–472, 2008.

[5] VENKATA RAO, V., MC, G. L. F. *The Minimum Weight Rooted Aborescence Problem: A Branch and Bound Solution.* IIMA Working Papers WP1984-06-01_00590, Indian Institute of Management Ahmedabad, Research and Publication Department, 1984.

[6] RAO, V. V., SRIDHARAN, R. "Minimum-weight rooted not-necessarily-spanning arborescence problem", *Networks*, v. 39, n. 2, pp. 77–87, 2002.

[7] MATEO, S., BLUM, C., FUA, P., et al. "Hybrid Algorithms for the Minimum-Weight Rooted Arborescence Problem." In: *ANTS*, pp. 61–72. Springer, 2012.

[8] LJUBIĆ, I., WEISKIRCHER, R., PFERSCHY, U., et al. "An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem", *Mathematical programming*, v. 105, n. 2-3, pp. 427–449, 2006.

[9] CHOPRA, S., GORRES, E. R., RAO, M. R. "Solving the Steiner Tree Problem on a Graph Using Branch and Cut", *ORSA Journal on Computing*,

v. 4, n. 3, pp. 320–335, 1992. doi: 10.1287/ijoc.4.3.320. Disponível em: <https://doi.org/10.1287/ijoc.4.3.320>.

[10] GOUVEIA, L., LOPES, M. J. "The capacitated minimum spanning tree problem: On improved multistar constraints", *European Journal of Operational Research*, v. 160, n. 1, pp. 47–62, 2005.

[11] VENKATA RAO, V., SRIDHARAN. *The minimum weight rooted arborescence problem: weights on arcs case*. Relatório técnico, IIMA Working Papers WP1992-05-01_01106, Indian Institute of Management Ahmedabad, Research and Publication Department, 1992.

[12] CORNUEJOLS, G., NEMHAUSER, G. "On the uncapacitated plant location problem", *Annals of Discrete Mathematics*, v. 1, pp. 163–177, 1977.

[13] SEGEV, A. "The node-weighted Steiner tree problem", *Networks*, v. 17, pp. 1–17, 1987.

[14] EDMONDS, J. "Optimum branchings", *Journal of Research of the National Bureau of Standards B*, v. 71, n. 4, pp. 233–240, 1967.

[15] MACULAN, N., SOUZA, P., CANDIA VEJAR, A. "An approach for the Steiner problem in directed graphs", *Annals of Operations Research*, v. 33, n. 6, pp. 471–480, 1991.

[16] TÜRETKEN, E., GONZÁLEZ, G., BLUM, C., et al. "Automated reconstruction of dendritic and axonal trees by global optimization with geometric priors", *Neuroinformatics*, v. 9, n. 2-3, pp. 279–302, 2011.

[17] CHOPRA, S., GORRES, E. R., RAO, M. "Solving the Steiner tree problem on a graph using branch and cut", *ORSA Journal on Computing*, v. 4, n. 3, pp. 320–335, 1992.

[18] CHOPRA, S., RAO, M. R. "The Steiner tree problem I: Formulations, compositions and extension of facets", *Mathematical Programming*, v. 64, n. 1, pp. 209–229, 1994.

[19] LEITNER, M., LJUBIĆ, I., LUIPERSBECK, M., et al. "A Dual Ascent-Based Branch-and-Bound Framework for the Prize-Collecting Steiner Tree and Related Problems", *INFORMS Journal on Computing*, v. 30, n. 2, pp. 402–420, 2018.

[20] WOLSEY, L. A. *Integer programming*. New York, NY, USA, Wiley-Interscience, 1998.

[21] GABOW, H. N., GALIL, Z., SPENCER, T., et al. "Efficient algorithms for finding minimum spanning trees in undirected and directed graphs", *Combinatorica*, v. 6, n. 2, pp. 109–122, 1986.

[22] GAVISH, B. "Formulations and algorithms for the capacitated minimal directed tree problem", *Journal of the ACM (JACM)*, v. 30, n. 1, pp. 118–132, 1983.

[23] MALIK, K., YU, G. "A branch and bound algorithm for the capacitated minimum spanning tree problem", *Networks*, v. 23, n. 6, pp. 525–532, 1993.

[24] TOTH, P., VIGO, D. "An exact algorithm for the capacitated shortest spanning arborescence", *Annals of Operations Research*, v. 61, n. 1, pp. 121–141, 1995.

[25] HALL, L. "Experience with a cutting plane algorithm for the capacitated spanning tree problem", *INFORMS Journal on Computing*, v. 8, n. 3, pp. 219–234, 1996.

[26] GOUVEIA, L., MARTINS, P. "The Capacitated Minimal Spanning Tree Problem: An experiment with a hop-indexedmodel", *Annals of Operations Research*, v. 86, pp. 271–294, 1999.

[27] GOUVEIA, L., MARTINS, P. "A hierarchy of hop-indexed models for the Capacitated Minimum Spanning Tree Problem", *Networks*, v. 35, n. 1, pp. 1–16, 2000.

[28] BERGSON, J. *Uma heurística lagrangeana para o problema da árvore geradora capacitada de custo mínimo*. Masters dissertation, Universidade Federal do Rio de Janeiro, 2002.

[29] GOUVEIA, L., MARTINS, P. "The capacitated minimum spanning tree problem: revisiting hop-indexed formulations", *Computers & operations research*, v. 32, n. 9, pp. 2435–2452, 2005.

[30] GZARA, F., GOFFIN, J.-L. "Exact solution of the centralized network design problem on directed graphs", *Networks*, v. 45, n. 4, pp. 181–192, 2005.

[31] ARAQUE, J. R., HALL, L. A., MAGNANTI, T. L. "Capacitated trees, capacitated routing, and associated polyhedra", 1990.

[32] GOUVEIA, L., HALL, L. "Multistars and directed flow formulations", *Networks*, v. 40, n. 4, pp. 188–201, 2002.

[33] HALL, L. A., MAGNANTI, T. L. "A polyhedral intersection theorem for capacitated spanning trees", *Mathematics of Operations Research*, v. 17, n. 2, pp. 398–410, 1992.

[34] LETCHFORD, A. N., SALAZAR-GONZÁLEZ, J.-J. "Projection results for vehicle routing", *Mathematical Programming*, v. 105, n. 2, pp. 251–274, 2006. ISSN: 1436-4646. doi: 10.1007/s10107-005-0652-x. Disponível em: <http://dx.doi.org/10.1007/s10107-005-0652-x>.

[35] ZHANG, N. *Facet-defining inequalities for capacitated spanning trees.* Tese de Doutorado, Princeton University, 1994.

[36] GOUVEIA, L. "A 2n constraint formulation for the capacitated minimal spanning tree problem", *Operations Research*, v. 43, n. 1, pp. 130–141, 1995.

[37] PRIM, R. C. "Shortest connection networks and some generalizations", *Bell system technical journal*, v. 36, n. 6, pp. 1389–1401, 1957.

[38] RESENDE, M. G., VELARDE, J. L. G. "GRASP: Greedy randomized adaptive search procedures", *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, v. 19, n. 1, pp. 61–76, 2003.