GERAÇÃO DE COLUNAS PARA O PROBLEMA DE EMPACOTAMENTO DAS ÁRVORES DE STEINER

Luidi Gelabert Simonetti

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Nelson Maculan Filho, D.Sc

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

Prof. Márcia Helena Costa Fampa, D.Sc.

Prof. Maria Helena Cautiero Horta Jardim, D.Sc.

Prof. João Neiva de Figueiredo, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

MAIO DE 2003

SIMONETTI, LUIDI GELABERT

Geração de colunas para o problema de empacotamento de árvores de Steiner [Rio de Janeiro] 2003

VIII, 52 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2003)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1 - Empacotamento de Árvores de Steiner

2 - Geração de Colunas

3 - Heurísticas primais

4 - Pré-processamento

I. COPPE/UFRJ II. Título (série)

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

GERAÇÃO DE COLUNAS PARA O PROBLEMA DE EMPACOTAMENTO DE ÁRVORES DE STEINER

Luidi Gelabert Simonetti

Maio/2003

Orientadores: Nelson Maculan Filho
Abílio Lucena Pereira Filho

Programa: Engenharia de Sistemas e Computação

Um algoritmo de solução exata, para uma das variantes do problema de empacotamento de árvores de Steiner, é proposto nesta tese. O algoritmo se baseia em uma nova formulação do problema, por nós introduzida, e é do tipo *branch-and-price*. É iniciado por uma heurística primal que gera, além de soluções viáveis (limites superiores) para o problema, um conjunto viável adicional de árvores de Steiner. Essas árvores são usadas para inicializar a geração dinâmica de colunas. Limites inferiores para o problema são obtidos através de relaxação linear do modelo, que é resolvido via geração de colunas. Limites superiores adicionais para o problema são obtidos através da solução ótima de um problema mestre (bastante) reduzido. Os resultados computacionais obtidos com o algoritmo redefinem o "estado da arte" para algoritmos de soluções exatas para o problema de empacotamento de árvores de Steiner.

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

COLUMN GENERATION FOR STEINER TREE PACKING PROBLEM

Luidi Gelabert Simonetti

May/2003

Advisors: Nelson Maculan Filho
Abílio Pereira de Lucena Filho

Department: Systems Engineering and Computer Science

An exact solution algorithm, for one of the variants of the Steiner tree packing problem, is introduced in this thesis. The algorithm is based on a new formulation of the problem, introduced by us. The algorithm is of the branch-and-price type and is initiated with a primal heuristic which generates, in addition to feasible solutions (upper bounds) to the problem, a feasible set of Steiner trees. These trees are used to initialize dynamic column generation. Lower bounds for the problem are obtained through a linear programming relaxation of the model, which is solved via column generation. Additional problem upper bounds are obtained through the optimal solution to a fairly restricted master problem. Computational results, obtained with the algorithm, push forward the state-of-the-art for exact solution algorithms to the Steiner tree packing problem.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Seja $M = \{1, \ldots, m\}$ um conjunto finito e $\mathcal{M} = \{M_j : j \in N\}$, onde $N = \{1, \ldots, n\}$, uma dada coleção de subconjuntos de M. Como exemplo, $\mathcal{M}$ pode representar uma coleção constituída por todos os subconjuntos de M com dimensão k, onde $k \leq m$. Denominamos $\mathcal{F} = \{M_j : j \in F\}$, $F \subseteq N$, um *empacotamento* de M, se $M_j \cap M_k = \emptyset$, $\forall\, j,\ k \in F$, $j \neq k$, ou seja, um empacotamento $\mathcal{F}$ de M é formado por subconjuntos disjuntos de M. Para um dado vetor n-dimensional $\mathbb{C} = \{c_1, \ldots, c_n\}$, onde c_j, $j \in N$, representa um custo associado ao uso de M_j, o custo do empacotamento $\mathcal{F} = \{M_j : j \in F\}$ é calculado como $c(F) = \sum_{j \in F} c_j$ e o *Problema de Empacotamento* (PE) consiste em encontrar um empacotamento de M de maior custo, ou seja, PE pode ser formulado, em termos genéricos, como o seguinte problema de otimização combinatória:

$$\max\{c(F) : \mathcal{F} = \{M_j : j \in F\} \text{ é um empacotamento de } M\}$$

O problema acima pode ser modelado como um problema de programação inteira $0-1$. Isso é feito, definindo-se uma matriz A, de dimensão $m \times n$,

para explicitar os elementos de M contidos em cada $M_j \in \mathcal{M}$. Dessa maneira, os elementos de A são definidos como

$$a_{ij} = \begin{cases} 1 \text{ se } i \in M_j \\ 0 \text{ se } i \notin M_j \end{cases}$$

Utilizando-se uma variável binária $0-1$, x_j, $j \in N$, para controlar a utilização de M_j em um empacotamento de M, teríamos então

$$\max \sum_{j \in N} c_j x_j \tag{1.1}$$

sujeito a

$$\sum_{j=1}^{n} a_{ij} x_j \leq 1 \qquad , i \in M \tag{1.2}$$

$$x_j \in \{0,1\} \qquad , j \in N \tag{1.3}$$

como uma formulação para o PE.

O problema de empacotamento em grafos pode ser tratado de maneira análoga. Considere um grafo $G = (V, E)$, onde V é um conjunto finito de vértices e E um conjunto de arestas. Podemos, então, representar as estruturas que desejamos empacotar no grafo na forma de uma matriz, com elementos assumindo valores binários $0-1$. Para isso, basta que cada coluna da matriz A represente uma estrutura distinta. Podemos, por exemplo, empacotar árvores em G e, sendo assim, cada coluna da matriz A representaria uma árvore do grafo. As colunas de A teriam, então, dimensão $|E|$, com um índice para cada aresta de A. O coeficiente a_{ij} seria igual a 1, se a aresta $i \in E$ pertencesse à árvore j, e zero, caso contrário. Nesse caso, um custo d_i é associado à aresta $i \in E$ e o custo da árvore definida por M_j, $j \in N$, é

dado por $c(M_j) = c_j = \sum_{i \in M_j} d_i$. Desta forma, o problema de empacotamento de árvores pode ser modelado por (1.1)-(1.3).

O problema de empacotamento de árvores de Steiner é um caso particular do problema de empacotamento de árvores em um grafo, já que ao invés de uma árvore qualquer de G, consideramos apenas árvores de Steiner definidas para aquele grafo.

1.1 Aplicações e Motivação

Os circuitos VLSI referem-se à tecnologia através da qual é possível implementar circuitos eletrônicos em silício. A tecnologia de VLSI tem sido usada com sucesso no desenvolvimento de microprocessadores, processadores de sinal, memórias de grande capacidade, controladores de memória e de entrada/saída (E/S), e muitos outros dispositivos. O nível de domínio tecnológico alcançado permite que hoje sejam construídos chips com centenas de milhões de transistores encapsulados. No entanto, projetar um circuito VLSI é uma tarefa extremamente complexa. Sendo assim, esse problema é normalmente dividido em duas etapas: o projeto lógico e o projeto físico. No projeto lógico, são especificadas quais unidades lógicas (células) serão usadas e determinadas quais unidades lógicas deverão ser interligadas. As células têm pontos de contato, chamados terminais, através dos quais as células são interligadas. Um conjunto de terminais, que devem ser interligados, é chamado de rede. A lista de células e a lista de redes são determinadas na fase do projeto lógico e constituem dados de entrada para a fase do projeto físico. No projeto físico, são determinadas as posições das células nas placas de silício e a forma como as redes devem ser conectadas, minimizando uma função objetivo, construída de acordo com certas regras. A maioria dos

problemas de projeto físico são NP-difíceis, e nas aplicações práticas, tanto o número de células quanto o número de redes é tão grande que o problema não pode ser resolvido à otimalidade. Para o problema se tornar tratável, a fase física é, então, decomposta em subproblemas. Primeiro, é resolvido o problema de localização das células. Depois, o curso dos fios que conectam as redes (roteamento global) é determinado aproximadamente. Em seguida, determina-se o caminho exato dos fios, de forma a reduzir o tamanho e o número de camadas (roteamento detalhado). O processo é repetido, até que se encontre uma solução aceitável.

Nesta tese, estuda-se o problema de roteamento detalhado, que pode ser tratado, de forma bastante realista, como um problema de empacotamento de árvores de Steiner. Nosso enfoque se baseia em minimizar o número de camadas, já que isso é fundamental para o desempenho e o custo do circuito. Mais informações sobre VSLI podem ser encontradas em [18, 4].

A principal aplicação do problema de empacotamento de árvores de Steiner é aquela que acabamos de descrever. No entanto, esse modelo pode ser também utilizado, dentre outras aplicações, em problemas associados a redes de distribuição de algum produto, como gás, rede telefônica ou elétrica.

1.2 Empacotamento de árvores de Steiner

Antes de definir, de forma mais precisa, o Problema de Empacotamento de Árvores de Steiner (PEAS), precisamos definir o que é uma árvore de Steiner. A seguir, apresentaremos tais definições.

1.2.1 Problema de Steiner

Seja $G = (V, E)$ um grafo finito não-direcionado, com um conjunto V de vértices e um conjunto E de arestas. Um subconjunto $T \subseteq V$ dos vértices é denominado conjunto dos vértices terminais. Um subgrafo de Steiner é qualquer subgrafo conexo de G que contenha todos os vértices em T. Dada uma função que atribui custos às arestas, o *Problema de Steiner em Grafos* (PSG) consiste em encontrar um subgrafo de Steiner de menor custo, isto é, com a menor soma dos custos das arestas. Quando todos os custos $\{c_e : e \in E\}$ atribuídos às arestas são não-negativos, um subgrafo de Steiner ótimo define, necessariamente, uma árvore de G. Um caso especial do PSG é aquele em que os subgrafos de Steiner estão restritos a definir árvores de G. Podemos, nesse caso, definir uma *Árvore de Steiner* como sendo uma árvore $G' = (V', E')$ de G que contenha todos os vértices em T. O *Problema da Árvore de Steiner* (PAS) consiste, então, em encontrar uma árvore de Steiner de menor custo ou *Árvore de Steiner Mínima* (ASM). Obviamente, diante de custos $\{c_e : e \in E\}$ não-negativos, PSG e PAS são equivalentes.

O PSG consta da lista original de problemas provados NP-difíceis [17]. Algumas referências para o PSG são [21, 27, 19].

Neste trabalho, utilizaremos a notação PAS(G,T) para referenciar o problema da árvore de Steiner para um grafo G e um conjunto de vértices terminais T.

1.2.2 Problema de empacotamento de árvores de Steiner

O problema de empacotamento de árvores de Steiner pode ser enunciado como a seguir.

Dado um grafo não direcionado $G = (V, E)$ e uma família de conjuntos de vértices $\mathcal{T} = \{T_1, \ldots, T_n\}$, com $n \in \mathbb{N}$, considere o problema de encontrar uma árvore de Steiner $S_k = (V_k, E_k)$, $V_k \subseteq V$, $E_k \subseteq E$, para cada T_k, $k = 1, \ldots, n$. Toda coleção de árvores de Steiner $\mathcal{S} = \{S_1, \ldots, S_n\}$ com essas propriedades é chamada de *Empacotamento de Árvores de Steiner*. Note que aqui relaxamos a noção do que seja um empacotamento, pois não é exigido que as árvores de Steiner sejam disjuntas (ou seja, não compartilham uma mesma aresta ou um mesmo vértice). Dependendo da função objetivo a otimizar e das eventuais restrições adicionais associadas, o PEAS tem várias versões. Apresentaremos, a seguir, exemplos freqüentemente citados na literatura, nos quais deseja-se encontrar uma coleção de árvores de Steiner $\mathcal{S}$, onde:

- as árvores de Steiner em $\mathcal{S}$ são disjuntas em relação aos vértices;
- as árvores de Steiner em $\mathcal{S}$ são disjuntas em relação às arestas;
- uma aresta $e \in E$ não pode estar contida em mais de λ_e árvores de Steiner em $\mathcal{S}$;
- o somatório dos custos das árvores de Steiner em $\mathcal{S}$ é o menor possível.

Dado um empacotamento $\mathcal{S}$ de árvores de Steiner, seja $\eta_e \geq 0$ o número de árvores em $\mathcal{S}$ que utilizam a aresta $e \in E$. Nesta tese, temos como objetivo encontrar um empacotamento de árvores de Steiner $\mathcal{S}$, onde $\max\{\eta_e : e \in E\}$ seja o menor possível. Isso implica no menor uso possível de uma mesma aresta por árvores diferentes. Mais informações sobre o PEAS podem ser obtidas em [14, 15, 18].

Na versão a ser tratada nesta tese, um modelo para o PEAS foi sugerido por CHOPRA. No entanto, não existe referência, na literatura, a um algoritmo exato para tratar o problema.

A variante do PEAS onde as árvores de Steiner a serem geradas são disjuntas e o somatório do custo das árvores é o menor possível foi tratada em [16].

No segundo capítulo desta tese, apresentamos formalmente o PEAS aqui estudado e o método escolhido para resolvê-lo. Primeiramente, apresentamos a formulação utilizada para o PEAS e em seguida, os métodos *geração de colunas* e *branch-and-price* usados para resolver o problema.

No terceiro capítulo, descrevemos uma heurística para o problema da árvore de Steiner e propomos uma heurística para o PEAS. Apresentamos, ainda, testes de pré-processamento que facilitam a resolução do problema da árvore de Steiner.

No quarto capítulo, descrevemos o algoritmo utilizado para resolver o PEAS. Já no quinto capítulo, apresentamos os resultados obtidos. Finalmente, no sexto capítulo, apresentamos nossas conclusões e sugestões de trabalhos futuros.

Capítulo 2

O problema de empacotamento de árvores de Steiner

Neste capítulo apresentamos, formalmente, o PEAS e descrevemos a abordagem escolhida para resolvê-lo.

2.1 Definição do problema

Considere um grafo $G = (V, E)$, onde são especificados n subconjuntos disjuntos de vértices terminais, $\mathcal{T} = \{T_i \subseteq V : i \in I\}$, com $I = \{1, \ldots, n\}$. Para um conjunto de arestas $E_i \subseteq E$, $i \in I$, denote por $V(E_i) \subseteq V$ o conjunto de vértices de G induzidos por E_i. Diz-se, então, que $S_i = (V(E_i), E_i)$ é uma árvore de Steiner para o grafo G e o conjunto de vértices terminais T_i, se S_i for uma árvore de G e $T_i \subseteq V(E_i)$.

Antes de apresentar uma formulação para o PEAS, temos que definir o que vem a ser um grafo grade, pois essa é uma característica básica da aplicação prática que motiva nosso estudo do problema.

Considere o conjunto de pontos no plano Euclidiano, definido pelas in-

terseções de um conjunto finito de retas paralelas ao eixo horizontal, com um conjunto finito de retas paralelas ao eixo vertical. Esses pontos podem ser interpretados como sendo os vértices de um grafo grade. Da mesma forma, teremos uma aresta do grafo grade correspondendo àqueles segmentos de retas introduzidos acima que contenham exatamente 2 vértices (pontos). Note que nesse caso, necessariamente, os pontos seriam extremidades dos segmentos. O custo de uma aresta corresponde ao comprimento Euclidiano do segmento de reta associado a ela. Um exemplo de grafo grade, com arestas de mesmo custo, pode ser visto na figura (2.1).

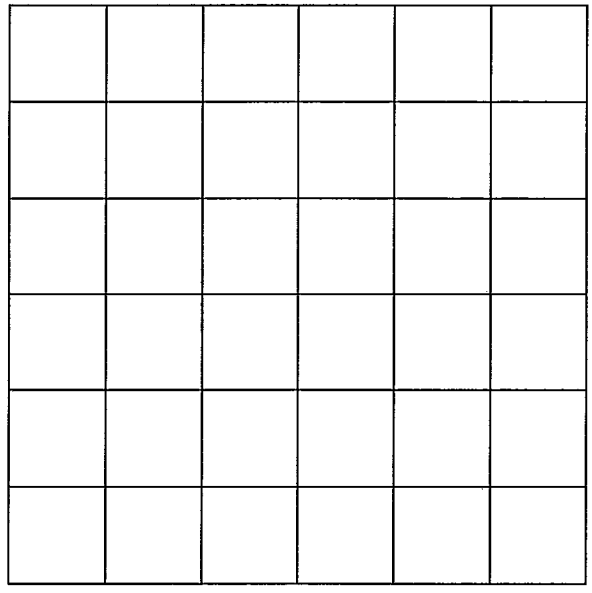

Figura 2.1: Exemplo de grafo grade

Nesta tese, aborda-se uma versão do problema de empacotamento de árvores de Steiner, descrita por:

Instância: Definida por um grafo grade $G = (V, E)$ não-direcionado, com custo unitário para as arestas, e um conjunto $\mathcal{T}$ de subconjuntos de vértices terminais.

Problema: Encontrar $\{E_i : i \in I\}$, tal que

1. $S_i = (V(E_i), E_i), i \in I$, seja uma árvore de Steiner para G e T_i.
2. $w = \max_{e \in E}\{\sum_{i \in I} |E_i \cap \{e\}|\}$ seja mínimo.

O problema acima consiste em encontrar um conjunto de árvores de Steiner de G, uma para cada $T_i \in \mathcal{T}$, onde a aresta $e \in E$ que mais aparece nas árvores de Steiner o faz o menor número possível de vezes.

2.2 Uma formulação para o PEAS

Propomos uma formulação para o PEAS, baseada nas seguintes idéias. Suponha que, para cada $i \in I$, é conhecido o conjunto $\mathcal{E}_i = \{E_{i1}, \ldots, E_{ip_i}\}$, contendo todos os conjuntos distintos de arestas que definem árvores de Steiner para T_i. Denote por $L_i = \{1, \ldots, p_i\}$ o conjunto de índices associados aos elementos de $\mathcal{E}_i$. Então, para resolver o PEAS, basta escolher n árvores de Steiner, uma para cada $i \in I$, de forma a minimizar w, como definido acima.

Para isso, associe variáveis binárias $\{y_{ij} : i \in I, j \in L_i\}$ aos conjuntos de arestas E_{ij} definidos acima e defina como $\mathcal{E}_i^e \subseteq \mathcal{E}_i$, $e \in E$, $i \in I$, o conjunto de todas as árvores de Steiner pertencentes a $\mathcal{E}_i$ que contenham a aresta e. De forma análoga, defina $L_i^e \subseteq L_i$ como sendo o conjunto de índices das árvores em $\mathcal{E}_i^e$.

Uma formulação para o problema em consideração é, então, dada por:

$$(FPEAS): \qquad \min w \tag{2.1}$$

sujeito a

$$\sum_{i \in I} \sum_{j \in L_i^e} y_{ij} \leq w \qquad , \forall\, e \in E \tag{2.2}$$

$$\sum_{j \in L_i} y_{ij} = 1 \qquad , \forall\, i \in I \tag{2.3}$$

$$y_{ij} \in \{0,1\} \qquad , i \in I, j \in L_i, \tag{2.4}$$

As restrições (2.2) garantem que cada aresta $e \in E$ aparece em, no máximo, w árvores de Steiner numa solução. As restrições (2.3) impõem que exatamente uma árvore, por conjunto de vértices terminais T_i, $i \in I$, seja utilizada numa solução. A restrições (2.4) impõem integralidade binária $0-1$ às variáveis do problema.

2.3 Geração de colunas

A geração de colunas foi sugerida por FORD e FULKERSON [11], em 1958, para o problema de fluxo de multimercadorias, e foi generalizada por DANTZIG e WOLFE [7]. A idéia foi aplicada ao problema *cutting stock* por GILMORE e GOMORY [12, 13], sendo essa a primeira aplicação da técnica à resolução de um problema NP-difícil. Entretanto, somente nos últimos vinte anos o método foi explorado computacionalmente com grande sucesso. Para mais informações, vide [2, 8].

Considere um problema de programação linear

$$\min z = \sum_{j=1}^{n} c_j x_j \tag{2.5}$$

sujeito a

$$\sum_{j=1}^{n} p_j x_j = b \tag{2.6}$$

$$x_j \geq 0 \qquad , j = 1, \ldots, n \tag{2.7}$$

onde p_j e b são vetores m-dimensionais e $m < n$. Assuma que uma solução básica inicial x_B, associada a uma base B, com coeficientes de custo c_B, está disponível. Então, o custo reduzido de uma variável x_j, em relação à x_B, para valores $\pi \in \mathbb{R}^m$, associados às variáveis duais correspondentes às restrições (2.6), obtidas na solução de Programação Linear, é definido como

$$r_j \quad = \quad c_j - \pi^T p_j, \tag{2.8}$$

onde $\pi = c_B B^{-1}$. Pela teoria de dualidade, uma solução básica x_B pode ser melhorada, se

$$\min_{j \notin B} r_j \quad = \quad r_s \quad < 0. \tag{2.9}$$

Nesse caso, intruduzir x_s na base (e convenientemente escolhendo uma variável para sair da mesma) nos levaria a uma nova solução básica de menor custo.

Caso o número de colunas em (2.5)-(2.7) seja elevado (assuma, por exemplo, que n é uma função exponencial de m), calcular explicitamente o custo reduzido de cada varíavel pode se tornar impraticável. Felizmente, em muitas aplicações, problemas de programação linear são de tal ordem, que o

conjunto de todas as suas colunas tem uma estrutura bem definida. Por exemplo, em nossa aplicação, as colunas de (2.5)-(2.7) estão associadas às árvores de Steiner, definidas pelos diferentes conjuntos de vértices terminais. No que se segue, vamos assumir que todas as colunas p_j pertencem a um conjunto Υ que, tipicamente, é um conjunto formado por todos os vetores m-dimensionais que satisfazem algum sistema de equações lineares bem definido. Mais uma vez, lançando mão de nossa aplicação, o conjunto Υ seria definido por formulações para problemas de Steiner em grafos, definidos para cada conjunto de vértices terminais. Sendo assim, a coluna a entrar na base seria escolhida através da solução do subproblema

$$\min_{p_j \in \Upsilon} \; c(p_j) - \pi p_j, \tag{2.10}$$

onde $c(p_j) \equiv c_j$ é uma dada função de p_j.

A técnica é chamada de geração de colunas porque, ao resolver o problema (2.10), apenas um pequeno subconjunto das colunas em Υ é examinado, e estas colunas são geradas explicitamente apenas quando requeridas. O problema (2.10) é chamado de *Subproblema* ou *Problema Auxiliar* (PA). Já o problema (2.5)-(2.7) é chamado *Problema Mestre* (PM). Quando o problema (2.5)-(2.7) é restrito a um subconjunto de suas variáveis, aquelas utilizadas na geração de colunas, ele é chamado *Problema Mestre Restrito* (PMR). Valores para as variáveis duais obtidas na solução do PMR são utilizados para gerar colunas através do PA.

Um pseudo-código para o algoritmo de geração de colunas, assumindo que não ocorre ciclagem, é descrito na Figura (2.2).

Algoritmo 1 : Geração de colunas

1: Escolha um "pequeno" subconjunto $\{x_j : j \in J\}$, $J \subset \{1, \ldots, n\}$, de variáveis.

2: Obtenha a solução básica ótima x_B do problema de programação linear

$$\min\{\sum_{j \in J} c_j x_j \ : \sum_{j \in J} p_j x_j \leq b, \ x_j \geq 0, j \in J\}$$

e determine os valores das variáveis duais π.

3: **Se** $r_j = c_j - \pi^T p_j \geq 0$ para toda variável $j \in \{1, \ldots, n\} \setminus J$ **então**

4: Pare.

5: **fim Se**

6: Adicione a coluna j com $r_j < 0$ em J.

7: Volte para (2)

Figura 2.2: Algoritmo de geração de colunas

2.3.1 Geração de colunas para o PEAS

No caso específico de nossa formulação para o PEAS, como $|L_i|$, $i \in I$, pode ter uma ordem de grandeza elevada para instâncias não triviais do problema, é inviável trabalhar diretamente com os conjuntos $\mathcal{E}_i$, $i \in I$. Optamos, então, por uma abordagem que usa geração de colunas, onde cada coluna considerada é uma solução viável E_{ij} do problema de Árvore de Steiner de G para um dado conjunto T_i, ou seja, estaremos sempre trabalhando com uma formulação para o PMR, que considera, ao invés do conjunto de árvores $\mathcal{E}_i$, $i \in I$, um subconjunto $\bar{\mathcal{E}}_i \subseteq \mathcal{E}_i$, indexado por $\bar{L}_i = \{1, \ldots, \bar{p}_i\}$, das mesmas. De forma análoga à feita anteriormente, defina como $\bar{\mathcal{E}}_i^e \subseteq \bar{\mathcal{E}}_i$, $e \in E$, $i \in I$, o conjunto de todas as árvores de Steiner pertencentes a $\bar{\mathcal{E}}_i$, $i \in I$, que contenham a aresta e. Defina $\bar{L}_i^e \subseteq \bar{L}_i$ como o conjunto de índices

das árvores em $\bar{\mathcal{E}}_i^e$.

Para um dado PMR, não se tem de antemão uma garantia de que a solução ótima para a relaxação linear de (2.1)-(2.4) possa ser obtida através unicamente das variáveis definidas pelas árvores em $\{\bar{\mathcal{E}}_i : i \in I\}$, já utilizadas no PMR. Dada uma relaxação linear ótima para o PMR, o método de geração de colunas procura encontrar variáveis associadas a árvores em $\mathcal{E}_i \setminus \bar{\mathcal{E}}_i$, com custo reduzido negativo. Caso inexistam tais variáveis, uma solução ótima para a relaxação linear do PMR corresponderá à relaxação linear de (2.1)-(2.4). Em caso contrário, o PMR é expandido, com a incorporação de variáveis de (2.1)-(2.4) com custo reduzido negativo, e reotimizado. Esse processo continua até que não existam mais colunas com custo reduzido negativo para se levar ao PMR. No nosso caso, as variáveis duais do PMR são dadas por $\pi = (\mu, \nu)$, onde $\mu \in \mathbb{R}^{|E|}$ e $\nu \in \mathbb{R}^n$. Seja r_{ij} o custo reduzido de uma variável não básica y_{ij}, associada à coluna p_{ij}, definida para (E_{ij}, e_i), onde $E_{ij} \in \mathbb{R}^{|E|}$ e $e_i \in \mathbb{R}^n$, onde e_i é um vetor com uma única entrada não nula igual a 1 na posição i. Dessa forma, para o coeficiente c_{ij} associado a y_{ij} na função objetivo, teremos

$$\begin{aligned} r_{ij} &= c_{ij} - \pi^T p_{ij} \\ &= c_{ij} - (\mu, \nu)^T (E_{ij}, e_i) \\ &= -\mu^T E_{ij} - \nu_i. \end{aligned}$$

Para o PMR na geração de colunas, mantemos as restrições (2.2) e (2.3) e relaxamos as restrições em (2.4), obtendo:

$$(PMR): \qquad \min w \tag{2.11}$$

sujeito a

$$\sum_{i \in I} \sum_{j \in \bar{L}_i^e} y_{ij} \leq w \qquad , \forall\, e \in E \tag{2.12}$$

$$\sum_{j \in \bar{L}_i} y_{ij} = 1 \qquad , \forall\, i \in I \tag{2.13}$$

$$0 \leq y_{ij} \leq 1 \qquad , i \in I, j \in \bar{L}_i, \tag{2.14}$$

Para gerar colunas atraentes para adicionar ao PMR, devemos resolver uma série de PAs, um para cada $T_i \in \mathcal{T}$. Esses problemas consistem basicamente em encontrar uma árvore de Steiner de G para T_i. Formalmente, teremos:

$$(PA_i): \qquad \min -\mu^T E_{ij} - \nu_i \tag{2.15}$$

sujeito a

$$E_{ij} \in PAS(G, T_i) \tag{2.16}$$

Sendo assim, em cada iteração de geração de colunas, resolvemos n problemas (2.15)-(2.16), um para cada conjunto de vértices terminais $T_i, i \in I$. Utilizamos a formulação do PSG proposta em [21] para obter árvores de Steiner ótimas.

A formulação do PSG, proposta em [21], pressupõe a utilização de um grafo orientado e a escolha de um vértice terminal para atuar como raiz. Um digrafo $D = (V, A)$ pode ser construído, a partir de $G = (V, E)$, associando-se a cada aresta $e = [j, k]$ de G dois arcos (j, k) e (k, j) em D, com custos $\mu_{jk} = \mu_{kj} = \mu_e$. Para cada conjunto de vértices terminais T_i, $i \in I$, escolha

um vértice qualquer $t_i^0 \in T_i$ para atuar como raiz da árvore de Steiner a ser gerada.

Para um dado conjunto de vértices terminais T_i, $i \in I$, com a raiz $t_i^0 \in T_i$, seja $\bar{T}_i = T_i \setminus t_i^0$. Para cada vértice terminal $l \in \bar{T}_i$, uma unidade de fluxo de um produto, também denotada por l, por simplicidade, é enviada de t_i^0 a l. Defina $x_e \in \{0,1\}$, $e \in E$, como uma variável que controla o uso ou não da aresta e do grafo G na solução. Seja, ainda, $z_{jk}^l \in \{0,1\}$, $\forall\ (j,k) \in A$, $\forall\ l \in \bar{T}_i$, uma variável que define se um arco $(j,k) \in A$ de D é utilizado para enviar uma unidade de fluxo do tipo l, da raiz t_i^0 ao vértice terminal l. Sendo assim, o PA associado ao conjunto de vértices terminais T_i é descrito por

$$(FPA_i :) \qquad \min \sum_{e \in E} -\mu_e\, x_e - \nu_i \tag{2.17}$$

sujeito a

$$\sum_{(j,k) \in A} z_{jk}^l - \sum_{(k,j) \in A} z_{kj}^l = \begin{cases} 1, \text{ se } j = t_i^0 \\ -1, \text{ se } j = l \\ 0, \text{ se } j \notin T_i \end{cases} \tag{2.18}$$

$$,\forall\ j \in V,\ \forall\ l \in \bar{T}_i$$

$$z_{jk}^l + z_{kj}^l \leq x_e \qquad ,\forall\ e = (j,k) \in E,\ l \in \bar{T}_i \tag{2.19}$$

$$z_{jk}^l \geq 0 \qquad ,\forall\ [j,k] \in A,\ \forall\ l \in \bar{T}_i \tag{2.20}$$

$$x_e \in \{0,1\} \qquad ,\forall\ e \in E. \tag{2.21}$$

Na função objetivo (2.17), as variáveis duais do PMR associadas às restrições em (2.12) são expressas pelo vetor μ. Por sua vez a variável dual, associada àquela restrição em (2.13), referente à unicidade de uma árvore E_{ij}, $i \in I$, $j \in \bar{L}_i$, na solução, é expressa por ν_i.

As restrições de equilíbrio de fluxos, (2.18), impõem a conectividade das

soluções. Para cada $l \in \bar{T}_i$, uma unidade da mercadoria l é enviada de t_i^0 ao vértice l. Dessa forma, se $j = t_i^0$, o lado direito da equação (2.18) será igual a 1, para cada $l \in \bar{T}_i$, pois uma unidade do fluxo l estará saindo de j. Quando $j = l$, o lado direito de (2.18) será igual a -1, pois uma unidade de mercadoria l estará chegando a j. Quando $j \notin T_i$, o lado direito de (2.18) será igual a zero, pois j atuará simplesmente como um vértice de passagem para as mercadorias enviadas a partir de t_i^0.

As desigualdades (2.19) garantem que só existe fluxo de um produto l passando pelo arco $(j,k) \in A$ ou pelo arco $(k,j) \in A$, se a aresta $e = (j,k)$, $e \in E$, estiver na solução. As restrições (2.20) garantem que as variáveis de fluxo z_{jk}^l, $\forall\ (j,k) \in A$, $\forall\ l \in \bar{T}_i$, sejam não-negativas. As restrições (2.21) impõem integralidade binária às variáveis x_e, $e \in E$.

Evitando a ocorrência de ciclos nas soluções

Para garantir que a solução ótima de (2.17)-(2.21) induza uma árvore de G, todos os custos $\{\mu_e : e \in E\}$ devem ser estritamente positivos. Mesmo diante de custos não-negativos, ciclos podem ocorrer quando algumas das arestas, na solução, tiverem custo nulo. No entanto, uma situação mais comum seria a ocorrência de árvores com folhas desnecessárias (isto é, folhas definidas por vértices não terminais, com um custo nulo para a aresta correspondente). Para garantir que uma solução ótima não contenha nenhum ciclo ou aresta desnecessária de custo nulo, perturbamos o vetor μ. A razão para isto é o fato de o vetor dual μ poder ter componentes com valor igual a zero. O problema foi resolvido da forma descrita a seguir.

Para um grafo $G = (V,E)$, com custos $\{\ \mu_e \geq 0 : e \in E\}$, particione as arestas de E em $E_0 = \{\ e \mid \mu_e = 0\}$ e $E_+ = \{\ e \mid \mu_e > 0\}$. Definindo $\bar{\mu} = \min\{\ \mu_e \mid e \in E_+\}$, faça $\mu_e = \frac{\bar{\mu}}{|E_0|+1}$, para todo $e \in E_0$. Dessa forma,

todas as arestas de E passam a ter um custo positivo e, sendo assim, uma aresta com um custo original nulo só será utilizada, numa solução ótima, por uma necessidade real de atender às restrições de conectividade.

Obtida uma solução ótima para (2.17)-(2.21), com perturbação dos custos, expressamos essa solução, sob os custos originais, para verificar se o valor da função objetivo correspondente é negativo ou não. Formalmente, seja $S_{ij} = \{ e \mid e \in E_{ij} \}$, onde E_{ij} é o conjunto das arestas que definem a árvore de Steiner ótima encontrada. Definindo $\chi = \{ e \mid e \in E_+ \cap S_{ij} \}$, o valor da função objetivo associado à árvore de Steiner é $\sum_{e \in \chi} \mu_e - \nu_i$.

Evitando gerar novamente uma coluna já introduzida no PMR

Para evitarmos que uma coluna seja gerada novamente, a cada solução E_{ij} obtida com custo reduzido negativo, inserimos o seguinte corte em PA_i (2.17)-(2.21), $i \in I$:

$$\sum_{e \in E_{ij}} x_e \leq |E_{ij}| - 1, \tag{2.22}$$

onde $x_e \in \{0, 1\}$ é a variável do modelo (2.17)-(2.21) associada à aresta e. Este corte impedirá que a solução expressa pelas arestas E_{ij}, encontrada anteriormente, seja gerada novamente. Da mesma forma, impede que uma solução expressa por $E_{ij} \cup \Gamma$, onde $\Gamma \subseteq (E \setminus E_{ij})$, possa ser gerada. Note que, sem a presença de (2.22) a solução associada a E_{ij} poderia ser gerada múltiplas vezes.

2.4 *Branch-and-Price*

O algoritmo de geração de colunas, descrito na seção (2.3), se aplica à resolução de um problema de Programação Linear. Caso o problema a ser resolvido seja de *Programação Inteira* (PI), devemos, inicialmente, resolver sua relaxação linear. Obviamente, como o problema de relaxação linear associado a PI é de Programação Linear, o algoritmo de geração de colunas anteriormente descrito pode ser aplicado a ele. Caso a solução ótima do problema relaxado seja viável para PI, essa solução inteira seria também ótima para PI. Em caso contrário, teríamos que recorrer ao método *branch-and-bound* para resolver PI de maneira exata. Um algoritmo *branch-and-bound* que utiliza geração de colunas é denominado *branch-and-price*. Essa versão do algoritmo *branch-and-bound* foi proposta em [8] e foi denominada *branch-and-price* por SAVELSBERGH [25].

Em um algoritmo *branch-and-price*, é necessário definir regras de ramificação, que não afetem a estrutura básica dos problemas auxiliares, nem afetem o corte de procura em árvore. As regras de corte de procura utilizadas são basicamente as mesmas do método *branch-and-bound*: (1) cortar quando a relaxação linear do problema corrente é maior ou igual ao limite superior encontrado até o momento; (2) cortar quando é encontrada uma solução inteira; (3) cortar quando o problema é inviável. As regras de ramificação são responsáveis por dividir a região de viabilidade do problema em regiões menores, com o objetivo de facilitar a obtenção de soluções inteiras e limites inferiores mais apertados. Ramificações são implementadas através da definição de nós em uma árvore de procura. Ao longo da procura em árvore, são gerados limites inferiores (i.e., relaxações lineares obtidas através de geração de colunas) e limites superiores (i.e., soluções inteiras viáveis), até que a otimalidade seja provada.

Mais informações sobre o assunto podem ser obtidas em [30, 2, 31].

2.4.1 *Branch-and-Price* para o PEAS

Propomos neste estudo uma regra de ramificação para o *branch-and-price*, que atua diretamente sobre as arestas do problema. Supondo que w_{inf} é o valor da função objetivo do problema de relaxação linear em mãos e w_{sup} é a melhor solução inteira até agora encontrada para o problema, escolhemos uma aresta $e \in E$, que esteja sendo utilizada w_{inf} vezes. Para tal aresta, dividimos o problema em w_{sup} ramos. Os ramos impõem o uso da aresta e em uma solução, respectivamente, $w_{sup} - 1$, $w_{sup} - 2$, $\cdots$ e 0 vezes.

A principal limitação dessa regra é a de possuir uma característica muito combinatória. Quando fixarmos a utilização da aresta e num valor $w \neq 0$, temos que fazer isso separadamente para cada subproblema. Como exemplo, suponha que $|T| = 3$, e que queremos impor a utilização de uma aresta e em exatamente duas árvores de Steiner numa solução. Isso é implementado através das seguintes ramificações: uma que força o uso da aresta nas árvores associadas unicamente a T_1 e T_2, outra para T_1 e T_3 e, finalmente, outra para T_2 e T_3. De forma geral, teremos $C^n_{w_{sup}}$ subregiões de viabilidade a explorar.

A implementação da regra é simples: podemos controlar o uso de uma aresta $e \in E$, definindo quantos e quais subproblemas impõem a utilização da aresta numa solução. Para fixarmos o uso de uma aresta e em (PA_i), $i \in I$, dividimos o conjunto $\bar{\mathcal{E}}_i$ em dois subconjuntos: $\bar{\mathcal{E}}_i^e$ e $(\bar{\mathcal{E}}_i \setminus \bar{\mathcal{E}}_i^e)$. As variáveis y_{ij} do problema (2.11)-(2.14), relacionadas às árvores de Steiner pertencentes ao subconjunto $(\bar{\mathcal{E}}_i \setminus \bar{\mathcal{E}}_i^e)$, são fixadas em zero. Para garantir a utilização da aresta e nas colunas a serem geradas, modificamos o limite inferior da variável x_e em (PA_i) (2.17)-(2.21) para 1. No entanto, por si só, isso não garante a conexidade da solução. Para obtermos essa garantia, inserimos o

corte descrito a seguir, que obriga a árvore de Steiner a ser obtida de (PA_i) (2.17)-(2.21) a utilizar a aresta $e = (j, k)$:

$$\sum_{l \in \bar{T}_i} z^l_{jk} + z^l_{kj} \geq 1, \tag{2.23}$$

No caso de desejarmos eliminar uma aresta e de (PA_i), fixamos em zero as variáveis y_{ij} correspondentes em (2.11)-(2.14), pertencentes ao subconjuto $\bar{\mathcal{E}}^e_i$. Em (PA_i) (2.17)-(2.21), modificamos o limite superior da variável x_e para zero. Utilizando os dois procedimentos de fixação e eliminação para cada (PA_i), $i \in I$, podemos controlar o uso de uma aresta e.

Capítulo 3

Heurísticas primais e pré-processamento

Neste capítulo, estudaremos procedimentos que visam a reduzir o tempo computacional requerido pelo algoritmo aqui proposto para resolver o PEAS. Isso é feito, encontrando, inicialmente, soluções viáveis (não necessariamente ótimas) para os problemas auxiliares, com a utilização de heurísticas. A seguir, com o objetivo de acelerar o processo de solução, tentaremos gerar um bom conjunto inicial de colunas para o PMR (2.11)-(2.14). Finalmente, tentaremos, também, pré-processar as instâncias a serem resolvidas, com o objetivo de eliminar variáveis garantidamente subótimas.

3.1 Soluções heurísticas para o PAS

No algoritmo proposto para a resolução do PEAS, a operação que mais consome tempo de processamento se refere à construção de árvores de Steiner ótimas para cada conjunto de vértices terminais em $\mathcal{T}$. É interessante notar que, dentre árvores de Steiner ótimas, por questões tecnológicas, aquelas

com menor cardinalidade de arestas são preferíveis. Dessa forma, para os nossos propósitos, uma árvore de Steiner gerada por uma heurística, para um dado conjuto $T_i, i \in I$, levando em conta a questão de cardinalidade, pode vir a ser tão boa para o processo de solução do PEAS, quanto aquela obtida por um algoritmo exato. Essa afirmativa procede, pois, em nossos problemas auxiliares, a função objetivo, que é constituída por variáveis duais do problema (2.11)-(2.14), normalmente possui muitos coeficientes de valor zero. Poderíamos assim, eventualmente, gerar uma árvore de Steiner com custo mínimo, mas com cardinalidade maior que o necessário. Outro ponto, ainda mais importante, é que para um PAS de dimensão reduzida (do tipo com o qual trabalhamos), algumas heurísticas propostas na literatura tendem a encontrar soluções ótimas com alta freqüência. Isso nos indica que o uso de heurísticas para resolver o PAS, para os conjutos de vértices terminais $T_i \in \mathcal{T}$, pode reduzir o tempo total de processamento para a solução do PEAS, de forma significativa. Com esse intuito, optamos por adotar o algoritmo de aproximação proposto por TAKAHASHI e MATSUYAMA [26], cujo uso é muito difundido e se mostrou rápido e eficiente, na prática.

O algoritmo de Takahashi e Matsuyama é uma modificação do tradicional algoritmo proposto por PRIM [24], para a resolução do *Problema da Árvore Geradora de custo Mínimo* (PAGM). Neste algoritmo, uma árvore geradora de custo mínimo é construída, seqüencialmente, a partir de um nó raiz, qualquer. Em cada passo, um vértice é adicionado à árvore geradora em construção, através de uma aresta de menor custo. Após $|V| - 1$ passos, todos os vértices do grafo terão sido incluídos na árvore, que se torna, assim, geradora. A heurística correspondente para o PAS opera de forma análoga, com um vértice terminal escolhido como raiz. Em cada iteração, adicionamos à solução o vértice terminal mais próximo, ou seja, aquele que

se encontra a uma distância mínima da árvore de Steiner em construção. No processo, adicionamos à solução todos os vértices no caminho mínimo que leva ao vértice terminal escolhido. Dessa maneira, em $|T_i| - 1$ iterações, uma árvore de Steiner é gerada para um conjunto de vértices terminais T_i. Um pseudo-código para a heurística é descrito na figura (3.1).

Algoritmo 2 Heurística para o PAS

1: Escolher um vértice terminal para atuar como raiz. Utilizar o algoritmo de DIJKSTRA [9] para encontrar um caminho de menor custo entre o nó raiz e outro vértice terminal. Incluir na solução todas as arestas nesse caminho.

2: Utilizar o algoritmo de DIJKSTRA [9] para encontrar um caminho de menor custo entre os vértices na solução e vértices terminais fora da solução. Incluir na solução todas as arestas nesse caminho.

3: Se existir um vértice terminal fora da solução, vá para o (2). Caso contrário, pare.

Figura 3.1: Algoritmo de Takahashi e Matsuyama

A complexidade do algoritmo de Takahashi e Matsuyama é $O(|T|(|E| + |V| \log |V|)$, usando as árvores de Fibonacci para implementação da fila de prioridades utilizada no algoritmo de Dijkstra. Propostas para tornar a heurística mais rápida, sem alterar a complexidade, podem ser encontradas em [1]. O algoritmo tem um fator de aproximação 2, ou seja, garante a obtenção de uma árvore de Steiner com um custo que excede o da solução ótima em no máximo 100%.

3.2 Problema de empacotamento de árvores de Steiner

Pela necessidade de inicializar o algoritmo *branch-and-price* com um conjunto de colunas que contenha uma solução viável para o PEAS e também para tentar agilizar o processo de convergência do algoritmo, introduzimos nesta tese uma heurística para o PEAS. A heurística consiste em utilizar repetidas vezes o algoritmo de TAKAHASHI e MATSUYAMA [26], para os diferentes conjuntos de vértices terminais em $\mathcal{T}$, de forma a construir, seqüencialmente, uma solução viável para o PEAS. A cada iteração, as arestas que aparecem em árvores de Steiner geradas em iterações anteriores são penalizadas (i.e., tornam-se menos atraentes) para iterações futuras. Dependendo da seqüência de vértices terminais T_i, $i \in I$, escolhida para as iterações, as soluções obtidas para o PEAS podem variar em qualidade. Outra forma de chegarmos indiretamente a uma solução viável para o PEAS é gerar, para cada T_i, $i \in I$, $n_3 \geq 1$ árvores de Steiner distintas. Isso é feito, seqüencialmente, penalizando as arestas associadas a uma árvore, de forma a torná-las menos atraentes para as árvores geradas posteriormente. Esse universo de $n_3 \times |\mathcal{T}|$ árvores geradas se mostrou bastante atraente, contendo soluções viáveis para o PEAS de excelente qualidade. Combinando as duas idéias para obter uma solução viável para o PEAS, propomos a heurística detalhada na figura (3.2).

Algoritmo 3 Heurística para o PEAS

Entrada: $n_1 \geq 0$, $n_1 \in \mathbb{Z}$, $n_2 \geq 0$, $n2 \in \mathbb{Z}$ e $n_3 > 0$

1: Executar o Procedimento 1

2: Executar o Procedimento 2

Figura 3.2: Heurística para o PEAS

Algoritmo 4 Procedimento 1

Entrada: $n_1 \geq 0$, $n_1 \in \mathbb{Z}$, $n_2 \geq 0$ e $n2 \in \mathbb{Z}$

1: $k \leftarrow 1$
2: **Para** $z = 1$ até n_1 **faça**
3: **Para** $i = 1$ até n **faça**
4: **Para todo** $e \in E$ **faça**
5: $c[e] \leftarrow 1$
6: **fim Para**
7: Construir o vetor O com tamanho n, onde o primeiro elemento tem valor i e os demais $n-1$ elementos possuem os valores $\{1, \ldots, n\} - \{i\}$, distribuídos de forma randômica. Essas seqüências não podem se repetir.
8: **Para** $w = 1$ até $n2$ **faça**
9: **Para** $j = 1$ até n **faça**
10: $S[k] \leftarrow$ Rodar o algoritmo (2) para $T_{(O_{[j]})}$
11: **Para todo** $e \in S[k]$ **faça**
12: $c[e] \leftarrow c[e] + |V| - 1$
13: **fim Para**
14: $k \leftarrow k + 1$
15: **fim Para**
16: **fim Para**
17: **fim Para**
18: **fim Para**

Algoritmo 5 Procedimento 2

Entrada: $n_3 > 0$

1: **Para** $i = 1$ até n **faça**
2: **Para todo** $e \in E$ **faça**
3: $c[e] \leftarrow 1$
4: **fim Para**
5: **Para** $j = 1$ até n_3 **faça**
6: $S[k] \leftarrow$ Rodar o algoritmo (2) para T_i
7: **Para todo** $e \in S[k]$ **faça**
8: $c[e] \leftarrow c[e] + |V| - 1$
9: **fim Para**
10: $k \leftarrow k + 1$
11: **fim Para**
12: **fim Para**

Utilizamos os valores 2, 2 e 10, respectivamente para n_1, n_2 e n_3. A complexidade da heurística proposta é $O(n_1 \times n_2 \times n^2 + |T| \times n_3)|T|(|E| + |V| \log |V|)$. Se constatarmos, ao término da heurística, que foi gerada uma mesma árvore de Steiner mais de uma vez, eliminamos as duplicatas.

3.3 Pré-processamento

Um fator fundamental para o bom desempenho dos algoritmos exatos recentemente propostos para o PAS, é, sem dúvida, a incorporação de uma fase de pré-processamento efetiva. Essa fase consiste em aplicar algoritmos polinomiais que têm por objetivo reduzir o tamanho de entrada do problema. Naturalmente, na aplicação desses algoritmos, deve estar previsto um mecanismo de "volta", para converter as soluções ótimas, obtidas para uma

instância reduzida, em soluções ótimas para a instância original.

O modo usual de executar esse pré-processamento é através de testes "condição-ação". Cada um desses testes pode ser visto como um teorema sobre o PAS: se a condição é verdadeira, então, uma solução ótima da instância reduzida corresponde a uma solução ótima da instância original. Testes de redução para o PAS podem ser encontrados em [10, 28].

Como a função objetivo do PAS aqui estudado tem por coeficientes as variáveis duais do problema (2.11)-(2.14) e G é um grafo grade, conseguimos obter expressivas reduções através apenas da aplicação de alguns testes bastante simples. Dessa maneira, optamos por não implementar testes mais sofisticados. Ainda assim, acreditamos que mesmo se utilizássemos testes mais sofisticados, não obteríamos grandes alterações nos resultados do pré-processamento.

Diz-se que uma aresta $e \in E$ é *escolhível*, se existir pelo menos uma *Árvore de Steiner de Custo Mínimo* (ASCM) que a contenha. Tão logo uma aresta escolhível $e = (u, v)$ é identificada, ela pode ser introduzida na solução, e os vértices u e v que a definem podem ser contraídos (em um único vértice terminal). Diz-se que uma aresta $e \in E$ é *redundante*, se existir pelo menos uma ASCM que não a contenha. As arestas redundantes são simplesmente removidas.

3.3.1 Testes de Redução

Antes de apresentarmos testes de redução, algumas definições são necessárias. Seja $\mathcal{P}(u, v)$ o conjunto dos diferentes caminhos entre dois vértices u e v pertencentes a V. A distância usual entre u e v pode ser definida como

$$d(u, v) = \min\{c(P) \mid P \in \mathcal{P}(u, v)\},$$

onde $c(P)$ é a soma dos custos das arestas em P. Seja P um caminho entre u e v, e defina $T(P)$ como sendo o conjunto formado pelos vértices u e v e todos os vértices terminais encontrados em P. A *distância de Steiner* $SD(P)$ ao longo de P é o custo do maior subcaminho em P, ligando dois vértices pertencentes a $T(P)$, que apareçam consecutivamente em P. A *distância de Steiner gargalo* entre u e v é definida como

$$B(u,v) = \min\{SD(P) \mid P \in \mathcal{P}(u,v)\}.$$

A distância de Steiner gargalo para u e v, restrita a caminhos que não incluem uma dada aresta e é definida como

$$B(u,v)^{-e} = \min\{SD(P) \mid P \in \mathcal{P}(u,v);\ e \notin P\}.$$

Observe que se e desconecta u e v, então $B(u,v)^{-e} = \infty$.

Apresentaremos, a seguir, os testes aqui utilizados.

Teste 3.3.1 (NTD1) *Um vértice não-terminal v de grau 1 e sua única aresta incidente (v,u) podem ser removidos.*

O teste se aplica, pois sempre que (v,u) faz parte de uma árvore de Steiner, uma árvore de Steiner de menor custo é obtida com a remoção daquela aresta.

Teste 3.3.2 (NTD2) *Um vértice não-terminal v de grau 2 e suas arestas adjacentes (v,u) e (v,w) podem ser substituídos por uma única aresta (u,w) com custo $c(v,u) + c(v,w)$.*

A justificativa aqui é que, pelo explicado acima, (v,u) e (v,w) não podem aparecer isoladamente em uma árvore de Steiner ótima.

Teste 3.3.3 (TD1) *Se* $|T| \geq 2$, *a aresta adjacente a um terminal de grau* 1 *é escolhível.*

A justificativa para a validade de TD1 é a imposição de que os vértices terminais façam parte de qualquer árvore de Steiner. Dessa forma, a única aresta incidente em um vértice terminal de grau 1 deve, necessariamente, fazer parte de qualquer árvore de Steiner. Comprimida a aresta, esta se torna, neste caso, um vértice terminal do grafo reduzido.

Teste 3.3.4 (LC) *A aresta* $(i, j) \in E$ *pode ser eliminada, se existe um caminho* $i - j$ *não trivial com distância* $d_{ij} \leq c_{ij}$.

A justificativa para a validade do teste é óbvia.

Teste 3.3.5 (SD) *Seja* (u, v) *uma aresta em* E. *Se* $B(u, v) < c(u, v)$, *então* (u, v) *é redundante.*

A justificativa para a validade do teste é dada pelo Teorema (3.3.1) abaixo.

Teste 3.3.6 (SDE) *Seja* (u, v) *uma aresta em* E. *Se* $B(u, v)^{-(u,v)} \leq c(u, v)$, *então* (u, v) *é redundante.*

A justificativa para a validade do teste é dada pelo Teorema (3.3.1) abaixo.

Teorema 3.3.1 (Duin [10]) *Os testes SD e SDE são válidos.*

Prova. Suponha que $B(u, v)^{-(u,v)} \leq c(u, v)$. Nesse caso, se $B(u, v) \leq c(u, v)$, então, $B(u, v)^{-(u,v)} = B(u, v)$. Seja $P \in \mathcal{P}(u, v), (u, v) \notin P$, um caminho tal que $SD(P) = B(u, v)^{-(u,v)}$ e seja $T(P) = \{u, v\} \cup (T \cap P)$. Suponha

que uma árvore de Steiner R use a aresta (u, v). Removendo (u, v) de R, obtemos duas sub-árvores; a que contém o vértice u será chamada de R_u e a que contém v, de R_v. Escolha dois vértices k e l de $T(P)$, consecutivos em P, tal que $k \in R_u$ e $l \in R_v$. Seja $P(k, l)$ o subcaminho em P indo de k até l. Como $c(P(k, l)) \leq B(u, v)^{-(u,v)} \leq c(u, v)$, $R' = R_u \cup R_v \cup P(k, l)$ é uma árvore de Steiner que não contém (u, v) e tal que $c(R') \leq c(R)$.

3.3.2 Procedimento de pré-processamento

Ao aplicar os testes de pré-processamento, devemos tomar um cuidado especial para evitar a geração de uma mesma árvore de Steiner mais de uma vez. Essa questão é, na realidade, recorrente, ao longo de todo o algoritmo proposto para a solução do PEAS. Nos PAs, resolvidos por programação inteira, ou seja, (2.17)-(2.21), utilizamos os cortes (2.22) e perturbações aleatórias na função objetivo para atacar o problema. Da mesma forma, nos problemas resolvidos por aproximação, também usamos perturbações aleatórias na função objetivo.

Em outros trabalhos sobre pré-processamento, a questão de não se gerar uma árvore de Steiner mais de uma vez não é levada em consideração. Isso se deve ao fato de que, nessas aplicações, o interesse é o de simplesmente encontrar uma ASCM. Para o nosso caso, construímos uma estrutura de dados, onde são armazenadas todas as árvores de Steiner já previamente geradas para um conjunto de vértices terminais T_i. Dessa forma, podemos assegurar que o grafo G', que é o grafo original G reduzido por pré-processamento, possui uma ASCM diferente das geradas, se e somente se G também a possui. Para facilitar o entendimento, apresentamos um exemplo a seguir. Dado o grafo G, representado na figura (3.3), suponha que já geramos uma árvore de Steiner para o conjunto de vértice terminais $T_1 = \{1, 5, 9\}$, contendo as

arestas $S_1 = \{1, 4, 7, 10\}$. Suponha, ainda, que em um dado momento na aplicação do algoritmo, devemos gerar outra árvore de Steiner para T_1, com os custos das arestas sendo iguais a $c = \{2, 4, 4, 3, 5, 3, 2, 6, 2, 1, 3, 1\}$. É facil ver que existem duas ASCM com custo igual a 8, onde uma é a árvore já gerada, S_1, e a outra é $S_2 = \{1, 4, 9, 12\}$. Dependendo da ordem de aplicação dos testes de redução, poderíamos, eventualmente, reduzir o grafo G de tal maneira que a árvore de Steiner S_1 estaria incluída no grafo reduzido G' e a árvore S_2, excluída do mesmo. No nosso caso, esse pré-processamento não seria válido, pois excluiria de G' uma árvore de Steiner ótima, alternativa.

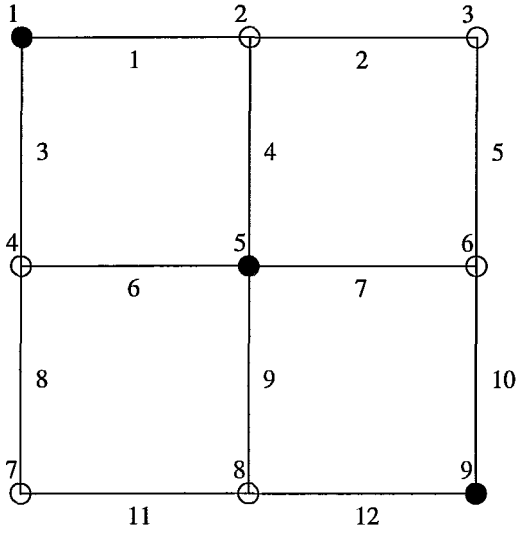

Figura 3.3: Grafo G.

Os testes NTD1, NTD2 e TD1 não são afetados pelo problema descrito acima, ao contrário dos testes LC, SD e SDE. Esses últimos são baseados em alternativas: "se condição, para qualquer árvore de Steiner R que use e, existe outra árvore de Steiner R' que não usa e e tal que $c(R') \leq c(R)$". Sendo testes de alternativas, só podemos eliminar uma aresta $e \in E$, se existir um conjunto de arestas P capazes de efetuar a conexão dos vértices terminais de G a um mesmo custo da aresta e. Com esse conjunto, podemos verificar

se as arestas que restam depois do pré-processamento não vão permitir que uma árvore de Steiner diferente das já construídas seja formada.

A ordem em que os testes são aplicados é importante para o resultado do pré-processamento. Então, pensando nas características do problema, decidimos por aplicar a ordem que se segue (Ilustrado na Figura 3.4).

Algoritmo 6 Procedimento de pré-processamento

1: Aplicar os testes SD, SDE e LC nas arestas com valor diferente de zero na função objetivo, antes de aplicar as perturbações, e depois aplicar o passo (5) nos vértices das arestas removidas.
2: Aplicar o passo (5) nos quatro vértices diagonais no grafo grade.
3: Aplicar o passo (5) em todo grafo restante.
4: Parar.
5: Aplicar os testes NTD1, NTD2 e LC e continuar aplicando esses testes recursivamente nas arestas eliminadas.

Figura 3.4: Pré-processamento para o PAS

O passo (1) do algoritmo (6), na figura (3.4), é intuitivo, pois se eliminarmos as arestas que possuem valor diferente de zero na função objetivo, temos certeza que qualquer árvore encontrada terá custo reduzido negativo, bastando para tanto que a constante ν_i, na função objetivo (2.17), seja menor que zero. Além disso, normalmente, essas são arestas fáceis de substituir. O passo (2) é, então, aplicado, pois os vértices das diagonais possuem grau dois e podem ser facilmente eliminados. Com isso, se inicializam "reações em cadeia", com a eliminação de arestas pelas extremidades do grafo. Finalmente, o passo (3) tem por objetivo eliminar arestas subótimas que não tenham sido eliminadas nos passos anteriores.

Capítulo 4

Um algoritmo exato para o PEAS

Neste capítulo, apresentaremos nosso algoritmo exato para resolução do PEAS. Esse algoritmo se fundamenta no material descrito nos capítulos 2 e 3. Como explicado anteriormente, pelas características do problema aqui estudado, optamos por utilizar geração de colunas para sua solução. Como o problema é de programação inteira, só é possível garantir a otimalidade da solução com a utilização do método *branch-and-price.*

4.1 Uma visão geral do algoritmo

Em nossa implementação do algoritmo *branch-and-price*, em cada nó da árvore de procura recorremos à geração de colunas para resolver a relaxação linear do problema de PI correspondente ao nó. Tipicamente, esse procedimento consome um tempo de processamento considerável. É então crucial evitar, tanto quanto possível, que a árvore de busca do método *branch-and-price* cresça em demasia. Com o intuito de ser capaz de provar a otimalidade

rapidamente, foram adotados vários procedimentos para acelerar o processo de procura.

Como é usual nessas aplicações, a inicialização do algoritmo é crucial para um bom desempenho do mesmo. Em nosso caso, a heurística de inicialização introduzida na seção 3.2 se mostrou capaz de acelerar o processo de convergência para uma solução ótima, ao retornar limites superiores de muito boa qualidade. Em muitas aplicações, através da utilização da heurística inicial, obtemos um limite superior para o problema, cujo valor está a menos de uma unidade do limite inferior obtido pela relaxação linear de (2.11)-(2.14). Dado que o valor de uma solução ótima é garantidamente um número inteiro, a otimalidade do limite superior estaria provada, nessas situações.

Depois de gerar as colunas iniciais para o PMR, através da heurística proposta para o PEAS, podemos iniciar a aplicação do método de geração de colunas. Em nossa aplicação, o método de geração de colunas resolve, em cada iteração, n subproblemas. Cada um desses subproblemas modela um PAS, e pode ser resolvido rapidamente, sem garantias de otimalidade, através de heurísticas, como descrito na seção (3.1). Aplicada a heurística, caso a solução obtida já tenha sido gerada anteriormente, ou seu custo seja não negativo, resolvemos de maneira exata o PA correspondente, (2.17)-(2.21). Como regra geral, utilizamos esse procedimento de resolver, através de heurísticas, os PAs, só recorrendo ao algoritmo exato quando a árvore gerada pela heurística não pode ser adicionada ao PMR. Como explicado anteriormente, terminada a fase de geração de colunas, obtemos um limite inferior válido para o PEAS, por relaxação linear. Com o limite inferior e o limite superior disponíveis, temos a chance de provar a otimalidade do limite superior (caso a diferença entre os limites seja de menos de uma unidade). Esses procedimentos são usados nos demais nós da árvore de procura. A

única diferença entre o nó zero e os demais é a utilização dos cortes (2.23) nos PAs e a fixação de valores imposta pela regra de ramificação utilizada no *branch-and-price*. Um fluxograma do algoritmo pode ser visto na figura (4.1).

Uma vez obtida uma coluna com custo reduzido negativo, após a resolução de um PA, caso essa coluna não tenha sido ainda gerada, temos que adicionar a desigualdade (2.22) correspondente a (2.17)-(2.21), para evitar a geração futura da mesma coluna. Como dito anteriormente, sempre que a heurística para o PAS não é capaz de gerar uma coluna com custo reduzido negativo, recorremos ao algoritmo de solução exata. Nesse caso, antes da aplicação do algoritmo exato, pré-processamos a instância, como uma forma de reduzir o tempo total de processamento. Obviamente, os testes de pré-processamento poderiam ser também aplicados antes de recorrermos à heurística. No entanto, em nossos experimentos computacionais, observamos ser mais atraente investir tempo de CPU em pré-processamento apenas antes de aplicar o algoritmo de solução exata.

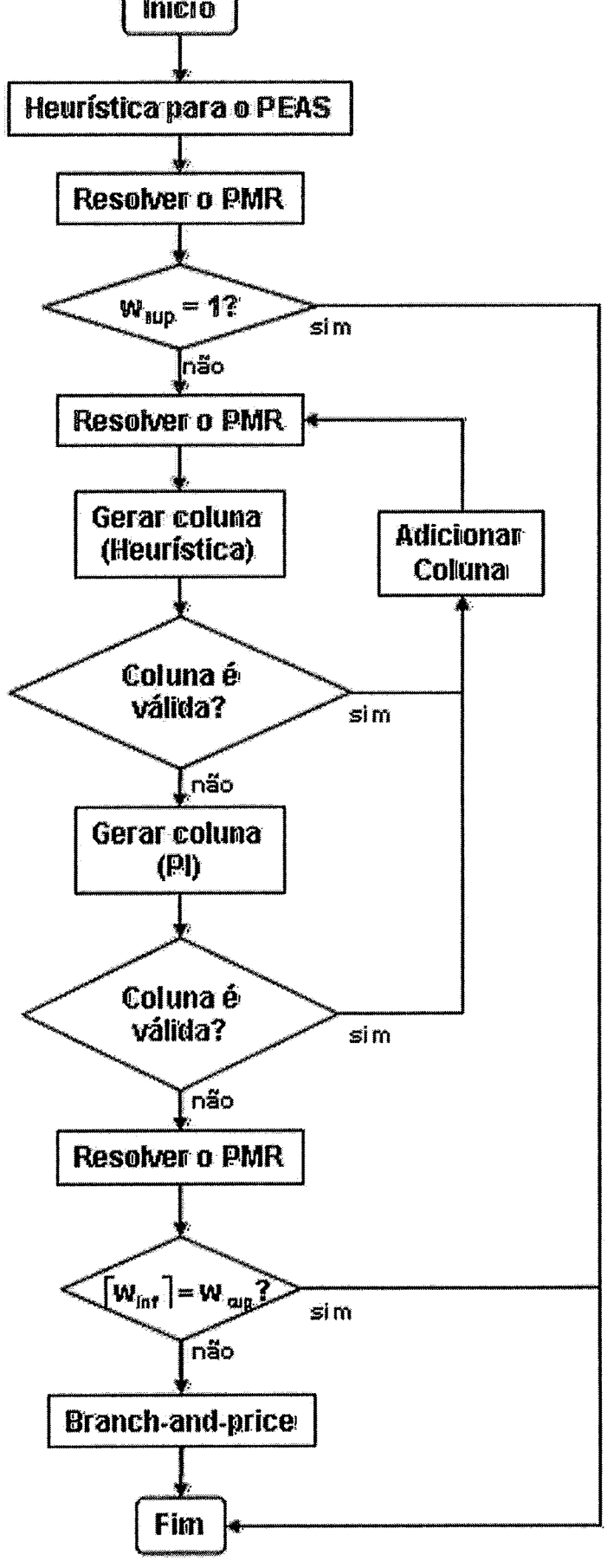

Figura 4.1: Algoritmo usado para resolver o PEAS.

Capítulo 5

Resultados Computacionais

Neste capítulo, apresentaremos um resumo dos resultados obtidos. Visando a facilitar a avaliação da qualidade desses resultados, todos os programas foram executados em máquinas **AMD Athlon XP 1800, 1.53GHz**, com **1GB** de memória **RAM**. Além disso, todos os códigos foram implementados na linguagem de programação **C++**, e compiladas pelo GNU **gcc**, versão **2.95**, para **Linux**. Foi utilizado o pacote de programação inteira mista XPRESS [23] para a resolução exata de problemas de árvores de Steiner.

5.1 Problemas testes

Para testar nosso algoritmo de resolução do PEAS, criamos 3 classes distintas de instâncias: cl_1, cl_2 e cl_3. Essas classes estão associadas a grafos grade completos, com características distintas para cada classe. Denote por L o número de retas paralelas ao eixo horizontal, que dão origem ao grafo grade. De maneira similar, denote por A o número de retas paralelas ao eixo horizontal, para o mesmo grafo. Sendo assim, para os grafos grade completos, o número de vértices é igual a $A \times L$ e o número de arestas é igual

a $2(A \times L) - A - L$.

A classe cl_1 é composta por instâncias de pequeno porte, construídas manualmente, de forma a testar um dado ponto específico do algoritmo e o seu desempenho. Essas instâncias foram construídas durante o processo de criação do algoritmo, onde cada uma testa um ponto diferente dos assuntos envolvidos no algoritmo. A classe cl_2 é constituída por instâncias geradas de forma similar àquelas utilizadas na literatura para testar algoritmos para resolução do PEAS, onde as árvores de Steiner encontradas devem ser, necessariamente, disjuntas em relação às arestas. Entretanto, alteramos a distribuição dos vértices terminais, sorteando de forma aleatória cada vértice terminal, para tentar impedir que a solução correspondente aos PEAS aqui tratados seja sempre igual a 1, ou seja, tenham sempre arestas disjuntas. Mais informações sobre a forma como são geradas essas instâncias podem ser obtidas em [14, 16, 20, 3, 5]. Finalmente, a classe cl_3 é constituída por instâncias onde o número de vértices terminais e suas distribuições no grafo são escolhidos aleatoriamente. A construção dessas instâncias é feita da seguinte forma: dados A, L e $|\mathcal{T}|$, sorteamos de forma aleatória para cada um dos $|\mathcal{T}|$ conjuntos de vértices terminais a cardinalidade do conjunto. Em seguida, para cada conjunto de vértices terminais, sorteamos de forma aleatória os vértices pertencentes ao conjunto. Os dados relativos às instâncias de cada classe podem ser vistos nas Tabelas 5.1, 5.2 e 5.3, referentes às classes cl_1, cl_2 e cl_3, respectivamente. A primeira coluna das tabelas descreve os nomes das instâncias. O número de retas paralelas ao eixo horizontal e o número de retas paralelas ao eixo vertical, associadas a cada instância gerada, são dadas nas colunas 2 e 3. A quarta coluna descreve o número de conjuntos de vértices terminais. As demais colunas descrevem as cardinalidades dos conjuntos de vértices terminais de cada instância.

Nome	A	L	$\lvert\mathcal{T}\rvert$	Distr. das redes				
				2	3	4	5	6
prob_1	3	3	2	0	2	0	0	0
prob_2	6	5	4	0	1	3	0	0
prob_3	6	6	7	1	2	0	3	1
prob_4	5	5	3	0	1	2	0	0
prob_5	4	10	12	2	9	1	0	0

Tabela 5.1: Dados dos problemas testes da classe cl_1

Nome	A	L	$\lvert\mathcal{T}\rvert$	Distr. das redes				
				2	3	4	5	6
Difícil	15	23	24	15	3	4	1	1
Difícil+	15	22	24	15	3	5	0	1
Term. int.	16	23	24	8	7	5	4	0
Denso	17	15	19	3	11	5	0	0
Denso+	18	16	19	3	11	5	0	0
Denso++	17	16	19	3	11	5	0	0
Pedagógico	16	15	22	14	4	4	0	0

Tabela 5.2: Dados dos problemas testes da classe cl_2

Nome	A	L	$\lvert\mathcal{T}\rvert$	Distr. das redes						
				2	3	4	5	6	9	18
prob_a	5	15	20	1	11	8	0	0	0	0
prob_b	10	10	12	2	9	1	0	0	0	0
prob_c	10	15	40	8	24	8	0	0	0	0
prob_d	12	15	15	0	4	3	0	8	0	0
prob_e	20	30	10	2	1	3	1	0	2	1
prob_f	30	40	20	3	9	3	3	2	0	0
prob_g	40	40	3	0	0	0	2	1	0	0

Tabela 5.3: Dados dos problemas testes da classe cl_3

5.2 Resultados da heurística para o PEAS

No início do nosso algoritmo, rodamos a heurística para o PEAS e obtivemos um limite superior, que em várias instâncias foi o suficiente para provar a otimalidade. Os resultados obtidos estão descritos na tabela 5.4. O número de colunas válidas geradas pela heurística aparece na coluna 2, e o número de colunas geradas em duplicatas aparece na coluna 3. A coluna 4 e 5 indicam o limite superior para o PEAS obtido pela heurística e o valor da solução ótima correspondente. Finalmente, na sexta coluna, apresentamos o tempo de CPU, em segundos, gasto na execução da heurística.

Os parâmetros n_1, n_2 e n_3 utilizados para as instâncias (vide figura 3.2) foram 2, 2 e 10, respectivamente, exceto para as intâncias da classe cl_1, onde foram utilizadas os valores 1, 1 e zero, devido às dimensões reduzidas dessas instâncias. Como observado, soluções ótimas para 15 das 19 instâncias testadas foram obtidas.

Inicialmente, utilizamos os parâmetros utilizados para a classe cl_1 para as demais intâncias. No entanto, isso levava a um número excessivo de iterações

do método de geração de colunas, e, em algumas instâncias, foi necessário utilizar o método *branch-and-price* para provar a otimalidade da solução. Com a mudança dos parâmetros, o número de iterações e o número de instâncias que utilizaram o método *branch-and-price* caiu drasticamente.

5.3 Resultados do algoritmo exato para o PEAS

Os resultados obtidos para resolver as instâncias do PEAS (incluindos todas as etapas do algoritmo de solução proposto) estão descritos na tabela 5.5, onde a segunda coluna indica o número de iterações necessárias para obter a relaxação linear do modelo pelo método de geração de colunas. Note que em 9 instâncias, esse número de iterações foi igual a 1, o que significa que apenas a aplicação da heurística primal foi suficiente para gerar todas as colunas presentes na relaxação linear. A terceira coluna fornece o número de nós da árvore de procura, necessários para provar a otimalidade. A quarta coluna indica o valor da relaxação linear obtida para cada instância, com a aplicação do método de geração de colunas. Finalmente, a quinta coluna apresenta o valor da solução ótima.

A heurística primal proposta para o PEAS apresentou um ótimo desempenho. Nove instâncias foram resolvidas unicamente através da heurística. Para aquelas instâncias onde a segunda e a terceira colunas possuem valor 1, isso indica que não foi necessário gerar colunas adicionais (além daquelas geradas pela heurística) para a resolução exata do PMR (usando programação inteira). Foi possível provar a otimalidade. Outro resultado a ressaltar foi aquele obtido com a relaxação linear da formulação aqui proposta para o PEAS, onde para apenas uma instância testada não obtivemos uma diferença menor do que uma unidade entre o valor da relaxação linear

Nome	Col. válidas	Col. eliminadas	w_{sup}	w^*	Tempo (s)
prob_1	4	0	1	1	0
prob_2	15	1	2	2	0.1
prob_3	46	3	2	2	0.1
prob_4	9	0	1	1	0.1
prob_5	115	29	3	3	0.1
Difícil	2299	245	2	2	101.7
Difícil+	2058	486	2	1	193
Term. int.	2432	112	2	2	207
Denso	1568	66	2	2	452
Denso+	1577	57	3	2	277
Denso++	1576	58	2	2	201
Pedagógico	1734	422	2	2	7.2
prob_a	1440	360	4	4	1.8
prob_b	569	127	2	2	2.3
prob_c	5973	827	4	4	230
prob_d	1029	21	2	2	39.8
prob_e	489	11	2	1	10.7
prob_f	1779	21	2	1	1050
prob_g	65	1	1	1	0.4

Tabela 5.4: Resultados da heurística para o PEAS

e o valor inteiro ótimo. Dificuldades associadas à utilização das regras de ramificação propostas para o método *branch-and-price* podem ser observadas nos 3 resultados onde foi necessário ramificar. Os tempos de CPU elevados, comparados aos tempos de CPU relativos apenas à aplicação da heurística, mesmo para aquelas instâncias onde apenas a heurística foi o suficiente para resolver o problema, se devem, principalmente, à resolução por programação linear inteira dos PRMs. A formulação exata do PAS, (2.17)-(2.21), quando utilizada, também exigiu um tempo de CPU elevado, principalmente quando comparado ao tempo requerido pela aplicação da heurística de Takahashi e Matsuyama.

Nome	Iterações	Nós	w_{inf}	w^*	Tempo (s)
prob_1	1	1	1	1	0
prob_2	11	13	0.8974	2	2
prob_3	37	1	1.4222	2	17
prob_4	42	1	0.7541	1	1
prob_5	1	1	3	3	1
Difícil	45	205	1.0089	2	9601
Difícil+	1	1	1.001	2	2823
Term. int.	3	1	1.3125	2	7601
Denso	1	1	1.1333	2	4590
Denso+	37	1	1.0399	2	3994
Denso++	1	1	1.0556	2	3005
Pedagógico	1	1	1.1333	2	806
prob_a	1	1	3.2	4	5.9
prob_b	1	1	1.44	2	3.76
prob_c	1	1	3.2	4	2941
prob_d	10	1	1.2563	2	2985
prob_e	240	11	0.6099	1	23500
prob_f	254	1	0.6525	1	39701
prob_g	3	1	0.3333	1	345

Tabela 5.5: Resultados do algoritmo exato para o PEAS

Capítulo 6

Conclusões

Neste trabalho foi proposto um algoritmo exato para a resolução do PEAS, que utiliza geração de colunas e o método *branch-and-price*. Introduzimos, também, uma heurística para o PEAS, que se mostrou bastante eficaz. O problema abordado nessa tese tem se mostrado, na prática, muito difícil de se resolver. Apesar de sua grande importância industrial, tem sido tratado apenas por heurísticas ou, de maneira exata, em versões mais simples (i.e., soluções com arestas disjuntas). Os resultados aqui obtidos definem o "estado da arte" para a resolução exata do problema.

O principal objetivo desse trabalho foi demonstrar que a formulação aqui proposta para a variante do PEAS que consideramos fornece um excelente limite inferior para o problema. Apesar disso, ainda não é "possível" resolver de forma exata problemas reais de grande porte, onde as dimensões envolvidas são excessivas.

Uma sugestão para trabalhos futuros seria utilizar um algoritmo semelhante ao aqui proposto, para resolver outras versões do PEAS. Como exemplo, a versão do PEAS aqui estudado que inclua a restrição de que o empacotamento a ser gerado tenha o menor custo possível.

Outra sugestão seria tentar tornar mais eficiênte o algoritmo utilizado para a resolução do PMR, em particular, a parte que envolve a utilização do pacote XPRESS [23]. Outra sugestão para tentar melhorar o desempenho do algoritmo seria a utilização de algoritmos mais eficientes, recentemente propostos, para a solução exata do PAS. Finalmente, a última sugestão seria, possivelmente, considerar outras regras de ramificação.

Referências Bibliográficas

[1] ARAGÃO, M. P., WERNECK, R. F. "On the implementation of MST-based heuristics for the Steiner problem in graphs". *Lecture notes in computer science*, v. 2409, pp. 1–15, 2002.

[2] BARNHART, C, JOHNSON, E. L., NEMHAUSER, G. L., SAVELSBERGH, M. W. P., VANCE, P. H. "Branch-and-price: Column generation for solving huge integer programs". *Operations Research*, v. 46, n. 3, pp. 316–329, 1998.

[3] BURSTEIN, M., PELAVIN, R. "Hierarchical wire routing". *IEEE Transactions on Computer-Aided-Design*, v. 2, pp. 223–234, 1993.

[4] CABRAL, L. A. F. *"Paralelizando a fase de roteamento de circuitos baseados em FPGAs"*. PhD thesis, Universidade Federal do Rio de Janeiro, 2001.

[5] COHOON, J. P., HECK, P. L. "BEAVER: A computational-geometry-based tool for switchbox routing". *IEEE Transactions on Computer-Aided-Design*, v. 7, pp. 684–697, 1998.

[6] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., STEIN, C. *"Introduction to Algorithms"*. MIT Press, 2001.

[7] DANTZIG, G. B., WOLFE, P. "Decomposition principle for linear programs". *Operations Research*, v. 8, pp. 101–111, 1960.

[8] DESROSIERS, J., SOUMIS, F., DESROCHERS, M. "Routing with time windows by column generation". *Networks*, v. 14, pp. 545–565, 1984.

[9] DIJSTRA, E. W. "A note on two problems in connexion with graphs". *Numerische Mathematik*, v. 1, pp. 269–271, 1959.

[10] DUIN, C., SMITH, J. M., RUBINSTEIN, J. H. *"Advances in Steiner trees"*, v. 6, *Combinatorial Optmization*, chapter Preprocessing the Steiner problem in graphs. Kluwer, 2000.

[11] FORD, L. R., FULKERSON, D. R. "A suggested computation for maximal multicommodity network flows". *Management Science*, v. 5, pp. 97–101, 1958.

[12] GILMORE, P. C., GOMORY, R. E. "A linear programming approach to the cutting stock problem". *Operations Research*, v. 9, pp. 849–859, 1961.

[13] GILMORE, P. C., GOMORY, R. E. "A linear programming approach to the cutting stock problem, part II". *Operations Research*, v. 11, pp. 863–888, 1963.

[14] GRÖTSCHEL, M., MARTIN, A., WEISMANTEL, R. "Packing Steiner trees: a cutting plane algorithm and computational results". *Mathematical programming*, v. 72, pp. 125–145, 1996.

[15] GRÖTSCHEL, M., MARTIN, A., WEISMANTEL, R. "Packing Steiner trees: polyhedral investigations". *Mathematical programming*, v. 72, pp. 101–123, 1996.

[16] JEONG, G., LEE, K., PARK, S., PARK, K. "A branch-and-price algorithm for the Steiner tree packing problem". *Computers & Operations Research*, v. 29, pp. 221–241, 2002.

[17] KARP, R. *"Reducibility among combinatorial problems"*, pp. 85–103. Complexity of Computer Computations. Plenum Press, New York, 1972.

[18] LENGAUER, T. *"Combinatorial algorithms for integrated circuit layout"*. Wiley & Sons, New York, 1990.

[19] LUCENA, A., BEASLEY, J. "A branch and cut algorithm for the Steiner problem in graphs". *Networks*, v. 31, pp. 39–59, 1998.

[20] LUK, W. K. "A greedy switchbox router". *Integration*, v. 3, pp. 129–149, 1985.

[21] MACULAN, N. "The Steiner problem in graphs". *Annals of Discrete Mathematics*, v. 31, pp. 185–212, 1987.

[22] NEMHAUSER, G. L., WOLSEY, L. A. *"Integer and Combinatorial Optimization"*. Wiley & Sons, New York, 1988.

[23] OPTIMIZATION, DASH. *"XPRESS-MP Optimiser Subroutine Library 12"*, 2000.

[24] PRIM, R. C. "Shortest connection networks and some generalizations". *Bell System Technical Journal*, v. 36, n. 6, pp. 1389–1401, 1957.

[25] SAVELSBERGH, M. W. P. *"A branch and price algorithm for the generallized assignment problem"*. Computational optimization center coc-93-02, Georgia Institute of Technology, Atlanta, 1993.

[26] TAKAHASHI, H., MATSUYAMA, A. "An approximate solution for the Steiner problem in graphs". *Mathematica Japonica*, v. 24(6), pp. 573–577, 1980.

[27] UCHOA, E. *"Algoritmos para problemas de Steiner com aplicações em projeto de circuitos VLSI"*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2001.

[28] UCHOA, E., ARAGÃO, M. P., RIBEIRO, C. C. "Prepocessing Steiner problems from VLSI layout". *Networks*, v. 40, pp. 38–50, 2002.

[29] VANDERBECK, F. *"Decomposition and Column Generation for Integer Programs"*. PhD thesis, Université Catholique de Louvain, 1994.

[30] Vanderbeck, F., WOLSEY, L. A. "An exact algorithm for IP column generation". *Operations Research Letters*, v. 19, n. 4, pp. 151–159, 1996.

[31] WOLSEY, L. A. *"Integer programming"*. Wiley & Sons, New York, 1998.