Problema das pseudo-arborescências capacitadas com localização de facilidades

Fabio Henrique Noboru Abe

Problema das pseudo-arborescências capacitadas com localização de facilidades

Fabio Henrique Noboru Abe

Orientadora: *Profª. Drª. Edna Ayako Hoshino*

Dissertação apresentada como requisito para obtenção do grau de Mestre em Ciência da Computação no programa de Pós Graduação em Ciência da Computação, da Universidade Federal de Mato Grosso do Sul.

UFMS - Campo Grande
agosto/2016

À Deus em primeiro lugar,

À minha esposa,
Ana Karolyne Abe.

Agradecimentos

Agradeço a Deus por sempre se fazer presente na minha vida, dando-me força e coragem para continuar neste desafio.

Agradeço à minha esposa Karol pelo amor, paciência, dedicação, companheirismo e compreensão que recebo todos os dias.

Aos meus pais Dirce e Pedro agradeço por sempre me apoiarem e me incentivarem a estudar.

À minha orientadora Edna agradeço pela paciência, pela compreensão e pelas horas dedicadas a me ensinar os conteúdos da área.

Aos meus amigos Lucas Rodrigues e Cleison Marin, pela amizade nas horas difíceis do curso e pelas risadas que fizeram da kitnet um "posto avançado" da minha casa.

Aos meus colegas de trabalho Beth e Giovanni, por segurarem as pontas durante a minha ausência.

Aos professores, técnicos e colegas da FACOM por fazerem parte desta etapa da minha vida.

Ao pesquisador Alessandro Hill, pelas sugestões e conselhos trocados por e-mail.

À FUNDECT pelo apoio financeiro.

Abstract

In this work, we present the capacitated pseudo arborescences with facility location problem, which is a new problem related to two classical problems: the capacitated vehicle routing problem and the capacitated facility location problem. The proposed problem generalizes the capacitated ring tree problem.

We propose in this study two integer programming formulations to model the problem. The first one is an extended formulation based on set partition and the second one is a compact formulation based on flow models. We also propose two exact algorithms to solve the problem. One of them uses the branch-and-price algorithm with the extended formulation and the other one is a branch-and-bound algorithm based on the compact formulation. We also implemented a primal heuristic and a pricing heuristic aiming to improve the performance of exact methods. Computational experiments shown that the extended formulation provides tighter bounds than the compact formulation. Additionally, we observed that the heuristics were relevant to accelerate the branch-and-price and the branch-and-bound algorithms, mainly the primal heuristic.

Keywords: capacitated pseudo arborescence, facility location, combinatorial optimization, integer linear programming, heuristic.

Resumo

Neste trabalho apresentamos o problema das pseudo-arborescências capacitadas com localização de facilidades, um problema novo e relacionado a dois outros problemas clássicos: o problema do roteamento de veículos capacitado e o problema da localização de facilidade capacitada. O problema estudado é uma generalização do problema da pseudo-arborescência capacitada, em inglês *capacitated ring tree problem*.

Propomos neste estudo duas formulações em programação linear inteira para modelar o problema. A primeira é uma formulação estendida baseada em partição de conjuntos e a segunda é uma formulação compacta baseada em fluxos. Propomos também dois algoritmos exatos para resolver o problema. Um deles utiliza a técnica de *branch-and-price* com a formulação estendida e o outro é um algoritmo do tipo *branch-and-bound* baseado na formulação compacta. Implementamos também uma heurística primal e uma heurística de *pricing* com o objetivo de melhorar o desempenho dos métodos exatos. Experimentos computacionais realizados em um grupo de instâncias testes mostram que a formulação estendida fornece limitantes muito mais apertados do que a formulação compacta. Além disso, as heurísticas foram relevantes para acelerar os métodos de *branch-and-price* e *branch-and-bound*, em especial a heurística primal.

Palavras-chave: pseudo-arborescência capacitada, localização de facilidade, otimização combinatória, programação linear inteira, heurística.

Sumário

Sumário xiv
Lista de Figuras xv
Lista de Tabelas xvii
Lista de Algoritmos xix

1 Introdução **1**

2 Trabalhos Relacionados **7**

3 Metodologia **11**
3.1 Problemas de Otimização Combinatória 11
3.2 Programação Linear Inteira 12
3.2.1 Programação Linear 13
3.2.2 Otimalidade, Relaxações e Limitantes 16
3.2.3 Branch-and-Bound 18
3.2.4 Método de Geração de Colunas 20
3.2.5 Branch-and-Price 22
3.3 Heurísticas e Metaheurísticas 23
3.3.1 GRASP 23

4 Trabalho Desenvolvido **27**
4.1 Modelos Matemáticos para o CPAFLP 27
4.1.1 Formulação estendida para o CPAFLP 27
4.1.2 Formulação compacta para o CPAFLP 32
4.2 Heurísticas Propostas 34
4.2.1 Heurística Primal 34
4.2.2 Heurística de *Pricing* 40

5 Resultados **49**
5.1 Instâncias de Testes 49
5.2 Definição dos Parâmetros 51

5.3 Resultados dos Testes 54
5.4 Considerações sobre os Resultados 68

6 Conclusões **71**

Referências **76**

Lista de Figuras

1.1 Exemplos de pseudo-arborescências. 3
1.2 Exemplo de solução para o CPAFLP ($|U_2|$ = 5, $|U_1|$ = 16, $|W|$ = 2, Q = 9, Q_t = 4, m = 3). 5

3.1 Par Primal e Dual em um problema de Minimização. 15
3.2 Problema Mestre Restrito e *Pricing*. 22

4.1 Exemplos de inserção de vértices. 37
4.2 Níveis da Heurística de *Pricing*. 41
4.3 Atualização do impacto dos candidatos. 43
4.4 Busca local *lsearch*. 43
4.5 Inserção de arborescências. 45
4.6 Grafo com 9 vértices e 3 arborescências. 48

5.1 Solução da instância R_075_A03 do grupo CF0_QT66. 68

Lista de Tabelas

5.1 Nomenclatura e configuração das instâncias utilizadas. 51
5.2 Conjunto de instâncias utilizadas para escolha do α. 52
5.3 Melhor resultado da heurística primal em cada instância. 53
5.4 Desempenho do α em 1000 iterações. 53
5.5 Características dos programas. 54
5.6 *GAP* do LB no nó raiz das formulações (SPF) e (MCF). 56
5.7 Resultados do grupo de instâncias CF0_QT1. 58
5.8 Resultados do grupo de instâncias CF0_QT66. 59
5.9 Resultados do grupo de instâncias CFMED_QT1. 60
5.10 Resultados do grupo de instâncias CFMED_QT66. 61
5.11 Resumo dos resultados do grupo de instâncias CF0_QT1. 62
5.12 Resumo dos resultados do grupo de instâncias CF0_QT66. 62
5.13 Resumo dos resultados do grupo de instâncias CFMED_QT1. . . 62
5.14 Resumo dos resultados do grupo de instâncias CFMED_QT66. . . 63
5.15 Desempenho SPF- × SPF. 63
5.16 Desempenho MCF- × MCF. 64
5.17 Desempenho SPF × SPF*. 65
5.18 Desempenho MCF × SPF. 65
5.19 Solução inicial. 66
5.20 Origem das pseudo-arborescências. 67

Lista de Algoritmos

1 Passos do algoritmo *Simplex*. 16
2 Passos de um Algoritmo de *Branch-and-Bound*. 19
3 Método da geração de colunas. 21
4 Bloco principal GRASP. 24
5 Função *Construcao_Gulosa_Aleatorizada*(). 25
6 Função *Busca_Local*(). 25

7 Heurística Primal. 35
8 Função *define_tipo*. 36
9 Função *Cobre_U*$_2$. 37
10 Função *Cobre_U*$_1$. 38
11 Função *Cobre_W*. 39
12 Heurística de *Pricing*. 42
13 Função *heur_arbo*. 44
14 Função *GeraArb*. 46

CAPÍTULO

1

Introdução

O problema das pseudo-arborescências capacitadas com localização de facilidades, em inglês *capacitated pseudo arborescences with facility location problem (CPAFLP)*, é um problema novo na literatura e foi sugerido por Alessandro Hill como um tema de pesquisa derivado do trabalho apresentado em Hill (2012).

O CPAFLP faz alusão a problemas reais envolvendo a entrega de bens e serviços. Por simplificação, chamaremos bens e serviços de produtos. Assim sendo, uma instância do problema é inicialmente composta por:

1. Um depósito central, simbolizado por d, de onde iniciam todas as entregas de produtos;
2. Um conjunto composto por dois tipos de clientes, tipo 1 e tipo 2 respectivamente denotados por U_1 e U_2, que possuem demanda por produtos;
3. Um conjunto de pontos opcionais, denotados por W, que não possuem demanda por produtos, mas podem ser utilizados nas entregas quando for conveniente para a diminuição dos custos.

Os clientes são diferenciados pelo nível de exigência na entrega dos produtos, sendo que os clientes do tipo 2 são prioritários. Para tratamento diferenciado dos clientes, são utilizadas diferentes formas de entrega do produto.

Para facilitar o entendimento das diferentes formas de entrega e definir formalmente o problema, utilizaremos alguns conceitos da Teoria de Grafos. Uma instância do CPAFLP corresponde a um grafo orientado simples, $G = (V,A)$, composto por um conjunto de arcos (denotado por $A(G)$) e um

conjunto finito de vértices (denotado por $V(G)$), em que V é constituído pelos conjuntos de clientes do tipo 1 $\{U_1\}$, clientes do tipo 2 $\{U_2\}$, pontos opcionais $\{W\}$ e um único depósito $\{d\}$.

Os pontos opcionais, são vértices do grafo que não possuem demanda por entrega de produtos e, portanto, podem ou não fazer parte da solução. Tais pontos são comumente denominados, na literatura, como **vértices de Steiner**.

Um arco que sai de um vértice i e chega em um vértice j é denotado por (i, j) ou, ainda, por ij. Os vértices i e j de um arco (i, j) são chamados **extremidades** do arco. O grau de um vértice i, denotado por $d(i)$, diz respeito à quantidade de arcos que chegam ou saem do vértice i. O **grau de entrada** de um vértice i, denotado por $d^-(i)$ é a quantidade de arcos que chegam no vértice i, e o **grau de saída** $d^+(i)$ é a quantidade de arcos que saem do vértice i. Denotaremos os arcos que chegam no vértice i de $\delta^-(i)$ e os arcos que se originam em i de $\delta^+(i)$. Dado um grafo orientado $G = (V,A)$, se S é um subconjunto de arcos de A, o conjunto dos **vértices induzidos** por S, denotado por $V[S]$, corresponde a $\{v \in V : v$ é extremidade de algum arco $(i, j) \in S\}$. Dado um grafo $G = (V,A)$ dizemos que $H = (V_H, A_H)$ é um **subgrafo** de G se $V_H \subseteq V$ e $A_H \subseteq A$. Dado um grafo $G = (V,A)$, se $S \subseteq V(G)$, o **subgrafo induzido** por S, denotado por $G[S]$, é um grafo H tal que $V(H) = S$ e $E(H) = \{(i, j) \in E(G) : i, j \in S\}$. Para simplificar a notação, o subgrafo $G[V(G) \setminus S]$ também será denotado por $G[\overline{S}]$

Considerando os conceitos de grafos, o CPAFLP permite duas formas de atender os clientes em U_1 e U_2: (i) ciclos e (ii) arborescências.

Um **caminho** é uma sequência de vértices $c = (v_0, v_1, \ldots, v_n)$ tal que $(v_i, v_{i+1}) \in A$, $\forall i = 0, \ldots, n-1$. Um **ciclo** é um caminho em que $v_0 = v_n$. Dizemos que um subconjunto de arcos S **induz um ciclo** se existir uma ordenação $(i_1 j_1, i_2 j_2, \ldots, i_{|S|} j_{|S|})$ dos arcos de S tal que a sequência de vértices $(i_1, i_2, \ldots, i_{|S|})$ define um ciclo.

O termo **arborescência** em grafos diz respeito a um grafo orientado que: (i) não possui vértices com grau de entrada maior que um; (ii) possui exatamente um único vértice de grau de entrada zero, sendo este, a raiz da arborescência e (iii) existe um único caminho com início na raiz, para cada um dos demais vértices do grafo. Dizemos que um subconjunto de arcos T de um grafo G **induz uma arborescência** se o subgrafo $(V[T], T)$ de G define uma arborescência.

Uma **pseudo-arborescência** é uma arborescência T com um arco (i,d) opcional no qual i é um vértice em T e d é a raiz de T. Note que um ciclo pode ser uma pseudo-arborescência quando T é um caminho $(v_0, v_1, \ldots, v_n)$ e o arco adicionado é (v_n, v_0). Além disso, uma arborescência também é uma pseudo-arborescência.

Por simplicidade, dizemos que um vértice v está **coberto** pela pseudo-arborescência T quando $v \in V(T)$.

Deste ponto em diante, utilizaremos o termo pseudo-arborescência, em consonância com a definição dada em Hoshino e Hill (2014), como sendo um par (R,T) em que R e T são subconjuntos de arcos de A que induzem um ciclo e uma família de arborescências em G, respectivamente, tais que:

- $R = \varnothing$ e T induz uma única arborescência cuja raiz é o depósito e este possui grau de saída um, ou;
- $R \neq \varnothing$ induz um ciclo que passa pelo depósito e T induz uma família de arborescências de modo que a raiz de cada arborescência é um vértice v em $V[R]$ e $v \neq d$.

A Figura 1.1 ilustra três exemplos de pseudo-arborescências. No exemplo 1 o subconjunto de arcos R é vazio e T induz uma arborescência com raiz em d; no exemplo 2 o subconjunto de arcos T é vazio e R induz um ciclo, já no exemplo 3 os subconjuntos de arcos R e T possuem elementos, dessa forma a pseudo-arborescência passa a ser composta por um ciclo, formado pelos vértices $(d, v1, v2, v3, v4)$, e uma arborescência com raiz em $v4$.

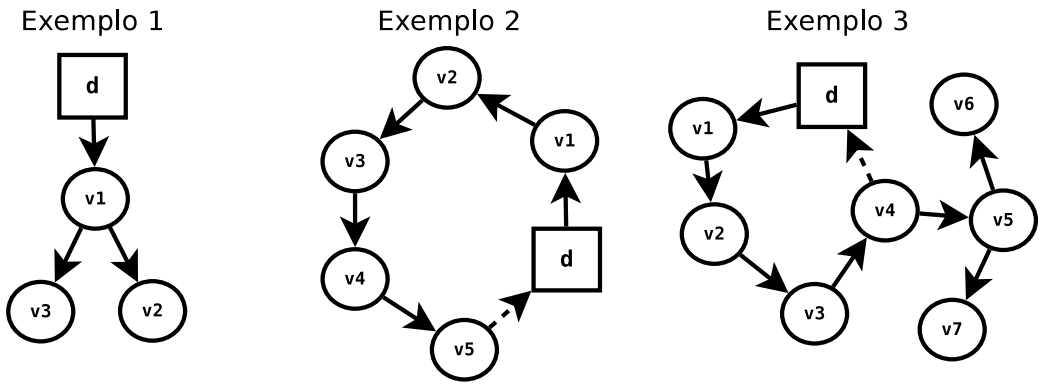

Figura 1.1: Exemplos de pseudo-arborescências.

A construção das pseudo-arborescências do CPAFLP é feita levando em consideração três limitantes inteiros, representados abaixo:

4. Um limite m de pseudo-arborescências;
5. Um limite Q de clientes que podem ser atendidos em cada uma das m pseudo-arborescências;
6. Um limite Q_t de clientes que podem estar presentes em cada arborescência, excluindo-se a raiz.

Uma solução do CPAFLP é considerada **válida** quando ela possui até m pseudo-arborescências e cada pseudo-arborescência obedece os limitantes Q e Q_t. É necessário também que a pseudo-arborescência, representada pelo par (R,T), possua $|R| \geq 3$ quando $R \neq \emptyset$, de forma que os arcos em R induzam a um

ciclo com mais de dois vértices. Consideramos que m, Q e Q_t são suficientes para cobrir todos os clientes de uma instância do CPAFLP.

Tendo em vista que o CPAFLP consiste em um problema de otimização combinatória que busca a minimização do custo no atendimento aos clientes, os dados de entrada para o problema devem ainda conter:

7. Um custo de instalação de facilidade, denominado c_v^{f}, adicionado a todo o vértice v em $V[R] \cap V[T]$, ou seja, v é a raiz de uma arborescência.
8. Um par de custos, c_a^{r} e c_a^{t}, associado a cada arco a do grafo G. Estes custos são referentes à utilização do arco em, respectivamente, um ciclo R e uma arborescência T.

Podemos então dizer que uma pseudo-arborescência $p = (R,T)$ é **(Q,Q_t)-capacitada** com respeito a $U_1 \cup U_2$ se o total de clientes é no máximo Q e cada arborescência induzida por T não contém mais do que Q_t clientes. Dessa forma, o custo de p, denotado por c_p, é calculado como:

$$c_p = \sum_{a \in R} c_a^{\mathrm{r}} + \sum_{a \in T} c_a^{\mathrm{t}} + \sum_{v \in V[R] \cap V[T]} c_v^{\mathrm{f}}.$$

O **problema das pseudo-arborescências capacitadas com localização de facilidades** consiste em encontrar, no máximo, m pseudo-arborescências válidas (Q,Q_t)-capacitadas disjuntas nos vértices, exceto pelo depósito, de forma que: (i) cada cliente do tipo 1 pertença a alguma pseudo-arborescência; (ii) cada cliente do tipo 2 pertença somente ao ciclo de alguma pseudo-arborescência e (iii) a soma dos custos das pseudo-arborescências seja mínima.

A Figura 1.2 mostra uma possível solução, contendo três pseudo-arborescências, para um grafo com 24 vértices. As pseudo-arborescências são representadas pelos conjuntos de vértices {1-9}, {10-19} e {20-23}. Para simplificar a visualização foram omitidos os custos c_a^{r} e c_a^{t} dos arcos.

O objetivo geral deste trabalho é avaliar o uso do método da geração de colunas para resolver de forma exata o problema da pseudo-arborescência capacitada com localização de facilidades, apresentando dois modelos matemáticos e duas heurísticas para acelerar a execução de implementações do tipo *branch-and-price* e *branch-and-bound*.

No Capítulo 2 serão abordados os trabalhos relacionados. A metodologia segue descrita no Capítulo 3. O trabalho desenvolvido e resultados obtidos seguem descritos, respectivamente, nos Capítulos 4 e 5. As contribuições acadêmicas, conclusões e trabalhos futuros seguem descritos no Capítulo 6.

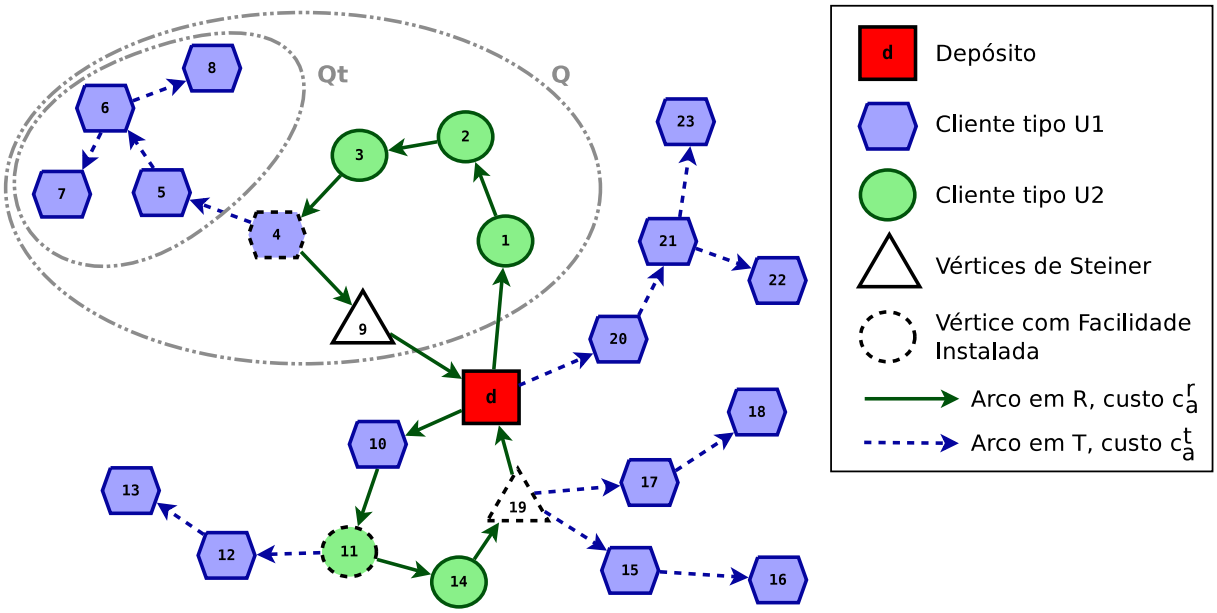

Figura 1.2: Exemplo de solução para o CPAFLP ($|U_2| = 5$, $|U_1| = 16$, $|W| = 2$, $Q = 9$, $Q_t = 4$, $m = 3$).

CAPÍTULO

2

Trabalhos Relacionados

Não há registros de trabalhos envolvendo o problema da pseudo-arborescência capacitada com localização de facilidades, de tal modo, apresentaremos nesta seção trabalhos que estão relacionados com o problema proposto e auxiliam no alcance dos objetivos apresentados.

O CPAFLP é uma derivação do ***capacitated ring tree problem*** **(CRTP)** e como tal, compartilham diversos elementos, como os conceitos de pseudo-arborescências, arborescências, ciclos e a busca por soluções de custo mínimo. Dessa maneira, para evidenciar as diferenças entre os problemas, faz-se necessário definir o CRTP sob as mesmas terminologias utilizadas para o CPAFLP.

Uma instância de entrada do CRTP consiste em duas constantes inteiras m e Q e um grafo simples orientado $G = (V, A)$, de forma que o conjunto de vértices V está particionado em quatro subconjuntos U_1, U_2, W e $\{d\}$, como no CPAFLP, mas apenas uma função custo c está associada aos arcos em A.

Uma pseudo-arborescência (R, T) é Q-capacitada se o número de clientes em $V[R \cup T]$ é no máximo Q. O custo de uma pseudo-arborescência é dado pela soma dos custos dos arcos em $R \cup T$.

O CRTP possui o mesmo objetivo do CPAFLP que consiste em encontrar, no máximo, m pseudo-arborescências Q-capacitadas disjuntas nos vértices, exceto pelo depósito, de forma que a soma dos custos das pseudo-arborescências seja mínima e que os clientes do tipo 2 somente façam parte de ciclos.

O trabalho descrito em Hoshino e Hill (2014) apresenta uma formulação matemática para o CRTP e aborda o problema com o método de geração de colunas.

Uma instância do CRTP pode ser reduzida a uma instância do CPAFLP em que $c_i^{\mathrm{f}} = 0$ para todo vértice i, $c_e^{\mathrm{t}} = c_e^{\mathrm{r}}$ para todo arco e e $Q_t = Q$.

O problema de roteamento de veículos, ***vehicle routing problem* (VRP)**, introduzido por Dantzig e Ramser (1959), diz respeito a problemas envolvendo a entrega de mercadorias, a partir de um depósito central, para clientes geograficamente dispersos. As entregas são feitas por m veículos que partem carregados do depósito, efetuam a distribuição da carga à medida que alcançam cada cliente e no fim retornam ao ponto de partida. A sequência de clientes, visitada por cada veículo, recebe o nome de rota. Cada rota possui um custo associado chamado de custo de roteamento, que diz respeito ao custo de visitar os clientes na sequência definida pela rota.

O objetivo do VRP é determinar as m rotas que atendam as demandas dos clientes de forma que a soma dos custos das m rotas seja mínima. É necessário considerar ainda, o fato de que cada cliente deve ser visitado uma única vez e cada veículo deve realizar apenas uma rota.

O VRP é um problema com muitas aplicações práticas e diversas variantes, então, ao considerar um limite de carga Q para cada veículo, o problema recebe o nome de ***capacitated vehicle routing problem* (CVRP)**, sendo este apresentado por Christofides et al. (1981). Dessa forma, a quantidade de clientes que podem ser visitados, em cada uma das m rotas, passa a ser limitada pela capacidade do veículo em atender a demanda por mercadorias dos clientes da rota.

O trabalho apresentado por Christofides et al. (1981) propõe um algoritmo *branch-and-bound* e diferentes relaxações e limitantes para o VRP. Já em Fukasawa et al. (2006) é proposto um algoritmo *branch-and-cut-and-price* para o CVRP. O trabalho de Toth e Vigo (2002) consiste em um apanhado bibliográfico que compara métodos de *branch-and-bound* para resolver o CRVP.

Uma instância do CVRP com demanda unitária pode ser reduzida a uma instância do CPAFLP em que $U_1 = \emptyset$, $W = \emptyset$, $U_2 = V \setminus \{d\}$, $c_i^{\mathrm{f}} = 0$ para todo vértice i, $Q_t = Q$ e $c_e^{\mathrm{t}} = c_e^{\mathrm{r}} = c_e$ para todo arco e. Como CVRP com demanda unitária é NP-difícil podemos assumir que o CPAFLP também é NP-difícil.

O problema dos anéis-estrelas capacitados, ***capacitated m-ring-star problem* (CmRSP)**, foi introduzido e abordado por Baldacci et al. (2007) como um problema de instalação de fibras óticas, porém em sua natureza é considerado uma variante do VRP, podendo ser facilmente visualizado em um contexto de entrega de mercadorias.

O CmRSP consiste em prover a entrega de mercadorias, a partir de um depósito central, para clientes geograficamente dispersos, tal qual o CVRP. O CmRSP também permite a utilização de pontos de Steiner, da mesma forma

que o CPAFLP. Os pontos de Steiner possuem demanda nula de produtos e são utilizados para reduzir o custo de roteamento. Outra forma de redução no custo do roteamento consiste em atender de forma indireta um cliente que não tenha sido visitado por alguma rota. O atendimento indireto é feito associando o cliente a um ponto da rota, o que na prática implica que o veículo de entrega deverá descarregar no ponto em questão a demanda de todos os clientes associados. Deslocar a mercadoria do ponto da rota para o cliente acarreta em um custo, chamado de custo de conexão.

No contexto do CmRSP, uma rota recebe o nome de anel e o conjunto de conexões recebe o nome de estrela, de forma que a união de uma rota e suas conexões recebe o nome de anel-estrela. O custo de uma rota anel-estrela é dado pelo custo de roteamento somado ao custo de todas as conexões realizadas. Sendo assim, o objetivo do CmRSP consiste em encontrar m anéis-estrelas que atendam as demandas dos clientes de forma que a soma dos m custos de roteamento e conexão sejam mínimos. Respeitando o fato de que todos os clientes devem ser atendidos e visitados uma única vez, bem como o limite da capacidade de mercadorias do veículo.

Baldacci et al. (2007) propõem um algoritmo *branch-and-cut* para resolver o CmRSP, Hoshino e De Souza (2012) propõem um algoritmo de *branch-and-cut-and-price* e Naji-Azimi et al. (2010) apresenta uma heurística para o CmRSP.

O CPAFLP e o CmRSP, embora compartilhem muitas características, não possuem uma redução imediata. O CmRSP pode ser facilmente visto como um caso especial de CPAFLP, em que apenas pseudo-arborescências $p = (R,T)$ com $R \neq \emptyset$ são admissíveis, $U_2 = \emptyset$, $c_i^{\mathrm{f}} = 0$ para todo vértice i, c^{t} representa a função de custo de conexão e toda arborescência em T, se houver, deve ter altura 1.

CAPÍTULO

3

Metodologia

Neste capítulo serão elencados os principais conceitos e métodos utilizados para o entendimento do problema a ser estudado.

3.1 Problemas de Otimização Combinatória

Segundo Johnson (1973) um **problema de otimização combinatória** (COP) P consiste em:

- um conjunto de instâncias $I_P = \{I_1, I_2, ..., I_n\}$;
- um conjunto de soluções $\mathcal{S}_P(I_i)$ associado a cada cada instância $I_i \in I_P$; e
- uma função $f_P : \mathcal{S}_P(I_i) \mapsto \mathbb{Q}$ que associa um valor para cada solução de cada instância $I_i \in I_P$.

O problema P é definido como um **problema de maximização**, quando o objetivo é encontrar, para cada instância $I_i \in I_P$, uma solução $S^* \in \mathcal{S}_P(I_i)$ com valor máximo em f, isto é, tal que $f(S^*) = \max\{f(S) : S \in S_P(I_i)\}$. Quando o objetivo for encontrar uma solução com valor mínimo, dizemos que P é um **problema de minimização**. Em ambos os casos, S^* é denominado **solução ótima** e $f(S^*)$ é conhecido como o **valor ótimo**.

Por outro lado, Wolsey (1998) define um **problema de otimização combinatória discreta** como um caso particular de COP em que uma instância do problema P pode ser representada por um conjunto finito de elementos $N = \{1, 2, \ldots, n\}$, uma função custo $c : N \mapsto \mathbb{Q}$ associada aos elementos de N e tal que o conjunto de soluções $\mathcal{S}$ é definido por uma coleção de

subconjuntos de elementos de N. O valor $f(S)$ associado a cada solução $S \in \mathcal{S}$ é determinado em função dos custos c_j dos elementos $j \in S$, por exemplo, $f(S) = \sum_{j \in S} c_j$. Esta função f é chamada **função objetivo**. Note que nem todo subconjunto de N é uma solução de P. Ao conjunto de condições para que um subconjunto $S \subseteq N$ pertença a $\mathcal{S}$ dá-se o nome de **conjunto de restrições** do problema e a cada subconjunto $S \subseteq N$ que satisfaz o conjunto de restrições do problema, isto é, a cada solução $S \in \mathcal{S}$ dá-se o nome de **solução viável**.

Desta forma, um problema de minimização P pode ser representado por:

$$(P) \qquad \min_{S \in \mathcal{S}} f(S). \tag{3.1}$$

3.2 Programação Linear Inteira

A programação linear inteira (PLI) é uma abordagem amplamente difundida no trato de problemas de otimização combinatória discreta. Os conceitos e resultados foram extraídos de Wolsey (1998).

Considere um problema de otimização combinatória discreta P, como apresentado em (3.1). Para caracterizar um subconjunto S de N, podemos utilizar n variáveis x, uma para elemento em N, de modo que $x_j = 1$ se o elemento $j \in S$ e $x_j = 0$, se $j \notin S$. Desta forma, resolver o problema P equivale a determinar o valor das variáveis x_j, $\forall j \in N$, que represente S^*. As variáveis x_j são chamadas **variáveis de decisão** do problema.

Um **problema de programação linear inteira (PLI)** é um problema de otimização combinatória discreta no qual as restrições que caracterizam a coleção de soluções viáveis $\mathcal{S}$ e a função objetivo do problema podem ser descritas através de inequações e equações lineares sobre o conjunto de variáveis de decisão do problema.

Um problema típico de PLI assume a seguinte forma:

$$\begin{aligned} \text{z} = \quad & \min \sum_{j=1}^{n} c_j x_j \\ \text{Sujeito a} \quad & \sum_{j=1}^{n} a_{ij} x_j = b_i, \forall i \in M \\ & x_j \in \mathbb{Z}, \forall j \in N, \end{aligned}$$

na qual x_j são as variáveis de decisão e M é o conjunto de índices das restrições lineares sobre as variáveis que definem o problema. Para cada $i \in M$, a componente a_{ij} representa o coeficiente de cada variável j na respectiva restrição, enquanto b_i denota o seu termo independente.

Um problema de PLI também pode ser escrito na forma matricial:

$$\text{(P)} \quad z = \min cx \tag{3.2}$$

$$\text{Sujeito a} \quad Ax = b \tag{3.3}$$

$$x \in \mathbb{Z}^n, \tag{3.4}$$

de forma que $c : 1 \times n$, $x : n \times 1$, $A : m \times n$ e $b : m \times 1$, graficamente representados:

$$c = \begin{bmatrix} c_1 & c_2 & \dots & c_n \end{bmatrix}, \; x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \; A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}, \; b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}.$$

A matriz A é denominada **matriz de coeficientes**, o vetor b é o vetor com os termos independentes das restrições, o vetor c representa o **custo das variáveis** na função objetivo. A matriz A e os vetores b e c são constantes e definem uma instância do problema. Em algumas situações, usamos a notação $A_{.j}$ e $A_{i.}$ para determinar a j-ésima coluna e a i-ésima linha da matriz A.

Entender um problema de otimização combinatória discreta e como modelá-lo como um problema de PLI consiste em uma das etapas propostas para o CPAFLP.

Antes de discutir as técnicas usadas para resolver problemas de PLI, apresentaremos alguns resultados importantes sobre programação linear, que pode ser vista como um caso mais geral de programação linear inteira.

3.2.1 Programação Linear

Programação linear inteira é utilizada para modelar diferentes situações práticas, nas quais as variáveis de decisão do problema modelado não podem admitir valores fracionários. Por exemplo, situações as quais estamos lidando com pessoas, configurações e objetos físicos, entre outros. Há outras situações em que admite-se que as variáveis de decisão assumam valores fracionários, por exemplo, quando deseja-se calcular tempo, quilogramas, entre outros. Para estas ocasiões utilizamos a **programação linear** (PL), que consiste em um problema de otimização combinatória, cujas restrições e função objetivo podem ser caracterizadas por inequações e equações lineares. Em suma, um problema de programação linear inteira é um caso especial de programação linear no qual as variáveis de decisão precisam ser inteiras.

No restante desta seção, são expostos alguns resultados importantes sobre programação linear, que foram compilados de Puccini (1972) e Luna e Goldbarg (2000).

Dualidade

Um dos resultados importantes em PL diz respeito à dualidade entre um par de problemas de PL, que chamamos de **Problema Primal** (P) e **Problema Dual** (D).

Considere (P) descrito por:

$$\begin{aligned} \text{(P)} \quad & z^* = \min cx \\ \text{Sujeito a} \quad & Ax \geq b \\ & x \geq 0. \end{aligned}$$

O problema dual (D) associado a (P) corresponde a:

$$\begin{aligned} \text{(D)} \quad & w^* = \max ub \\ \text{Sujeito a} \quad & uA \leq c \\ & u \geq 0. \end{aligned}$$

Segundo Luna e Goldbarg (2000), um problema dual preserva as seguintes condições:

- Se o problema primal for de maximização, o dual será de minimização e vice-versa;
- Existe uma simetria na descrição das restrições. Se o primal possuir restrições do tipo $\leq$ então o dual possuirá restrições $\geq$;
- Os termos independentes no primal surgem como os coeficientes da função objetivo no dual e vice-versa;
- O número de restrições do primal é igual ao número de variáveis do dual e vice-versa;
- A matriz de restrição do primal é a transposta da matriz de restrição do dual e vice-versa.

As variáveis do problema (P) recebem o nome de **variáveis primais** e do problema (D) de **variáveis duais**. Uma solução viável do problema (P) recebe o nome de **primal viável** e do problema (D), **dual viável**.

O **Teorema da Dualidade Forte** diz que se os problemas (P) e (D) são viáveis, então ambos possuem soluções ótimas finitas e suas funções objetivo apresentam resultados iguais.

A Figura 3.1, mostra a relação entre um par de problemas primal e dual.

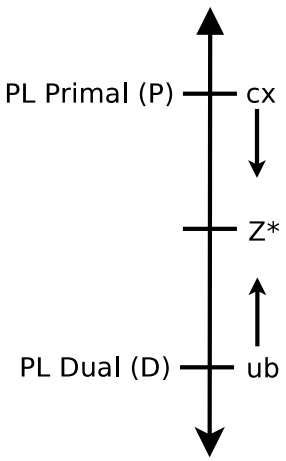

Figura 3.1: Par Primal e Dual em um problema de Minimização.

Método Simplex

O método *Simplex* consiste em um algoritmo utilizado para encontrar a solução ótima de um problema de PL. Segundo Luna e Goldbarg (2000) o algoritmo *Simplex* destaca-se como uma das grandes contribuições à programação matemática, pois trata-se de um algoritmo simples e, na prática, em geral resolve o problema em pouco tempo computacional.

Considere um problema de PL (P) dado por $z = \{\min cx : x \in X\}$, no qual $X = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$, A é a matriz de coeficientes de ordem $m \times n$, b é o termo independente de ordem $m \times 1$ e c é o custo de ordem $1 \times n$. O algoritmo *Simplex* baseia-se nos seguintes resultados:

- o conjunto das soluções viáveis X define um poliedro convexo;
- se P é viável e limitado, existe uma solução ótima de P e ela está associada a um ponto extremo do poliedro convexo X;
- um ponto extremo de X satisfaz n restrições linearmente independentes de X na igualdade e a solução associada ao ponto é chamada **solução básica**.

O método *Simplex* procura uma solução ótima considerando apenas as soluções básicas, isto é, os pontos extremos do poliedro. Toda solução básica corresponde a uma **base**, que é uma matriz quadrada inversível de ordem $m \times m$ obtida das colunas da matriz de coeficientes A. Note que cada coluna $A_{.j}$ da matriz A corresponde aos coeficientes de uma variável x_j nas restrições do problema. Logo, uma base corresponde a um conjunto de variáveis, as quais são denominadas **variáveis básicas**. As variáveis que não correspondem à base são chamadas **variáveis não-básicas**. O algoritmo começa com uma base inicial e sucessivamente muda de base de modo a melhorar o valor da função objetivo. A mudança de base depende de um passo do algoritmo, chamado

pricing, que é responsável por encontrar uma variável não básica, cuja coluna possa entrar na base para melhorar o valor da função objetivo. A melhoria de cada variável x_j na função objetivo é dada pelo que chamamos de **custo reduzido**, denotado por $\bar{c}_j$ e calculado pela seguinte equação:

$$\bar{c}_j = c_j - uA_{.j}, \tag{3.5}$$

de forma que u corresponde aos valores das variáveis duais do problema dual de (P). O valor de u pode ser calculado por $u = c_B B^{-1}$, sendo c_B o vetor de custo das variáveis básicas e B^{-1} é a inversa da base. A solução ótima é detectada pelo passo de *pricing* ao não existir uma variável não-básica de custo reduzido negativo.

O pseudocódigo 1 apresenta um esboço do *Simplex*.

Algoritmo 1: Passos do algoritmo *Simplex*.

Entrada: Problema P = $\{\min cx : Ax = b, x \geq 0, x \in \mathbb{R}^n\}$
Saída: Solução ótima x^* de P

```
início
    Encontre uma base inicial B;
    enquanto verdadeiro faça
        /* Passo de pricing */
        Escolha r tal que r = arg min_{j∈N}(c̄_j = (c_j − c_B B^{-1} A_{.j}))
        se c̄_r ≥ 0 então
            Pare e retorne a solução básica ótima x_B = B^{-1} b e x_N = 0
        y_r = B^{-1} A_{.r}
        se y_r ≥ 0 então
            Pare, P é ilimitado
        /* Mudança de base*/
        Escolha s tal que s = arg min_{(y_r)i>0}(B^{-1})_{i.} b/(B^{-1})_{i.} A_{.r}
        B ← B {A_{.s}} ∪ {A_{.r}}
```

3.2.2 Otimalidade, Relaxações e Limitantes

Considere um problema de PLI (P) dado por (3.2), (3.3) e (3.4) e reescrito na seguinte forma: $z = \{\min cx : x \in X \cap \mathbb{Z}^n\}$, na qual $X = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$.

Segundo Wolsey (1998), para provarmos a otimalidade de uma solução x^*, isto é, para provarmos que uma solução x^* é ótima, devemos encontrar um limitante superior $\bar{z} \geq z$ e um limitante inferior $\underline{z} \leq z$ de forma que $\bar{z} \geq z^* \geq \underline{z}$. Na prática isso significa dizer que o algoritmo encontrará uma sequência

decrescente de limitantes superiores

$$\bar{z}_1 > \bar{z}_2 > \ldots > \bar{z}_s \geq z$$

e uma sequência crescente de limitantes inferiores

$$\underline{z}_1 < \underline{z}_2 < \ldots < \underline{z}_t \leq z$$

até encontrar um valor ε tal que

$$\varepsilon \geq \bar{z}_s - \underline{z}_t,$$

sendo ε um valor não negativo pequeno o suficiente para provar a otimalidade.

Em um COP de maximização o limitante inferior é chamado de **limitante primal**, e o limitante superior de **limitante dual**. Portanto, para resolver um problema de otimização combinatória, o algoritmo precisa encontrar limitantes primais e duais. À diferença entre limitantes dá-se o nome de ***gap* de dualidade**.

Em um problema de maximização (ou minimização), qualquer solução viável $\bar{x} \in X \cap \mathbb{Z}^n$ fornece um limitante primal dado por $\bar{z} = c(\bar{x})$. Para alguns problemas de PLI, encontrar soluções viáveis é uma tarefa simples, nesse caso a questão passa a ser como encontrar boas soluções. Para outros problemas, encontrar uma solução viável pode ser tão complexo quanto o próprio PLI, motivo este que torna a utilização de heurísticas, para a descoberta de limitantes primais, um tópico de relevância para o trabalho. A Seção 3.3 abordará este tema.

Para encontrar um limitante dual é utilizada a técnica de relaxação do problema. A ideia da relaxação consiste em substituir um problema "difícil" de maximização, ou minimização, em PLI por um problema mais simples de otimização, tendo este, um valor ótimo pelo menos tão grande, ou pequeno, quanto z.

Um problema $z^R = \min\{f(x) : x \in T\}$ é uma relaxação de um problema de minimização $z = \min\{c(x) : x \in X\}$ se: (i) $X \subseteq T$ e (ii) $f(x) \leq c(x), \forall x \in X$.

A **relaxação linear** de um problema de PLI consiste no problema de programação linear obtido ao eliminar as restrições de integralidade das variáveis de decisão, permitindo que as mesmas assumam valores fracionários. A relaxação linear do problema $z = \{\min cx : x \in X \cap \mathbb{Z}^n\}$, na qual $X = \{x \in \mathbb{R}^n : Ax = b\}$, consiste em $z^{LP} = \{\min cx : x \in X\}$.

3.2.3 Branch-and-Bound

O método de *branch-and-bound*, proposto por Land e Doig (1960), é um paradigma de projeto de algoritmos que consiste no desenvolvimento de uma enumeração inteligente dos pontos candidatos à solução de um problema de otimização. O termo *branch* refere-se ao fato de que o método efetua partições no espaço de soluções e o termo *bound* ressalta que a prova da otimalidade da solução utiliza-se de limitantes calculados ao longo da enumeração dos candidatos, os quais também permitem a eliminação de candidatos não promissores.

Considere o problema (P) $z = \min\{cx : x \in S\}$, seja $S = S_1 \cup S_2 \cup \ldots \cup S_k$ uma decomposição de S em conjuntos menores, e seja $z^k = \min\{cx : x \in S_k\}$ para $k = 1, 2, \ldots, K$, logo $z = \min_k z^k$. A forma típica de representação dessa decomposição é através de uma árvore de enumeração, de forma que cada nó representa um subconjunto do espaço de soluções e cada nó filho uma componente de sua decomposição. A raiz da árvore representa todo conjunto S, ou seja, todo o espaço de soluções do problema.

Sejam $\bar{z}^k$ e $\underline{z}^k$, respectivamente um limite superior e inferior para z^k. Então $\bar{z} = \max_k \bar{z}^k$ é um limite superior em z e $\underline{z} = \max_k \underline{z}^k$ é um limite inferior em z. A primeira etapa do algoritmo consiste em encontrar os limitantes primal $(\bar{z})$ e dual $(\underline{z})$ da raiz da árvore de enumeração. Caso a solução ótima da relaxação de (P) seja um número inteiro, então o algoritmo chega ao fim, pois foi encontrada a solução ótima z^* de (P). Caso contrário, uma operação de *branching* é realizada, decompondo assim o espaço de soluções S em dois ou mais subconjuntos $S_1, \ldots, S_K$ que, consequentemente, dão origem a novos nós da árvore de enumeração.

A técnica de *branch-and-bound* evita a criação de toda a árvore de enumeração através de um método de remoção de nós, chamada de **poda**. A poda de um nó tem a função de inibir futuros *branchs* que não melhorarão a solução do problema (P). Um nó pode ser podado por: (i) inviabilidade, (ii) limitantes e (iii) otimalidade.

Nós que ainda não foram podados e ainda não foram decompostos por operações de *branching* recebem o nome de **nós ativos**. Considerando que os nós ativos são os únicos que podem gerar um limitante primal melhor para (P), a otimalidade pode ser detectada pelo algoritmo de *branch-and-bound* quando não há mais nós ativos ou quando o maior dos limitantes duais dos nós ativos for menor que o melhor limitante primal para (P). Sendo assim, podemos inferir que quanto melhor forem os limitantes duais e primais, melhor será o desempenho de um algoritmo de *branch-and-bound*.

O Algoritmo 2 apresenta um esboço de um algoritmo de *branch-and-bound* para o caso de árvores de enumeração binária.

Algoritmo 2: Passos de um Algoritmo de *Branch-and-Bound*.

Entrada: Problema P = $\{\min cx : x \in S \cap \mathbb{R}^n\}$
Saída: Melhor solução (ou ótima) x^* de P e limitantes duais e primais

```
1  início
2      /* iniciar a lista de problemas L */
3      L = {P};
4      z̄ = +∞;
5      z = −∞;
6      enquanto (L ≠ 0) e (z̄ − z ≥ ε) faça
7          /* passo de seleção de nó */;
8          Escolha um problema P^k ∈ L;
9          L = L − {P^k};
10         /* passo de otimização */;
11         Resolva a relaxação linear de P^k;
12         Seja x a solução ótima e z o valor ótimo da relaxação linear de P^k;
13         /* passo de poda */;
14         se P^k inviável então
15             Poda por inviabilidade;
16         senão
17             se z ≥ z̄ então
18                 Poda por limitante;
19             senão
20                 se x inteiro então
21                     se z < z̄ então
22                         z̄ = z;
23                         x* = x;
24                     Poda por otimalidade
25                 senão
26                     /* passo de branching */
27                     Crie dois subproblemas P_1 e P_2 a partir de P^k usando uma
                       regra de branching e coloque em L;
28         Atualize z̄ = min{limitante dual de P' : P' ∈ L};
29     Devolva x*, z̄, z;
```

O passo de *branching*, demonstrado na Linha 26 do Algoritmo 2, diz respeito ao processo de escolha do próximo nó a ser otimizado e recebe o nome de **critério de seleção do nó**. Em Wolsey (1998) são sugeridas duas estratégias de seleção. A primeira sugestão é chamada de estratégia *Depth-First Search* a qual realiza uma busca em profundidade nos nós da árvore de enumeração. Tal estratégia se mostra viável, pois a cada nível de profundidade atingido pelos nós, aumentam-se a quantidade de restrições, o que facilita a resolução da relaxação. A segunda sugestão recebe o nome de estratégia *Best-Node First* e tem como objetivo explorar o nó que possui o melhor limitante dual, no nosso exemplo, o maior limitante superior. Por exemplo, escolher o nó s de forma que $\bar{z}_s = \max_t \bar{z}_t$. Esta estratégia se mostra positiva, no sentido de que não serão explorados nós que possuem valor inferior ao ótimo.

3.2.4 Método de Geração de Colunas

Considere um problema de PL (P) dado por $z = \{\min cx : x \in X\}$, na qual $X = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$, A é a matriz de coeficientes de ordem $m \times n$, b é o termo independente de ordem $m \times 1$ e c é o custo de ordem $1 \times n$. Quando n, o número de variáveis, é muito grande, por exemplo, exponencial em relação a m, torna-se impossível resolver (P) usando o algoritmo *Simplex*. Uma das maneiras de se resolver tais problemas, é utilizando o método de geração de colunas, que consiste em resolver (P) dividindo-o em dois problemas: problema mestre restrito e problema de *pricing*. Para explicar o método, considere o seguinte problema de minimização, em que $\mathcal{P}$ representa o conjunto dos índices $\{1,2,\ldots,n\}$ das colunas da matriz A:

$$\begin{aligned} \text{Problema Mestre (PM)} \quad & \min \sum_{p \in \mathcal{P}} c_p \lambda_p \\ \text{s.a.} \quad & \sum_{p \in \mathcal{P}} a_i^p \lambda_p \geq b_i, \forall i \\ & \lambda_p \geq 0, \forall p \in \mathcal{P}. \end{aligned}$$

Sendo $|\mathcal{P}|$ exponencial no tamanho da entrada, é inviável resolvê-lo de forma tradicional. Portanto, considerando $\bar{P} \subseteq \mathcal{P}$ sendo $|\bar{P}| \ll |\mathcal{P}|$, obtemos um problema de minimização denominado Problema Mestre Restrito (PMR) que pode ser facilmente resolvido.

$$\begin{aligned} \text{(PMR)} \quad & \min \sum_{p \in \bar{P}} c_p \lambda_p \\ \text{s.a.} \quad & \sum_{p \in \bar{P}} a_i^p \lambda_p \geq b_i, \forall i \\ & \lambda_p \geq 0, \forall p \in \bar{P}. \end{aligned}$$

Entretanto, sendo $\bar{P} \subseteq \mathcal{P}$ a solução ótima do PMR não é necessariamente a solução ótima do Problema Mestre. Se não existir uma coluna $p \in \mathcal{P} \setminus \bar{P}$ tal que $\bar{c}_p$ seja negativo temos que a solução ótima do PMR é também a solução ótima do PM.

O problema de *pricing* equivale ao problema de se resolver o passo de *pricing* do algoritmo *Simplex*, o Linha 5 do Algoritmo 1. O **problema de *pricing*** consiste em encontrar, de forma implícita, $p^* \in \mathcal{P} \setminus \bar{P}$ tal que $p^* = arg\min_{p \in \mathcal{P} \setminus \bar{P}} \bar{c}_p$. Note que é inviável calcular explicitamente o custo reduzido de cada coluna p em $\mathcal{P} \setminus \bar{P}$, dado que potencialmente existem muitas colunas fora de $\bar{P}$. Em geral, as colunas $p \in \mathcal{P}$ estão associadas a alguma estrutura do problema, como ciclos de um grafo, árvores, etc. Desta forma, resolver o problema de *pricing* equivale a resolver outro problema de otimização combinatória, cuja função objetivo é minimizar $\bar{c}_p$. Se p^* é a solução ótima do problema de *pricing* e $\bar{c}_{p^*} < 0$ então adiciona-se p^* em $\bar{P}$. Caso contrário, a geração de colunas chega ao fim e a solução é devolvida.

O Algoritmo 3 apresenta o pseudocódigo do método da geração de colunas.

Algoritmo 3: Método da geração de colunas.

Entrada: Problema PM = $\{\min_{p \in \mathcal{P}} c_p \lambda_p : A\lambda \leq b, \lambda \geq 0, \lambda \in \mathbb{R}^{|\mathcal{P}|}\}$

Saída: Solução ótima x^* de PM

```
1  início
2      Considere P̄ ⊆ P tal que |P̄| ≪ |P|;
3      enquanto verdadeiro faça
4          Resolva o PMR = {min_{p∈P̄} c_p λ_p : Aλ ≤ b, λ ≥ 0, λ ∈ ℝ^{|P̄|}};
5          Seja λ* a solução ótima do PMR e π o valor dos duais do PMR;
6          /* Passo de pricing */
7          Encontre p* tal que p* = arg min_{p∈P\P̄} c̄_p = c_p − πA^P ;
8          se c̄_{p*} ≥ 0 então
9              Pare e retorne a solução ótima λ*;
10         senão
11             P̄ = P̄ ∪ {p*};
```

A Figura 3.2 exemplifica a relação entre o PMR e *Pricing*.

O método de geração de colunas é um tópico profundamente estudado e possui inúmeras referências valiosas, por exemplo os trabalhos desenvolvidos em Barnhart et al. (1998), Du Merle et al. (1999), Gondzio e Sarkissian (1996), Lübbecke e Desrosiers (2005), Vanderbeck e Wolsey (1996), Rousseau et al. (2007), Desaulniers et al. (2005), Vance et al. (1994), entre outros.

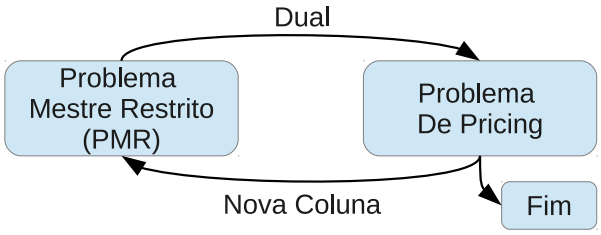

Figura 3.2: Problema Mestre Restrito e *Pricing*.

3.2.5 Branch-and-Price

Considere um problema de PLI (P) dado por $z = \{\min cx : x \in X \cap \mathbb{Z}^n\}$, no qual $X = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$ e tal que o número de variáveis n é exponencial em relação ao número de linhas da matriz de coeficientes A. Neste caso, não é possível usar um algoritmo de *branch-and-bound* convencional em que os limitantes de cada nó, dados pela relaxação linear, são obtidos pelo algoritmo *Simplex*. Uma solução, nestas situações, é usar um algoritmo de *branch-and-price*, que consiste no algoritmo de *branch-and-bound* em que o método da geração de colunas é usado para resolver a relaxação linear em cada nó.

Regras de *branching* convencionais, como *branching* sobre variáveis de decisão da forma $x_j \leq 0$ e $x_j \geq 1$, não funcionam adequadamente em algoritmos *branch-and-price*, pois não impedem que a coluna associada à variável x_j seja novamente gerada pelo problema de *pricing* no ramo da enumeração do *branching* $x_j \leq 0$. Uma regra de *branching* bastante conhecida foi proposta por Ryan e Foster (1981), que tem como objetivo gerar uma árvore de enumeração balanceada, de forma que o resultado dos ramos $x_j \leq 0$ e $x_j \geq 1$ resultem em uma mudança significativa no valor da função objetivo.

Considerando uma solução fracionária, resultado da relaxação linear, podemos assumir a existência de pelo menos um par de restrições r_1 e r_2, tal que:

$$0 < \sum_{j \in J(r_1, r_2)} x_j < 1,$$

sendo $J(r_1, r_2)$ o conjunto de todas as variáveis x_j tais que ambos $A_{r_1 j}$ e $A_{r_2 j}$ são iguais a 1. Dizemos que as restrições r_1 e r_2 são cobertas simultaneamente pelas colunas em $J(r_1, r_2)$. Sendo assim, as seguintes restrições de *branching* são criadas:

$$\sum_{j \in J(r_1, r_2)} x_j \leq 0, \tag{3.6}$$

e

$$\sum_{j \in J(r_1, r_2)} x_j \geq 1. \tag{3.7}$$

Dessa forma, a inequação (3.6) implica que as restrições r_1 e r_2 não devem ser cobertas juntas, já a inequação (3.7) força a cobertura simultânea de r_1 e r_2.

Portanto, a regra de *branching* de Ryan e Foster trabalha com um conjunto de variáveis $J(r_1, r_2)$ ao invés de uma única variável, tendo como principal vantagem um melhor balanceamento na árvore de enumeração.

3.3 Heurísticas e Metaheurísticas

Dado um problema de otimização (P), uma **heurística** consiste em um algoritmo que encontra, de forma rápida, soluções viáveis para (P) de modo que seus valores sejam próximos ao valor ótimo de (P). Uma heurística não dá uma garantia da qualidade do valor das soluções viáveis geradas e é, em geral, avaliada de forma empírica a partir dos resultados obtidos.

Metaheurísticas são modelos ou métodos para se construir heurísticas. Segundo Glover e Kochenberger (2003), elas consistem em controlar técnicas heurísticas promovendo a interação de processos de melhoria local e estratégias de nível superior para criar um processo capaz de escapar de resultados ótimos locais e realizar uma pesquisa consistente dentro de um espaço de solução.

Neste trabalho abordaremos a metaheurística Procedimento de Busca Adaptativa Gulosa e Aleatorizada (em inglês GRASP).

3.3.1 GRASP

O GRASP (*Greedy Randomized Adaptive Search Procedure*), em português, "Procedimento de Busca Gulosa Adaptativa Aleatória" é uma metaheurística originalmente proposta, em Feo e Resende (1989) e posteriormente detalhada em Feo e Resende (1995). Consiste em um algoritmo iterativo composto por duas fases: a fase de construção, em que uma solução é gerada elemento a elemento, e uma fase de busca local, na qual é pesquisado um resultado ótimo local na vizinhança da solução encontrada. Após todas as iterações do algoritmo, a melhor solução é retornada como resultado.

Para os próximos parágrafos, vamos considerar o problema de otimização combinatória, definido pelo conjunto de itens finito $E = \{1, 2, ..., n\}$, um conjunto exponencial de soluções possíveis $F \subseteq 2^E$ e uma função objetivo $f : 2^E \to \mathbb{R}$. Assumiremos que o problema P consiste em encontrar uma solução ótima $S^* \in F$ tal que $f(S^*) \leq f(S), \forall S \in F$, portanto um problema de minimização, tal qual descrito em Glover e Kochenberger (2003).

O Algoritmo 4 ilustra o bloco principal da Metaheurística GRASP, sendo *Max_Iteracoes* o número de iterações à ser realizado, *Semente* significa a semente inicial para o gerador de números pseudoaleatórios. α é um número real que

varia de 0 a 1 e tem relação com o nível de aleatoriedade do algoritmo. As variáveis *Solucao* e *Melhor_Solucao* armazenam, respectivamente, o conjunto de elementos de uma possível solução da respectiva iteração e a melhor solução encontrada entre todas as iterações, até então, realizadas. As funções *Construcao_Gulosa_Aleatorizada*() e *Busca_Local*() representam as duas fases do GRASP e serão detalhadas nas próximas subseções.

Algoritmo 4: Bloco principal GRASP.

Entrada: *Max_Iteracoes*, *Semente*, α
Saída: *Melhor_Solucao*

```
1 início
2     Leia_Entrada();
3     para k ← 1 até Max_Iteracoes faça
4         Solucao ← Construcao_Gulosa_Aleatorizada(Semente, α) ;
5         Solucao ← Busca_Local(Solucao) ;
6         Atualiza_Solucao(Solucao, Melhor_Solucao) ;
7     retorna Melhor_Solucao;
```

Fase de Construção

A fase iterativa de construção consiste em compor uma solução F, elemento a elemento, para o problema P. A cada iteração dessa fase, uma função avalia o custo incremental $c(e)$ para todo o elemento que faz parte da lista de possíveis candidatos C, sem considerar o impacto global da inclusão do referido item à solução (daí o nome *Greedy*). Na primeira iteração, a lista de possíveis candidatos C é composta por todo o conjunto de itens E.

Após a avaliação do custo incremental de $c(e) \in C$ o algoritmo seleciona os candidatos a partir do parâmetro α, repassado pelo bloco principal do GRASP, Linha 4 do Algoritmo 4, ordenando-os em uma Lista de Candidatos Restrita (*LCR*), conforme a natureza de maximização ou minimização do problema. Sobre a *LCR* é aplicado a componente aleatória do algoritmo (*Randomized*), que escolhe um dos itens da *LCR* para entrar na solução. Após a inclusão do item à solução, os custos incrementais $c(e)$ são recalculados (*Adaptative*) para o início da próxima iteração. As iterações são encerradas quando a lista de candidatos estiver vazia.

O Algoritmo 5 detalha os passos citados no parágrafo anterior. Os itens c^{min} e c^{max} significam respectivamente o menor e o maior custo incremental encontrado na lista de candidatos C. A linha 8, $LCR \leftarrow \{e \in C \mid c(e) \leq c^{min} + \alpha * (c^{max} - c^{min})\}$, justifica-se pela característica do problema de minimização P, caso o problema fosse de maximização a fórmula seria $LCR \leftarrow \{e \in C \mid c(e) \geq c^{max} - \alpha * (c^{max} - c^{min})\}$. Quanto mais próximo de 1 o α estiver, mais elementos farão parte da *LCR* e, consequentemente, maior será a aleatoriedade das

soluções encontradas.

Algoritmo 5: Função *Construcao_Gulosa_Aleatorizada*().

Entrada: *Semente*, α
Saída: *Solucao*

```
1  início
2      Solucao ← 0;
3      Inicializa a lista de candidatos: C ← E;
4      Avalia o custo incremental c(e) para todo e ∈ C;
5      enquanto C ≠ 0 faça
6          c^min ← mínimo {c(e) | e ∈ C};
7          c^max ← máximo {c(e) | e ∈ C};
8          LCR ← {e ∈ C | c(e) ≤ c^min + α * (c^max − c^min)};
9          Seleciona um elemento s da LCR aleatoriamente;
10         Solucao ← Solucao ∪ {s} ;
11         Atualiza a lista de candidatos C;
12         Reavalia o custo incremental c(e) para todo e ∈ C;
13     retorna Solucao;
```

Fase de Busca Local

A etapa de Busca Local é exemplificada no Algoritmo 6 e visa melhorar a solução obtida pela fase de construção, através de buscas locais na vizinhança $N()$ da *Solucao*. A fase de Busca Local só termina quando não existirem mais soluções possíveis que melhorem a função de custo $f(Solucao)$. Neste ponto, chegamos a uma solução localmente ótima.

Segundo Glover e Kochenberger (2003), a efetividade de um procedimento de busca local depende de vários fatores como a estrutura da vizinhança, a técnica de busca na vizinhança, a velocidade na avaliação do custo incremental dos vizinhos e até mesmo a qualidade da solução inicial fornecida pela Fase de Construção.

Algoritmo 6: Função *Busca_Local*().

Entrada: *Solucao*
Saída: *Solucao*

```
1  início
2      enquanto Solucao não é localmente ótima faça
3          Encontre Solucao' ∈ N(Solucao) com f(Solucao') < f(Solucao);
4          Solucao ← Solucao';
5      retorna Solucao;
```

CAPÍTULO

4

Trabalho Desenvolvido

Neste capítulo serão apresentados dois modelos matemáticos desenvolvidos para o CPAFLP, uma heurística primal e uma heurística de *pricing* para acelerar o processo de geração de colunas.

4.1 Modelos Matemáticos para o CPAFLP

O primeiro modelo matemático é uma formulação estendida que considera cada pseudo-arborescência (Q,Q_t)-capacitada como uma coluna em um modelo de partição de conjuntos. O segundo modelo é um modelo *multi-commodity flow* e faz menção aos trabalhos realizados em Hill e Voß (2014).

4.1.1 Formulação estendida para o CPAFLP

Seja P o conjunto de todas as pseudo-arborescências de uma instância do CPAFLP. Uma pseudo-arborescência $p = (R,T)$ em P pode ser representada por dois vetores característicos r e t de dimensões $(|V|+|A|)\times 1$. Os componentes de r são tais que r_i^p é igual a 1 se e somente se o vértice i pertence a $V[R]$ e r_{ij} é 1 se e somente se o arco ij está em R. O mesmo padrão é válido para os componentes de t, sendo que t_i^p é igual a 1 se e somente se o vértice i pertence a $V[T] \setminus V[R]$ e t_{ij} é 1 se e somente se o arco ij está em T. É necessário associar uma variável de decisão λ_p para cada pseudo-arborescência $p \in P$. Dessa forma, uma formulação estendida para o CPAFLP pode ser escrita como:

$$
\begin{aligned}
\text{(SPF)} \quad & \min \sum_{p \in P} c_p \lambda_p & & & \\
\text{sujeito a} \quad & \sum_{p \in P} \lambda_p \leq m & & (\pi) & (4.1) \\
& \sum_{p \in P} (r_i^p + t_i^p)\lambda_p = 1 & \forall i \in U_1 & (\alpha) & (4.2) \\
& \sum_{p \in P} r_i^p \lambda_p = 1 & \forall i \in U_2 & (\mu) & (4.3) \\
& \sum_{p \in P} (r_i^p + t_i^p)\lambda_p \leq 1 & \forall i \in W & (\gamma) & (4.4) \\
& \sum_{p \in P} (r_{ij}^p + t_{ij}^p)\lambda_p \leq 1 & \forall ij \in A & (\beta) & (4.5) \\
& 0 \leq \lambda_p \leq 1 & \forall p \in P. & & (4.6)
\end{aligned}
$$

A restrição (4.1) é referente ao limite de pseudo-arborescências que podem ser selecionadas de P. As restrições (4.2) e (4.3) forçam cada cliente do tipo U_1 ($i \in U_1$) a estar em somente uma pseudo-arborescência e cada cliente do tipo U_2 ($i \in U_2$) a estar em exatamente um ciclo. A restrição (4.4) impede os vértices de Steiner de aparecer, simultaneamente, em duas ou mais pseudo-arborescências. A restrição (4.5) limita a presença do arco a somente um ciclo ou uma arborescência em cada pseudo-arborescência. Por fim, a restrição (4.6) diz respeito a pseudo-arborescência p, que faz parte da solução se $\lambda_p = 1$ e caso contrário $\lambda_p = 0$.

Para lidar com o número exponencial de variáveis do CPAFLP utilizaremos o método de geração de colunas e a relaxação linear da formulação (SPF).

Sejam π, α, μ, γ e β as variáveis duais relacionadas, respectivamente, às restrições (4.1), (4.2), (4.3), (4.4), e (4.5) na relaxação linear de (SPF). Logo, o dual da relaxação linear (SPF) é:

$$
\begin{aligned}
\text{(dual)} \quad & \max \pi m + \sum_{i \in U_1} \alpha_i + \sum_{i \in U_2} \mu_i + \sum_{i \in W} \gamma_i + \sum_{ij \in A} \beta_{ij} \\
\text{sujeito a} \quad & \pi + \sum_{i \in U_1} (r_i^p + t_i^p)\alpha_i + \sum_{i \in U_2} r_i^p \mu_i + \sum_{i \in W} (r_i^p + t_i^p)\gamma_i + \sum_{ij \in A} (r_{ij}^p + t_{ij}^p)\beta_{ij} \leq c_p, \forall p \in P \\
& \pi \in \mathbb{R}, \alpha \in \mathbb{R}^{|U_1|}, \mu \in \mathbb{R}^{|U_2|}, \gamma_i \leq 0, \forall i \in W, \beta_{ij} \leq 0, \forall ij \in A.
\end{aligned}
$$

Relembrando da Seção 3.2.1, temos que o custo reduzido de uma coluna j é calculado pela equação 3.5, isto é, $\bar{c_j} = c_j - uA_{\cdot j}$, em que u é o vetor dos duais, c_j é o custo da variável x_j na função objetivo e $A_{\cdot j}$ é a coluna da variável x_j na matriz de coeficientes das restrições do problema.

Portanto, o custo reduzido de cada pseudo-arborescência $p = (R,T) \in P$ é dado por:

$$\bar{c}_p = c_p - \pi - \sum_{i \in U_1} (r_i^p + t_i^p)\alpha_i - \sum_{i \in U_2} r_i^p \mu_i - \sum_{i \in W} (r_i^p + t_i^p)\gamma_i - \sum_{ij \in A} (r_{ij}^p + t_{ij}^p)\beta_{ij},$$

que pode ser reescrito da seguinte forma:

$$\begin{aligned}\bar{c}_p = & \sum_{ij \in R} c_{ij}^{\mathrm{r}} + \sum_{ij \in T} c_{ij}^{\mathrm{t}} + \sum_{i \in V[R] \cap V[T]} c_i^{\mathrm{f}} \\ & -\pi - \sum_{i \in U_1} (r_i^p + t_i^p)\alpha_i - \sum_{i \in U_2} r_i^p \mu_i - \sum_{i \in W} (r_i^p + t_i^p)\gamma_i - \sum_{ij \in A} (r_{ij}^p + t_{ij}^p)\beta_{ij},\end{aligned}$$

ou ainda,

$$\begin{aligned}\bar{c}_p = & -\pi + \sum_{ij \in R, j \in U_1} (c_{ij}^{\mathrm{r}} - \beta_{ij} - \alpha_j) + \sum_{ij \in T, j \in U_1} (c_{ij}^{\mathrm{t}} - \beta_{ij} - \alpha_j) \\ & + \sum_{ij \in R, j \in U_2} (c_{ij}^{\mathrm{r}} - \beta_{ij} - \mu_j) \\ & + \sum_{ij \in R, j \in W} (c_{ij}^{\mathrm{r}} - \beta_{ij} - \gamma_j) + \sum_{ij \in T, j \in W} (c_{ij}^{\mathrm{t}} - \beta_{ij} - \gamma_j) \\ & + \sum_{i \in V[R] \cap V[T]} c_i^{\mathrm{f}}.\end{aligned}$$

Sejam $\bar{c}_{ij}^{\mathrm{r}} = c_{ij}^{\mathrm{r}} - \beta_{ij} - k_j$, o custo reduzido de cada arco ij do ciclo R e $\bar{c}_{ij}^{\mathrm{t}} = c_{ij}^{\mathrm{t}} - \beta_{ij} - k_j$, o custo reduzido de cada arco ij na arborescência T, nos quais

$$k_j = \begin{cases} 0 & , \text{se } j = d \\ \alpha_j & , \text{se } i \in U_1 \\ \mu_j & , \text{se } i \in U_2 \\ \gamma_j & , \text{se } i \in W. \end{cases}$$

Considere uma variável binária f_j que é igual a 1 se e somente se $j \in V[R] \cap V[T]$. Utilizando $\bar{c}_{ij}^{\mathrm{r}}$, $\bar{c}_{ij}^{\mathrm{t}}$ e f_j, temos que o custo reduzido de uma pseudo-arborescência $p = (R,T)$ é

$$\bar{c}_p = \sum_{ij \in A} \bar{c}_{ij}^{\mathrm{r}} r_{ij}^p + \sum_{ij \in A} \bar{c}_{ij}^{\mathrm{t}} t_{ij}^p + \sum_{i \in V} c_i^{\mathrm{f}} f_i - \pi.$$

O problema de *pricing* para (SPF) requer a computação de $\min_{p \in P} \bar{c}_p$, que é uma generalização do *Profitable Tour Problem (PTP)* de Dell'Amico et al. (1995). O PTP consiste em, dados um grafo $G = (V,E)$ com custos nas arestas e lucros nos vértices, encontrar um ciclo C em G que maximize a soma dos lucros dos vértices em C menos a soma dos custos das arestas em C. Dessa forma, o PTP é um caso especial do problema de *pricing* em que $U_1 = W = \emptyset$ e $Q = |V|$, de modo que os clientes somente apareçam em ciclos, não há vértices de *Steiner* e não há limite no número de clientes atendidos. Segundo Feillet et al. (2005), o *PTP* é NP-difícil.

Segue abaixo uma formulação em programação linear inteira para o

problema de *pricing*.

$$
\begin{aligned}
\text{(Pric)} \quad & \min\Big(\sum_{e\in A}\overline{c}_e^{\,r}x_e+\sum_{e\in A}\overline{c}_e^{\,t}w_e+\sum_{i\in V}c_i^{f}f_i\Big) && && \\
\text{sujeito a} \quad & \sum_{e\in\delta^+(i)}w_e\le|V|(y_i+\sum_{k\in V}z_i^k) && \forall i\in V\setminus\{d\} && (4.7)\\
& \sum_{e\in\delta^+(d)}w_e\le z_d && && (4.8)\\
& \sum_{e\in\delta^-(i)}w_e=\sum_{k\in V}z_i^k && \forall i\in U_1\cup W && (4.9)\\
& \sum_{e\in\delta^+(i)}x_e=y_i && \forall i\in V && (4.10)\\
& \sum_{e\in\delta^-(i)}x_e=y_i && \forall i\in V && (4.11)\\
& y_i+\sum_{k\in V}z_i^k\le 1 && \forall i\in U_1\cup W && (4.12)\\
& y_d+z_d\le 1 && && (4.13)\\
& \sum_{i\in U_1\cup U_2}y_i+\sum_{i\in U_1}\sum_{k\in V}z_i^k\le Q && && (4.14)\\
& \sum_{i\in U_1}z_i^k\le f_kQ_t && \forall k\in V && (4.15)\\
& y_k\ge f_k && \forall k\in V\setminus\{d\} && (4.16)\\
& z_d\ge f_d && && (4.17)\\
& (i+1)y_i+|V|w_{ij}-\sum_{k\in U}(k+1)z_j^k+\sum_{k\in U}(k+1)z_i^k\le|V| && \forall ij\in A && (4.18)\\
& -(i+1)y_i+|V|w_{ij}+\sum_{k\in U}(k+1)z_j^k-\sum_{k\in U}(k+1)z_i^k\le|V| && \forall ij\in A && (4.19)\\
& f_k\le\sum_{i\in U_1\cup W}z_i^k && \forall k\in V && (4.20)\\
& \sum_{e\in\delta^+(i)}w_e\ge\sum_{k\in U}z_i^k && \forall i\in W && (4.21)\\
& \sum_{\substack{i,j\in S\\ j\ne i}}x_{ij}\le|S|-1 && \forall S\subseteq V\setminus\{d\},|S|\ge 2 && (4.22)\\
& \sum_{\substack{i\in S\\ j\in S\cap(U_1\cup W)\\ j\ne i}}w_{ij}\le|S|-1 && \forall S\subseteq V,|S|\ge 2 && (4.23)\\
& x\in\{0,1\}^{|V|\times(|V|-1)},y\in\{0,1\}^{|V|},z\in\{0,1\}^{|V\setminus U_2|\times|V|},\\
& w\in\{0,1\}^{|V|\times(|U_1|+|W|)},f\in\{0,1\}^{|V|},z_d\in\{0,1\}^1. && && (4.24)
\end{aligned}
$$

Neste modelo, utilizamos cinco grupos de variáveis de decisão para caracterizar uma pseudo-arborescência (R,T), nos quais:

$$
x_{ij}=\begin{cases}1, & \text{se o arco } ij\in A \text{ faz parte do ciclo } R;\\ 0, & \text{caso contrário;}\end{cases}
$$

$$y_i = \begin{cases} 1, & \text{se o vértice } i \in V \text{ pertence a } V[R]; \\ 0, & \text{caso contrário;} \end{cases}$$

$$z_i^k = \begin{cases} 1, & \text{se } i \in (U_1 \cup W) \text{ pertence a uma arborescência em } T \text{ com raiz } k \in V; \\ 0, & \text{caso contrário;} \end{cases}$$

$$w_{ij} = \begin{cases} 1, & \text{se } ij \in A \text{ é um arco de uma arborescência em } T; \\ 0, & \text{caso contrário;} \end{cases}$$

$$f_i = \begin{cases} 1, & \text{se o vértice } i \in V \text{ pertence a } V[R] \cap V[T]; \\ 0, & \text{caso contrário.} \end{cases}$$

e

$$z_d = \begin{cases} 1, & \text{se o depósito é a raiz de uma pseudo-arborescência } (R,T), \text{ com } R = \emptyset; \\ 0, & \text{caso contrário.} \end{cases}$$

A restrição (4.7) garante que um arco ij pertença a uma arborescência somente se i está no ciclo ou na arborescência. A restrição (4.8) garante que somente um arco da arborescência inicie no depósito, neste caso o depósito é a raiz da arborescência, caso contrário nenhum arco da arborescência pode sair do depósito. A restrição (4.9) garante que cada vértice i de uma arborescência tenha grau de entrada um. As restrições (4.10) e (4.11) garantem que para cada vértice i de um ciclo, existam exatamente um arco chegando em i e um arco saindo de i. As restrições (4.12) e (4.13) impedem um vértice de pertencer a um ciclo e uma arborescência ao mesmo tempo. A restrição (4.14) limita o número de clientes na pseudo-arborescência. A restrição (4.15) limita a quantidade de clientes conectados por uma arborescência a cada facilidade. Dizemos que um cliente está conectado a uma facilidade quando existe um caminho entre o vértice com facilidade instalada e o respectivo cliente. A restrição (4.16) garante que somente sejam instaladas facilidades em vértices do ciclo, exceto o depósito. A restrição (4.17) garante que seja instalada facilidade no depósito, somente quando este for raiz de uma arborescência. As restrições (4.18) e (4.19) forçam os vértices i e j a terem a mesma raiz quando o arco ij faz parte de uma arborescência. A restrição (4.20) garante que f_k é zero quando não há vértices de arborescências com raiz em k. A restrição (4.21) proíbe a utilização de vértices de Steiner como folhas nas arborescências. A restrição (4.22) e (4.23) evitam o acontecimento de subciclos em um ciclo e ciclos em uma arborescência. Por fim (4.24) são as restrições de integralidade das variáveis.

É importante ressaltar que caso f_d seja 1, conforme dado pela restrição (4.17), a pseudo-arborescência com raiz no depósito deve obedecer o

limite de Q_t clientes dado pela restrição (4.15).

Ao invés das restrições (4.22) e (4.23), podemos utilizar as restrições polinomiais de Miller et al. (1960):

$$|V|w_{ij} + u_i - u_j \leq |V| - 1, \forall i \in V \setminus \{d\}, \forall j \in U_1 \cup W, i \neq j \tag{4.25}$$

$$|V|x_{ij} + u_i - u_j \leq |V| - 1, \forall i \in V \setminus \{d\}, \forall j \in V \setminus \{d\}, i \neq j. \tag{4.26}$$

As equações (4.25) e (4.26) utilizam uma variável inteira não negativa u_i menor ou igual a $|V|$ para cada $i \in V \setminus \{d\}$. As variáveis u_i são definidas para cada vértice do grafo e elas impõem uma ordem entre os vértices de acordo com o sentido das arestas selecionadas em x e em w. O valor de u_i deve ser menor que u_j se a variável $x_{ij} = 1$ ou $w_{ij} = 1$. Dessa forma, evitando a formação de um subciclo na solução.

4.1.2 Formulação compacta para o CPAFLP

Considere $U = U_1 \cup U_2$. Para cada cliente $k \in U$ e cada arco $(i, j) \in A$, é definida uma variável f_{ij}^k para medir o fluxo através do arco ij para atender o cliente k. A variável x_a é igual a 1 se o arco a pertence a uma pseudo-arborescência na solução ao passo que z é o vetor característico dos arcos que pertencem ao ciclo de uma pseudo-arborescência. A variável y_i é igual a 1 se uma facilidade está instalada no vértice i. Variável g_a^k é utilizada para medir o fluxo através do arco a que pertence a uma arborescência para atender o cliente k. Seja $h_i^i = 1, \forall i \in U$, $h_d^k = -1, \forall k \in U$, e $h_i^k = 0$ em todos os outros casos. Logo, o CPAFLP pode ser formulado como:

$$\text{(MCF)} \quad \min\Big(\sum_{a\in A} c_a^{\mathrm{t}} x_a + \sum_{a\in A}(c_a^{\mathrm{r}} - c_a^{\mathrm{t}}) z_a + \sum_{i\in V} c_i^{\mathrm{f}} y_i\Big) \tag{4.27}$$

$$\text{subject to} \quad \sum_{a\in\delta^+(d)} x_a \le m \tag{4.28}$$

$$\sum_{ij\in\delta^-(i)} f_{ij}^k - \sum_{ij\in\delta^+(i)} f_{ij}^k = h_i^k \qquad \forall i\in V, \forall k\in U \tag{4.29}$$

$$f_a^k \le x_a \qquad \forall a\in A, \forall k\in U_1 \tag{4.30}$$

$$f_a^k \le z_a \qquad \forall a\in A, \forall k\in U_2 \tag{4.31}$$

$$z_a \le x_a \qquad \forall a\in A \tag{4.32}$$

$$\sum_{a\in\delta^-(i)} x_a \le 1 \qquad \forall i\in W \tag{4.33}$$

$$\sum_{a\in\delta^-(i)} x_a = 1 \qquad \forall i\in U \tag{4.34}$$

$$\sum_{k\in U} f_a^k \le Q \qquad \forall a\in\delta^+(d) \tag{4.35}$$

$$\sum_{k\in U}\sum_{a\in\delta^+(i)} g_a^k \le Q_t \qquad \forall i\in V\setminus\{d\} \tag{4.36}$$

$$\sum_{k\in U} g_a^k \le Q_t \qquad \forall a\in\delta^+(d) \tag{4.37}$$

$$\sum_{a\in\delta^-(i)} z_a = \sum_{a\in\delta^+(i)} z_a \qquad \forall i\in V\setminus\{d\} \tag{4.38}$$

$$x_{ij} + x_{ji} \le 1 \qquad \forall\{i,j\}\in A \tag{4.39}$$

$$\sum_{a\in\delta^+(i)} z_a + |V| y_i \ge \sum_{a\in\delta^+(i)} x_a - |V| + |V| \sum_{a\in\delta^-(i)} z_a \qquad \forall i\in V \tag{4.40}$$

$$g_a^k \ge f_a^k - z_a \qquad \forall a\in A, \forall k\in U \tag{4.41}$$

$$f_a^k,\ g_a^k \ge 0 \qquad \forall a\in A, \forall k\in U \tag{4.42}$$

$$x\in\{0,1\}^{|A|}, z\in\{0,1\}^{|A|}, y\in\{0,1\}^{|V|} \tag{4.43}$$

Considerando os custos de utilização dos arcos em ciclos (c_a^{r}), os custos dos arcos em arborescências (c_a^{t}) e os custos de instalação de facilidade (c_i^{f}) para todo vértice, a função objetivo do (MCF) pode ser vista em (4.27).

A restrição (4.28) limita o número de pseudo-arborescências em m. A conservação do fluxo dos arcos e eliminação de subciclos é garantida por (4.29) ao passo que os fluxos e os arcos são conectados em (4.30), (4.31) e (4.32). As inequações (4.33) e (4.34) forçam cada cliente a ter grau de entrada 1 na pseudo-arborescência e cada vértice de Steiner a, no máximo, 1. As inequações (4.35), (4.36) e (4.37) são restrições de capacidade que dizem respeito a Q e Q_t. A formação dos ciclos é garantida pela restrição (4.38). A restrição (4.39) proíbe a formação de ciclos com dois vértices. A restrição (4.40) força a instalação de facilidade nos vértices do ciclo que são atravessados por fluxos de arborescências. A restrição (4.41) obriga a

existência de fluxo g_a^k sempre que houver um fluxo f_a^k em um arco $a \in T$ para atender um cliente k, desde que $a \notin R$. Dessa forma é possível controlar a quantidade de clientes e a orientação das arborescências conforme mostrado pelas restrições (4.36) e (4.37). As restrições (4.42) e (4.43) referem-se à não negatividade e integralidade das variáveis, respectivamente.

4.2 Heurísticas Propostas

A primeira heurística apresentada é a Heurística Primal e tem como objetivo prover uma base inicial para o método de geração de colunas e gerar uma solução primal. A segunda heurística apresentada é uma Heurística de *Pricing* que tem como objetivo gerar pseudo-arborescências com custo reduzido negativo.

4.2.1 Heurística Primal

O objetivo da heurística primal é obter até max_iter soluções válidas que cobrem todos os clientes do grafo e respeitam os limitantes m, Q e Q_t.

O algoritmo proposto constrói uma solução para o CPAFLP de maneira incremental, de modo que as m pseudo-arborescências são aumentadas vértice a vértice até a cobertura de todos os clientes. A cobertura dos vértices é iniciada pelos clientes U_2, seguido pelos clientes U_1 e finalizada com os vértices de Steiner W. Os vértices de Steiner são utilizados somente quando implicam na redução do custo da solução ou quando são necessários para tornar a solução válida.

Por simplicidade, as pseudo-arborescências da solução são definidas como **tipo 1**, quando esta é formada por uma arborescência com raiz no depósito, e como **tipo 2**, quando a pseudo-arborescência formada possui um ciclo que passa através do depósito e tem, ou não, arborescências com raiz em algum vértice v em $V[R]$ e $v \neq d$.

O primeiro passo do algoritmo consiste na definição dos tipos de cada uma das m pseudo-arborescências que farão parte da solução. Após a definição o algoritmo procede com a escolha dos vértices para compor a solução. A escolha é realizada através dos passos de criação, seleção e inclusão de candidatos nas pseudo-arborescências. Tendo em vista que a heurística primal busca obter soluções válidas, o algoritmo prioriza a criação de candidatos para as pseudo-arborescências ainda inválidas e somente depois são gerados candidatos para todas as pseudo-arborescências com menos de Q clientes.

As principais estruturas de dados utilizadas pela heurística primal são a instância de entrada I, o vetor de pseudo-arborescências PA e o vetor de

soluções *SOL*.

A instância de entrada $I = (G, c_v^{\mathrm{f}}, c_a^{\mathrm{r}}, c_a^{\mathrm{t}}, m, Q, Q_t)$ é composta por um grafo orientado $G = (V, A)$, um vetor de custos de instalação de facilidade $c_v^f \ \forall v \in V$, um vetor de custo de utilização do arco em um ciclo $c_a^r \ \forall a \in A$ e um vetor de custo de utilização do arco em uma arborescência $c_a^t \ \forall a \in A$. A instância ainda é composta pelos limitantes m, Q e Q_t. O conjunto de vértices do grafo G é composto por vértices dos tipos U_2, U_1 e W, sendo os clientes representados por $U = (U_2 \cup U_1)$.

O vetor de pseudo-arborescências *PA* é utilizado para armazenar o tipo de cada pseudo-arborescência que compõe a solução, os vértices cobertos, as facilidades instaladas, os arcos utilizados e seus respectivos custos.

O vetor *SOL* armazena todas as soluções encontradas para uma mesma instância I.

A heurística também recebe como entrada duas constantes, $\alpha \in [0,1]$ e *max_iter*. A constante α diz respeito ao nível de aleatoriedade dos algoritmos de cobertura. Quanto mais próximo de 0 for o valor de α, mais gulosos os algoritmos se tornam, e quanto mais próximo de 1 mais aleatória será a cobertura dos vértices. A constante *max_iter* possui valor mínimo 1 e diz respeito ao quantitativo de soluções que se deseja obter de uma mesma instância.

O corpo principal do algoritmo, conforme pode ser visto no Algoritmo 7, busca definir o tipo de cada uma das m pseudo-arborescências e realizar as chamadas das funções responsáveis pela cobertura dos vértices U_2, U_1 e Steiner.

Algoritmo 7: Heurística Primal.

```
   Entrada: I, α, max_iter
   Saída: PA
1  início
2      SOL ← ∅;
3      para it = 0 to max_iter faça
4          PA ← define_tipo(I);
5          PA ← Cobre_U2(PA, I, α);
6          PA ← Cobre_U1(PA, I, α);
7          PA ← Cobre_W(PA, I, α);
8          SOL ← SOL ∪ PA;
9      retorna SOL
```

A função *define_tipo*, descrita no Algoritmo 8, calcula a quantidade de pseudo-arborescências do tipo 2 necessárias para comportar a cobertura de todos os vértices, respeitando os limitantes m, Q e Q_t.

A Linha 2 do Algoritmo 8 calcula a quantidade de pseudo-arborescências do tipo 2, necessárias à cobertura dos clientes U_2. A Linha 4 verifica se as pseudo-

Algoritmo 8: Função *define_tipo*.

Entrada: I
Saída: PA

```
1 início
2     aux1 ← ⌈|U2|/Q⌉;
3     aux2 ← ∅;
4     se (m − aux1) ∗ Qt < |U1| então
5         aux2 ← ⌈(|U1| − (m − aux1) ∗ Qt)/(Q − Qt)⌉;
6     x ←mínimo(aux1 + aux2, m);
7     Marque (x) PA como tipo 2 e (m − x) PA como tipo 1;
8     retorna PA
```

arborescências do tipo 1, são suficientes para acomodar todos os clientes U_1, caso contrário, a Linha 5 calcula a quantidade de pseudo-arborescências do tipo 2, necessárias para cobrir tais clientes. A Linha 6 limita em m a quantidade de pseudo-arborescências do tipo 2. Por fim, a Linha 7 marca o tipo, 1 ou 2, de cada uma das pseudo-arborescências que fazem parte do vetor PA.

Observe que o número de pseudo-arborescências do tipo 2, calculado nas linhas 2 e 5, pode ser maior que o necessário e, inclusive, maior que o limite m devido ao uso da função teto. Essa situação é corrigida na Linha 6, que só é possível pois, conforme descrito no Capítulo 1, todas as instâncias possuem uma solução viável.

Após a definição dos tipos de cada pseudo-arborescência de PA, é realizada a chamada da função responsável por cobrir os clientes U_2. Os detalhes da função $Cobre\_U_2$ podem ser vistos no Algoritmo 9.

O Algoritmo 9 realiza a cobertura de todos os clientes U_2 ao mesmo tempo que busca tornar válidas todas as pseudo-arborescências do tipo 2. O primeiro passo do algoritmo $Cobre\_U_2$ consiste em selecionar as pseudo-arborescências inválidas com menos de Q clientes, etapa realizada na Linha 4. Caso não existam pseudo-arborescências inválidas, são armazenadas em S todas as pseudo-arborescências que ainda comportam clientes, conforme Linha 6.

Na Linha 7 acontece a geração da lista de candidatos L, de forma que cada elemento em L é uma tripla composta por uma lista de arcos $\mathcal{J}$ que devem ser inseridos e/ou retirados de uma pseudo-arborescência pa para incluir um cliente ainda não coberto cl. A função *insere_ciclo* participa na geração desta lista calculando o menor custo de inserção de cl e retornando a dupla $(\mathcal{J}, pa)$. A forma de inserção pode ser visualizada no exemplo $a)$ da Figura 4.1, que demonstra a inserção do vértice k, através da remoção do arco (i, j) e adição dos arcos (i,k) e (k, j). Os exemplos $b)$, $c)$ e $d)$ serão tratados no decorrer do capítulo.

Algoritmo 9: Função $Cobre\_U_2$.

Entrada: PA, I, α
Saída: PA

```
1  início
2      cobertos ← ∅;
3      faça
4          S ← {(R,T) ∈ PA : |V[R∪T] ∩ U| < Q e |V[R]| < 3};
5          se S = ∅ então
6              S ← {(R,T) ∈ PA : |V[R∪T] ∩ U| < Q};
7          L ← {(cl, J, pa) : cl ∈ U2 \ cobertos, (J, pa) = insere_ciclo(cl, S, I)};
8          LCR ← {cand ∈ L : c(cand) ≤ c^min + α * (c^max − c^min)};
9          (cl', J', pa') ← aleatorio(LCR);
10         cobertos ← cobertos ∪ {cl'};
11         Insere cl' em pa' usando J';
12         Atualiza(PA, pa');
13     enquanto |L| > 1;
14     retorna PA
```

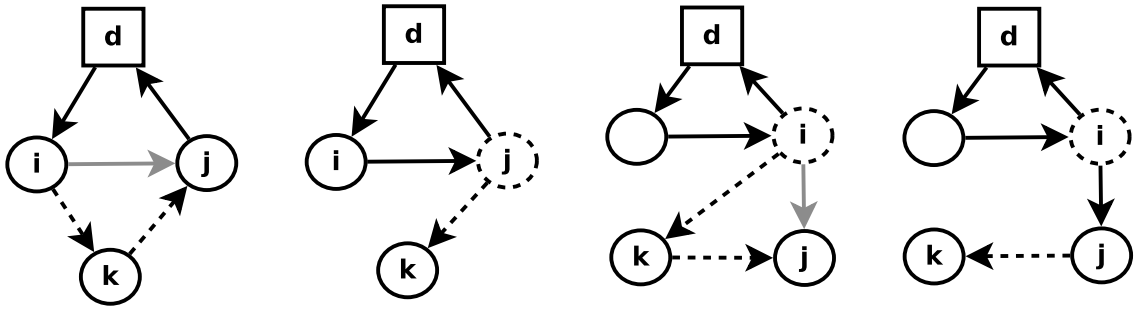

Figura 4.1: Exemplos de inserção de vértices.

O custo de um candidato, em L, está diretamente relacionado com a forma de inserção do cliente na pseudo-arborescência. O cálculo do custo de um candidato é dado por $\mathcal{S}^{+\mathrm{r}} + \mathcal{S}^{+\mathrm{t}} - c^{-\mathrm{r}} - c^{-\mathrm{t}} + c^{+\mathrm{f}}$ sendo que:

- $\mathcal{S}^{+\mathrm{r}}$: é a somatória dos custos dos arcos, em ciclo, incluídos;
- $\mathcal{S}^{+\mathrm{t}}$: é a somatória dos custos dos arcos, em arborescência, incluídos;
- $c^{-\mathrm{r}}$: é o custo do arco, em ciclo, a ser removido, se necessário.
- $c^{-\mathrm{t}}$: é o custo do arco, em arborescência, a ser removido, se necessário.
- $c^{+\mathrm{t}}$: é o custo de instalação de facilidade em um vértice, se necessário.

Após a criação da lista de candidatos L, o algoritmo gera uma lista de candidatos restrita que recebe o nome de LCR, conforme pode ser observado na Linha 8 do Algoritmo 9. A lista LCR é gerada a partir do cálculo envolvendo os custos dos candidatos de L e α. Os valores c^{max} e c^{min} dizem respeito, respectivamente, ao maior e menor custo encontrado dentre todas as triplas em L.

Na Linha 9 é escolhido, de forma aleatória, um cliente da *LCR*. A partir do candidato escolhido $(x', \mathcal{J}', pa')$, são realizados três passos. O primeiro passo consiste na marcação do vértice x' como um vértice já coberto pelo algoritmo, conforme mostrado na Linha 10. O segundo passo acontece na Linha 11 e consiste na inserção do vértice x' na pseudo-arborescência pa', para isso o algoritmo executa as remoções e inserções de arcos em pa' conforme registrado em $\mathcal{J}'$. O terceiro passo é realizado na Linha 12 e diz respeito à atualização de pa' na lista de pseudo-arborescências PA.

O Algoritmo 9 finaliza quando não existem mais candidatos a serem inseridos, conforme Linha 13, e devolve o vetor de pseudo-arborescências PA.

Somente após a cobertura de todos os clientes U_2 é realizada a chamada da função responsável por cobrir os clientes U_1. A função $Cobre\_U_1$ segue detalhada no Algoritmo 10.

Algoritmo 10: Função $Cobre\_U_1$.

Entrada: PA, I, α
Saída: PA

```
1  início
2      cobertos ← ∅;
3      faça
4          S ← {(R,T) ∈ PA : |V[R∪T] ∩ U| < Q e |V[R]| < 3};
5          se S ≠ ∅ então
6              L ← {(cl, J, pa) : cl ∈ U1 \ cobertos, (J, pa) = insere_ciclo(cl, S, I)};
7          senão
8              S ← {(R,T) ∈ PA : |V[R∪T] ∩ U| < Q};
9              L ← {(cl, J, pa) : cl ∈ U1 \ cobertos, (J, pa) = insere_pa(cl, S, I)};
10         LCR ← {cand ∈ L : c(cand) ≤ c^min + α * (c^max − c^min)};
11         (x', J', pa') ← aleatorio(LCR);
12         cobertos ← cobertos ∪ {x'};
13         Insere x' em pa' usando J';
14         Atualiza(PA, pa');
15     enquanto |L| > 1;
16     retorna PA
```

A grande diferença entre as funções $Cobre\_U_2$ e $Cobre\_U_1$ diz respeito à criação da lista de candidatos L, etapa que acontece na Linha 7 do Algoritmo 9 e entre as Linhas 5 e 9 do Algoritmo 10.

Na Linha 4 do Algoritmo 10 são selecionadas todas as pseudo-arborescências de PA que sejam inválidas e possuam menos de Q clientes. Caso ainda existam pseudo-arborescências em tal condição o algoritmo força a geração de candidatos somente para estas pseudo-arborescências. A geração de candidatos acontece na Linha 6 e utiliza a função *insere_ciclo* para o preenchimento da tripla $(cl, \mathcal{J}, pa)$.

Caso a lista de pseudo-arborescências inválidas seja vazia o algoritmo

passa a considerar, para a criação de candidatos, todas as pseudo-arborescências que ainda possuam menos de Q clientes, conforme pode ser visualizado na Linha 8. Devido ao fato dos clientes U_1 serem dispostos em qualquer local de uma pseudo-arborescência, foi criada a função *insere_pa*. Tal função participa na Linha 9 do preenchimento da tripla $(cl, \mathcal{J}, pa)$ e difere da função *insere_ciclo* pois considera a possibilidade de inserir o vértice cl em ciclos e em arborescências, além de habilitar a criação de novas arborescências. Os exemplos de inserções de vértices realizados pela função *insere_pa* podem ser vistos na Figura 4.1.

No exemplo a) da Figura 4.1 a inserção do vértice k é feita através da remoção do arco (i, j) e adição dos arcos (i,k) e (k, j). Já no exemplo b), o vértice k é incluído na pseudo-arborescência adicionando o arco (j,k) e o custo de instalação de facilidade em j. No exemplo c), o vértice i já possui custo de instalação de facilidade e a inserção de k acontece através da remoção do arco (i, j) e da adição dos arcos (i,k) e (k, j). Por fim, no exemplo d), o vértice i já possui facilidade instalada e a adição do vértice k acontece através da inclusão do arco (j,k).

As etapas seguintes do Algoritmo 10, como criação da *LCR*, seleção de candidatos e inserção do candidato selecionado, são idênticas ao apresentado no Algoritmo 9.

Após a cobertura dos clientes U_2 e U_1 é realizado a inserção dos vértices de Steiner, conforme pode ser verificado no Algoritmo 11.

Algoritmo 11: Função *Cobre_W*.

Entrada: PA, I, α
Saída: PA
1 **início**
2 $cobertos \leftarrow \emptyset$;
3 **faça**
4 $S \leftarrow \{(R,T) \in PA : |V[R]| < 3\}$;
5 **se** $S \neq \emptyset$ **então**
6 $L \leftarrow \{(cl, \mathcal{J}, pa) : cl \in W \setminus cobertos, (\mathcal{J}, pa) = insere\_ciclo(cl, S, I)\}$;
7 **senão**
8 $S \leftarrow \{(R,T) \in PA\}$;
9 $L \leftarrow \{(cl, \mathcal{J}, pa) : cl \in W \setminus cobertos, (\mathcal{J}, pa) = insere\_pa(cl, S, I)$ e $c(cl, \mathcal{J}, pa) < 0\}$;
10 $LCR \leftarrow \{cand \in L \ : \ c(cand) \leq c^{min} + \alpha * (c^{max} - c^{min})\}$;
11 $(x', \mathcal{J}', pa') \leftarrow aleatorio(LCR)$;
12 $cobertos \leftarrow cobertos \cup \{x'\}$;
13 Insere x' em pa' usando $\mathcal{J}'$;
14 Atualiza(PA, pa');
15 **enquanto** $|L| > 1$;
16 **retorna** PA

Os algoritmos *Cobre_*U_1 e *Cobre_W* são essencialmente muito parecidos uma vez que os vértices de Steiner, tal qual os clientes U_1 podem ser dispostos em ciclos ou arborescências. Dessa forma, a diferença principal entre os algoritmos pode ser identificada na Linha 9 do Algoritmo 11 quando a lista L é preenchida somente com candidatos de custo menor que zero. Isso significa que somente serão adicionados vértices de Steiner quando estes forem utilizados para validar uma pseudo-arborescência ou contribuírem para a redução no custo. As demais etapas do Algoritmo 11 são idênticas ao Algoritmo 10.

Após a cobertura dos vértices o vetor de pseudo-arborescências *PA* é armazenado em *SOL*. A heurística primal encerra quando *max_iter* é atingido.

Por fim, é retornado o vetor *SOL*, o qual contém todas as soluções válidas encontradas para a instância I.

4.2.2 Heurística de Pricing

A heurística de *pricing* proposta é constituída por um algoritmo de dois níveis, que tem por objetivo produzir, a cada iteração, pseudo-arborescências válidas de custo reduzido negativo.

Considerando a necessidade de controlar a variabilidade dos resultados, tanto o primeiro nível quanto o segundo nível da heurística foram implementados utilizando características do *GRASP*. As diferenças essenciais entre os níveis, seguem descritas abaixo:

- O primeiro nível do algoritmo é responsável por gerar um ciclo, representado por C. Este nível considera o depósito $\{d\}$ como ciclo inicial e, a cada iteração entre os níveis 1 e 2, realiza o incremento do ciclo em um vértice. Podem fazer parte de C os clientes de U_1, U_2 e os vértices de *Steiner*. A quantidade de clientes em C obedece o limitante de Q clientes;

- O segundo nível do algoritmo é responsável por gerar arborescências que apresentem, no máximo, Q_t clientes. As arborescências são construídas levando em consideração somente os vértices U_1 e W, que não fazem parte do ciclo. Ao final, o algoritmo realiza a união das arborescências geradas com o ciclo C, de forma a não violar o limitante Q.

A Figura 4.2 apresenta as iterações entre os dois níveis da heurística de *pricing*.

A heurística recebe como entrada a instância $I = (G, Q, Q_t, \overline{c}^{\mathrm{r}}, \overline{c}^{\mathrm{t}}, c^{\mathrm{f}})$, composta pelo grafo orientado G, os limitantes Q e Q_t, e três vetores de custo $\overline{c}^{\mathrm{r}}$, $\overline{c}^{\mathrm{t}}$ e c^{f} que dizem respeito, respectivamente, ao custo reduzido dos arcos quando utilizados em ciclos, custo reduzido dos arcos quando utilizados em arborescências e custo de instalação de facilidade em cada vértice.

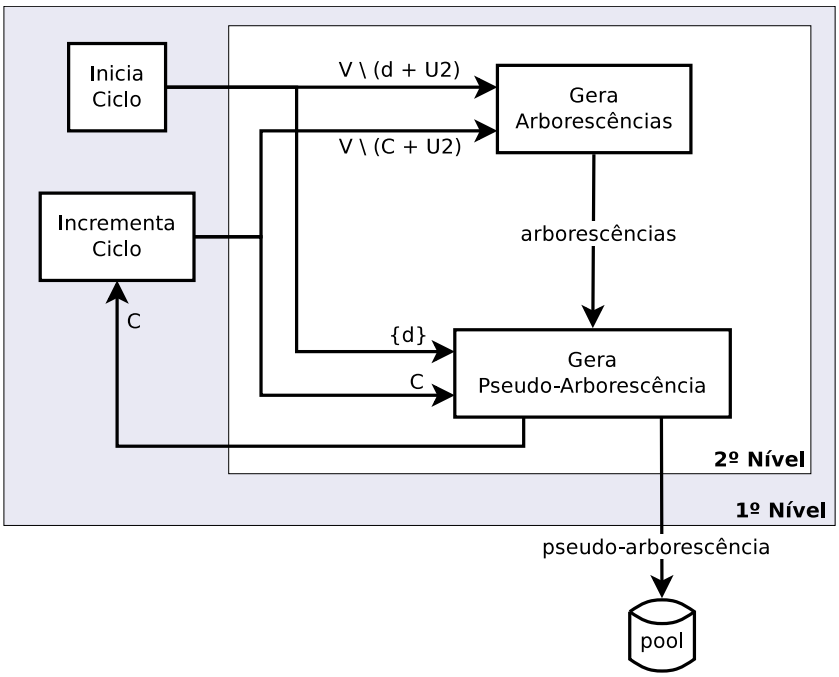

Figura 4.2: Níveis da Heurística de *Pricing*.

A entrada da heurística é composta também pelas constantes α, β e max_iter. As constantes α e β são referentes aos critérios de aleatoriedade do primeiro e segundo níveis, enquanto que max_iter diz respeito ao número de vezes que o ciclo C deve ser reiniciado.

A heurística armazena em um vetor de soluções, nominado $pool$, todas as pseudo-arborescências de custo reduzido negativo e finaliza quando max_iter é atingido. Todas as etapas do primeiro nível da heurística podem ser visualizadas no Algoritmo 12.

Na Linha 4 do Algoritmo 12 é realizado a inicialização do ciclo C, sendo todos os vértices da instância I, exceto o depósito, potenciais candidatos para compor C.

A lista de candidatos L é criada na Linha 5, de forma que cada item de L é uma tripla (v, ant, pos), sendo ant e pos os vértices em C que, respectivamente, antecederão e sucederão v, no ciclo C.

A função $heur\_arbo$, chamada nas linhas 6 e 15 do Algoritmo 12, é responsável por criar as arborescências e realizar o processo de união destas ao ciclo. No final, é retornado uma pseudo-arborescência pa que pode, ou não, ser o próprio ciclo. O Algoritmo 13 detalha as etapas de processamento de $heur\_arbo$.

Todas as pseudo-arborescências de custo reduzido negativo são armazenadas em um *pool*, conforme pode ser observado nas linhas 8 e 18 do Algoritmo 12.

O laço de iteração responsável pelo incremento de vértices no ciclo C é iniciado na Linha 9. Na Linha 10 é gerada uma lista de candidatos restrita,

Algoritmo 12: Heurística de *Pricing*.

Entrada: $I, \alpha, \beta, max\_iter$
Saída: *pool* de pseudo-arborescências

```
1  início
2      pool ← ∅;
3      para it = 0 to max_iter faça
4          C ← d;
5          L ← {(v, ant, pos) : v ∈ V \ {d}, ant = d, pos = d};
6          pa ← heur_arbo(I, C, β);
7          se c̄(pa) < 0 então
8              pool ← pool ∪ pa;
9          faça
10             LCR_r ← {cand ∈ L : imp(cand) ≤ imp^min + α * (imp^max − imp^min)};
11             (v', ant', pos') ← aleatorio(LCR_r);
12             Insere v' em C, entre os vértices ant' e pos';
13             L ← L \ (v', ant', pos');
14             L ← atualiza_candidatos(L, C, esc);
15             pa ← heur_arbo(I, C, β);
16             lsearch(pa);
17             se c̄(pa) < 0 então
18                 pool ← pool ∪ pa;
19         enquanto |C| ≤ Q;
20     retorna pool
```

nominada LCR_r, a qual é resultado da computação do impacto de cada candidato. O impacto de um candidato representa o valor da contribuição para com o custo da pseudo-arborescência, caso o vértice v seja escolhido. O valor do impacto é dado pela função *imp*, que soma o custo reduzido dos arcos adicionados e subtrai o custo reduzido do arco removido para que v se torne parte do ciclo C, conforme a fórmula: $\bar{c}^r_{(ant,v)} + \bar{c}^r_{(v,pos)} - \bar{c}^r_{(ant,pos)}$. Para o cálculo da LCR_r, ainda são utilizados imp^{min} e imp^{max} que representam o menor e maior impacto encontrado por *imp* em toda a lista L.

Na Linha 11 do Algoritmo 12 é realizada a escolha aleatória de um candidato da *LCR* e na Linha 12 o algoritmo faz a inclusão deste candidato no ciclo C. Após a inclusão, o candidato (v', ant', pos') é retirado da lista L.

A etapa adaptativa da heurística de *pricing* acontece na Linha 14 do Algoritmo 12, quando a função *atualiza_candidatos* refaz os cálculos de impacto dos candidatos em L e atualiza, sempre que necessário, os vértices *ant* e *pos*. Para a atualização dos candidatos são considerados o ciclo atual C e o último candidato inserido, v'. A Figura 4.3 demonstra as duas situações que conduzem à atualização dos vértices *ant* e *pos* dos candidatos.

No Exemplo *a*) da Figura 4.3, o vértice v possui os mesmos vértices *ant* e *pos* que o último vértice escolhido v'. Isso significa que v' foi inserido no mesmo

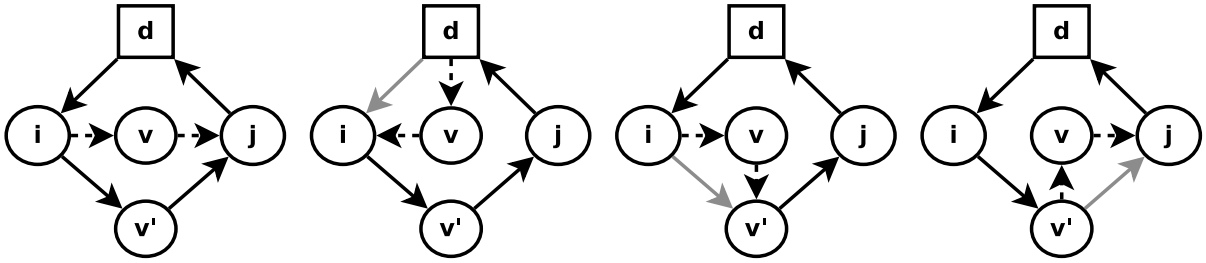

Figura 4.3: Atualização do impacto dos candidatos.

local que seria inserido v, dessa forma faz-se necessário percorrer todos os arcos de C e verificar em qual posição o vértice v gera o impacto de menor custo, atualizando então os vértices *ant* e *pos* de v.

Já no Exemplo $b.1)$ da Figura 4.3, o vértice escolhido v' não ocupou o mesmo local que o vértice v, nessa situação a função compara o menor impacto de v em três locais: no local atual, conforme Exemplo $b.1)$, entre os vértices i e v', conforme Exemplo $b.2)$, ou entre os vértices v' e j, conforme Exemplo $b.3)$. Por fim, o algoritmo atualiza os vértices *ant* e *pos* de v.

Na Linha 16 do Algoritmo 12 é realizada uma busca local, através da função *lsearch*, com o objetivo de reduzir o custo da pseudo-arborescência *pa*. A função *lsearch* baseia-se no algoritmo *2-Opt*, de Croes (1958), que realiza a troca entre duas arestas não consecutivas de um ciclo buscando diminuir o custo da solução. A diferença implementada no *lsearch* é que, devido ao ciclo ser composto de arcos orientados de um grafo G assimétrico, a troca de dois arcos implica na mudança de orientação de uma parte do ciclo, conforme pode ser observado na Figura 4.4.

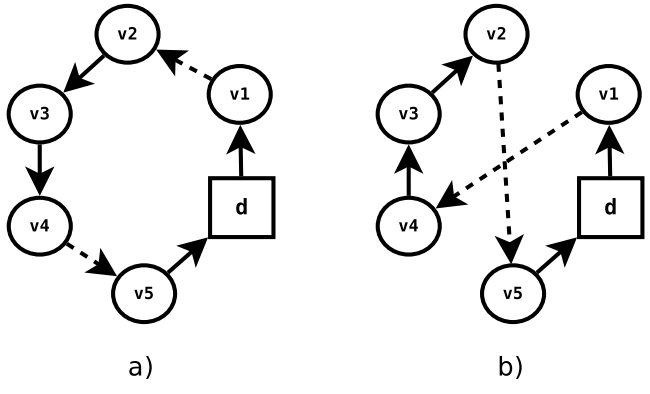

Figura 4.4: Busca local *lsearch*.

O Exemplo $a)$ da Figura 4.4 mostra um ciclo orientado composto pelos vértices $(d, v1, v2, v3, v4, v5)$ e os arcos $(v1, v2)$ e $(v4, v5)$ que serão substituídos dentro da função *lsearch*. O exemplo $b)$ mostra o ciclo após a substituição dos arcos $(v1, v2)$ e $(v4, v5)$ pelos arcos $(v1, v4)$ e $(v2, v5)$, de forma que é possível observar que o trecho do ciclo composto pelos vértices $(v2, v3, v4)$ precisou ser

alterado para $(v4, v3, v2)$. A função *lsearch* compara os custos dos exemplos $a)$ e $b)$, e aplica a mudança, caso o custo reduzido de $b)$ seja menor.

O segundo nível da heurística de *pricing* acontece dentro da função *heur_arbo* que consiste na criação de arborescências, de tamanho máximo Q_t, e na união destas ao ciclo C, com o objetivo de retornar uma pseudo-arborescência *pa*. A função *heur_arbo* segue detalhada no Algoritmo 13.

Algoritmo 13: Função *heur_arbo*.

Entrada: I, C, β
Saída: pseudo-arborescência *pa*

```
1  início
2      tam ← MIN(Qt, Q − |V(C ∩ U)|);
3      T ← GeraArb(G \ V(C ∪ U2), tam, c̄^t);
4      sol ← C;
5      faça
6          L ← {(T, K, fi) : T ∈ T e |V(T ∩ U)| + |V(sol ∩ U)| ≤ Q,
           (K, fi) = melhor_uniao(sol, T, c̄^r, c̄^t, c^f) e (c̄(K) + c̄(T) + c^f_fi) < 0};
7          se |L| > 0 então
8              LCR_t ← {cand ∈ L : sumrc(cand) ≤ sumrc^min + β * (sumrc^max − sumrc^min)};
9              (T', K', fi') ← aleatorio(LCR_t);
10             Insere T' em sol usando K' e adicionando c^f_fi', caso necessário;
11             T ← T \ {(T', K', fi')};
12     enquanto |L| > 0;
13     retorna sol
```

A primeira etapa da função *heur_arbo* consiste na definição da quantidade máxima de clientes a serem cobertos em cada arborescência. Tal avaliação é necessária para limitar a geração de arborescências com mais de $Q - V(C \cap U) < Q_t$ clientes. Assim, o algoritmo reduz a quantidade de clientes em cada arborescência de forma a possibilitar a união destas ao ciclo sem extrapolar o limitante Q, conforme pode ser verificado na Linha 2 do Algoritmo 13.

A fase de construção das arborescências acontece na Linha 3 do Algoritmo 13, sendo realizada pela função *GeraArb* que recebe como entrada o subgrafo de G, que contém somente os vértices que ainda podem ser cobertos por arborescências, $G \setminus V(C \cup U_2)$, a quantidade máxima de clientes em cada arborescência, *tam*, e o vetor de custos dos arcos quando utilizados em arborescências, $\bar{c}^{\mathrm{t}}$. Todas as arborescências geradas são armazenadas no vetor $\mathcal{T}$. O detalhamento da função *GeraArb* será realizado ao final deste capítulo.

Na Linha 4 do Algoritmo 13 o ciclo C, recebido do primeiro nível, é utilizado como pseudo-arborescência inicial. A partir da combinação da pseudo-arborescência *sol* com as arborescências armazenadas em $\mathcal{T}$, é gerada uma lista de candidatos L. Enquanto existirem candidatos possíveis, o laço de

iteração responsável por gerar, selecionar e inserir tais candidatos é repetido entre as linhas 5 e 12 do Algoritmo 13.

A construção da lista de candidatos L acontece na Linha 6 do Algoritmo 13, de forma que cada candidato em L é uma tripla composta por uma arborescência T, um conjunto de arcos $\mathcal{K}$ e um vértice fi. O conjunto $\mathcal{K}$ representa os arcos que devem ser adicionados e/ou removidos para que a arborescência T faça parte da pseudo-arborescência sol. O vértice fi é utilizado quando se faz necessário instalar uma facilidade na união de T e sol. Para que um candidato seja adicionado na lista L é necessário que a soma das quantidades de clientes da arborescência T e da pseudo-arborescência sol não ultrapasse o limite de Q clientes. Além disso, é necessário que a soma do custo de instalação de facilidade em fi com os custos reduzidos de T e $\mathcal{K}$, seja negativo.

A função *melhor_uniao*, responsável por retornar $\mathcal{K}$, percorre todos os arcos que compõe o ciclo em sol e retorna o conjunto $\mathcal{K}$ que representa o melhor custo reduzido para a inserção da arborescência T em sol. A Figura 4.5 ilustra as possíveis formas de inserção de uma arborescência T, com raiz em k, testadas pela função *melhor_uniao*.

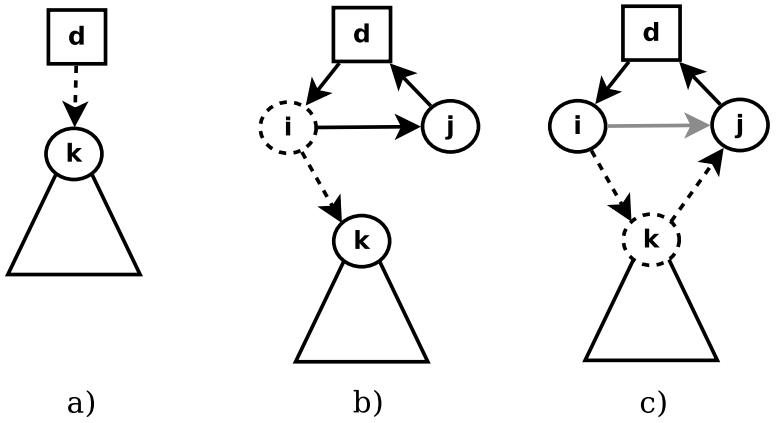

Figura 4.5: Inserção de arborescências.

No exemplo a) da Figura 4.5, a pseudo-arborescência sol é composta unicamente pelo depósito d, dessa maneira, a função *melhor_uniao* retorna somente o arco (d,k) e calcula o custo reduzido como $\overline{c}^{\mathrm{t}}_{(d,k)}$. Caso o ciclo seja composto por mais de um vértice, são testadas, para cada arco do ciclo, duas possibilidades de inserção de T em sol, conforme demonstrado nos exemplos b) e c). No exemplo b), a função retorna o arco (i,k) e o vértice i, caso i ainda não tenha facilidade instalada. Assim, o custo reduzido do exemplo b) é dado pela fórmula $\overline{c}^{\mathrm{t}}_{(i,k)} + c^{\mathrm{f}}_{i}$. Já no exemplo c) a função *melhor_uniao* retorna para $\mathcal{K}$ os arcos que devem ser adicionados (i,k), (k,j) e o arco a ser removido (i,j), sendo retornado para fi, o vértice k. Dessa maneira, o custo reduzido do exemplo c) é dado pela fórmula $\overline{c}^{\mathrm{r}}_{(i,k)} + \overline{c}^{\mathrm{r}}_{(k,j)} - \overline{c}^{\mathrm{r}}_{(i,j)} + c^{\mathrm{f}}_{k}$.

Caso exista pelo menos um candidato em L para compor a pseudo-arborescência sol, o algoritmo gera, na Linha 8 do Algoritmo 13, uma lista de candidatos restrita, nominada LCR_t. São implementados na geração da lista LCR_t, os mesmos conceitos utilizados na Linha 10 do Algoritmo 12, porém ao invés de considerar somente o impacto de cada candidato, a função $sumrc$ retorna a soma dos custos do trio $(T, \mathcal{K}, fi)$, ou seja $\overline{c}(T) + \overline{c}(\mathcal{K}) + c^{\mathrm{f}}_{fi}$. Para o cálculo do custo reduzido de $\mathcal{K}$ é levado em consideração se os arcos adicionados/removidos são arcos do ciclo ou da arborescência.

Na Linha 9 do Algoritmo 13 a função $aleatorio$ seleciona um candidato esc da LCR_t. Na Linha 10 o candidato $(T', \mathcal{K}', fi')$ é inserido na solução e logo após é removido da lista $\mathcal{T}$, conforme Linha 11.

A etapa de geração de arborescências, realizada pela função $GeraArb$ e utilizada na Linha 3 do Algoritmo 13, tem como objetivo retornar até $|V(G) \setminus V(C \cup U_2)|$ arborescências de custo reduzido mínimo de forma que cada arborescência possua no máximo tam clientes. Tal função é uma adaptação do algoritmo de Edmonds (1967) uma vez que este produz uma única arborescência geradora de custo mínimo.

A função $GeraArb$ segue detalhada no Algoritmo 14. A etapa inicial desse algoritmo consiste na ordenação dos arcos do grafo G, conforme Linha 3. A ordenação é realizada pela função $OrdenaCrescente$ que recebe os arcos do grafo G' e seus respectivos custos reduzidos de utilização em arborescências, $\overline{c}^{\mathrm{t}}$, e devolve um vetor de arcos ordenados Arc.

Algoritmo 14: Função $GeraArb$.

Entrada: $G', tam, \overline{c}^{\mathrm{t}}$
Saída: Lista $\mathcal{T}$

```
1  início
2      Arcos ← ∅;
3      Arc ← OrdenaCrescente(A(G'), c̄ᵗ);
4      para cada v ∈ V(G') faça
5          S[v] ← v;
6          se v ∈ U então
7              U[v] ← 1;
8          senão
9              U[v] ← 0;
10     para cada arco (i, j) em Arc em ordem crescente faça
11         se (S[j] = j) e (S[i] ≠ S[j]) e (U[S[i]] + U[S[j]] ≤ tam) então
12             U[S[i]] ← U[S[i]] + U[S[j]];
13             AtualizaRaiz(S, i, j);
14             Arcos ← Arcos ∪ {i, j};
15     T ← ReconstroiArborescencias(Arcos, S);
16     retorna T
```

Após a ordenação dos arcos é realizada, entre as Linhas 4 e 9, a criação de arborescências unitárias, uma para cada vértice do grafo, e a contagem da quantidade de clientes em cada arborescência. Essa quantidade é armazenada no vetor T.

Na Linha 10 do Algoritmo 14 a função *GeraArb* percorre todos os arcos em *Arc*, com o objetivo de atualizar as raízes das arborescências e separar os arcos utilizados. O algoritmo mantém um vetor S, no qual $S[i]$ representa a raiz da árvore que contém o vértice i. Os três principais critérios condicionais para a criação das arborescências são apresentados na Linha 11 e detalhados conforme abaixo:

a) $S[j] = j$: Esta condição verifica se o vértice j, do arco a ser utilizado na união de arborescências, é a raiz de uma arborescência;

b) $S[i] \neq S[j]$: Esta condição impede a utilização de arcos que fechem ciclos em arborescências, uma vez que impede a utilização de arcos que conectem dois vértices de mesma raiz;

c) $U[S[i]] + U[S[j]] \leq tam$: Esta condição impede que a soma da quantidade de clientes das arborescências, dadas pelo vetor U, ultrapasse o limite *tam*.

Caso as três condições citadas sejam atendidas, as arborescências, representadas pelas raízes dos vértices do arco (i, j), podem ser unidas. A primeira etapa da união acontece na Linha 12 e consiste na atualização da quantidade de clientes da arborescência com raiz em $S[i]$. A segunda etapa da união é realizada na Linha 13, sendo a função *AtualizaRaiz* responsável por percorrer todos vértices da arborescência com raiz em j e substituir estas raízes para i. A terceira etapa da união acontece na Linha 14, sendo o arco (i, j) armazenado no vetor *Arcos*.

Na Linha 15 do Algoritmo 14 são realizados a construção e o armazenamento das arborescências. Esta etapa é realizada pela função *ReconstroiArborescencias* que considera cada raiz distinta em S como uma arborescência em $\mathcal{T}$. Além disso, a função percorre todos os arcos armazenados em *Arcos* para popular os vértices e arcos de cada arborescência em $\mathcal{T}$.

A Figura 4.6 mostra exemplos de arborescências que podem ser encontradas em $\mathcal{T}$ a partir da aplicação do Algoritmo 14.

O Algoritmo 14 finaliza ao devolver o vetor de pseudo-arborescências $\mathcal{T}$.

Por fim, a heurística de *pricing* retorna um *pool* de pseudo-arborescências com custo reduzido negativo, construídas a partir de uma instância I e parametrizada pelas constantes α, β e *max_iter*.

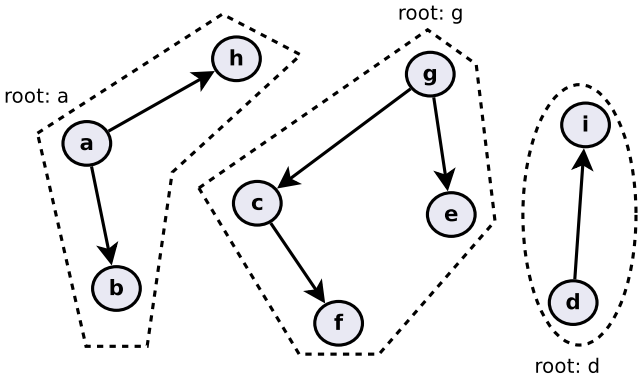

Figura 4.6: Grafo com 9 vértices e 3 arborescências.

CAPÍTULO

5

Resultados

Neste capítulo serão apresentados as instâncias de testes utilizadas e os resultados obtidos com a implementação dos algoritmos exatos, baseados nas formulações (MCF) e (SPF), e das implementações da heurística primal e da heurística de *pricing*.

5.1 Instâncias de Testes

Devido à inexistência de bibliografia referente ao CPAFLP e, consequentemente, de instâncias de testes, optou-se por adaptar as instâncias do CRTP propostas em Hill (2012), para obter 4 grupos de testes que combinam variações no custo de instalação de facilidade (CF) e limite de clientes por facilidade (QT), conforme segue:

- **CF0_QT1:** Todas as instâncias neste grupo possuem vértices com custo de instalação de facilidade igual a 0 e o tamanho máximo de cada arborescência é limitado a 1 cliente;
- **CF0_QT66:** Todas as instâncias neste grupo possuem vértices com custo de instalação de facilidade igual a 0 e o tamanho máximo de cada arborescência é limitado a, aproximadamente, 66% ($\lfloor Q/3 \rfloor \cdot 2$) do tamanho da pseudo-arborescência;
- **CFMED_QT1:** Em todas as instâncias neste grupo o tamanho máximo de cada arborescência é limitado a 1 cliente e os vértices possuem custo de instalação de facilidade igual à média entre o custo do arco mais barato e do arco mais caro da instância;

- **CFMED_QT66:** Em todas as instâncias neste grupo o tamanho máximo de cada arborescência é limitado a, aproximadamente, 66% ($\lfloor Q/3 \rfloor \cdot 2$) do tamanho da pseudo-arborescência e os vértices possuem custo de instalação de facilidade igual à média entre o custo do arco mais barato e do arco mais caro da instância.

Foram escolhidas 45 instâncias que apresentam variações na distribuição da quantidade de cada tipo de vértice, com exceção do depósito que é unitário. Todas as instâncias utilizadas são compostas por grafos simétricos de 26 vértices e 650 arcos. Os custos de utilização dos arcos em arborescências (c_a^{t}) e utilização dos arcos em ciclos (c_a^{r}) foram configurados de maneira que $c_a^{\text{t}} = \lfloor c_a^{\text{r}}/2 \rfloor$ para cada arco a do grafo $G = (V, A)$.

Para fins de organização, a nomenclatura das instâncias obedece o padrão R_**XXX**_A**YY**, de maneira que **XXX** representa a proporção aproximada de vértices U_1 em relação a quantidade total de clientes $(U_1 + U_2)$ e **YY** representa a sequência numérica da instância. Os nomes, quantidades de vértices por tipo, limitantes m e Q das instâncias utilizadas, podem ser visualizados na Tabela 5.1.

Instância	$\lvert U_1 \rvert$	$\lvert U_2 \rvert$	$\lvert W \rvert$	m	Q
R_000_A01	0	12	13	3	5
R_000_A02	0	12	13	4	4
R_000_A03	0	12	13	5	3
R_000_A04	0	18	7	3	7
R_000_A05	0	18	7	4	5
R_000_A06	0	18	7	5	4
R_000_A07	0	25	0	3	10
R_000_A08	0	25	0	4	7
R_000_A09	0	25	0	5	6
R_025_A01	3	9	13	3	5
R_025_A02	3	9	13	4	4
R_025_A03	3	9	13	5	3
R_025_A04	5	13	7	3	7
R_025_A05	5	13	7	4	5
R_025_A06	5	13	7	5	4
R_025_A07	7	18	0	3	10
R_025_A08	7	18	0	4	7
R_025_A09	7	18	0	5	6
R_050_A01	6	6	13	3	5
R_050_A02	6	6	13	4	4
R_050_A03	6	6	13	5	3
R_050_A04	9	9	7	3	7
R_050_A05	9	9	7	4	5
R_050_A06	9	9	7	5	4
R_050_A07	13	12	0	3	10
R_050_A08	13	12	0	4	7

continua na próxima página.

Tabela 5.1 – continuação.

Instância	$\|U_1\|$	$\|U_2\|$	$\|W\|$	m	Q
R_050_A09	13	12	0	5	6
R_075_A01	9	3	13	3	5
R_075_A02	9	3	13	4	4
R_075_A03	9	3	13	5	3
R_075_A04	14	4	7	3	7
R_075_A05	14	4	7	4	5
R_075_A06	14	4	7	5	4
R_075_A07	19	6	0	3	10
R_075_A08	19	6	0	4	7
R_075_A09	19	6	0	5	6
R_100_A01	12	0	13	3	5
R_100_A02	12	0	13	4	4
R_100_A03	12	0	13	5	3
R_100_A04	18	0	7	3	7
R_100_A05	18	0	7	4	5
R_100_A06	18	0	7	5	4
R_100_A07	25	0	0	3	10
R_100_A08	25	0	0	4	7
R_100_A09	25	0	0	5	6

Tabela 5.1: Nomenclatura e configuração das instâncias utilizadas.

5.2 Definição dos Parâmetros

Considerando o grande número de combinações possíveis entre os parâmetros α e *max_iter*, apresentados no Capítulo 4 e presentes tanto na heurística primal quanto na heurística de *pricing*, e tendo em vista ainda o extenso tempo necessário para a execução de cada grupo de instâncias, fez-se necessário aprofundar o estudo acerca da melhor escolha dos referidos parâmetros.

Para os testes foram selecionadas 15 instâncias de cada um dos 4 grupos de testes, o que correspondeu a 33,33% do total de instâncias. O critério de seleção considerou as variações na proporção de clientes do tipo U_1 e a quantidade de vértices de Steiner (W). As instâncias selecionadas e suas respectivas configurações podem ser visualizadas na Tabela 5.2.

Os testes foram processados utilizando uma implementação, em linguagem C, da heurística primal proposta na Subseção 4.2.1 do Capítulo 4. O programa recebeu como entrada a constante *max_iter*, o valor α e a instância a ser executada. A saída do programa continha o custo da melhor solução encontrada e o tempo consumido. Considerou-se como melhor solução a solução de menor custo.

Optou-se por utilizar a heurística primal ao invés da heurística de *pricing*

Instância	$\|U_1\|$	$\|U_2\|$	$\|W\|$	m	Q
R_000_A01	0	12	13	3	5
R_000_A05	0	18	7	4	5
R_000_A09	0	25	0	5	6
R_025_A01	3	9	13	3	5
R_025_A05	5	13	7	4	5
R_025_A09	7	18	0	5	6
R_050_A01	6	6	13	3	5
R_050_A05	9	9	7	4	5
R_050_A09	13	12	0	5	6
R_075_A01	9	3	13	3	5
R_075_A05	14	4	7	4	5
R_075_A09	19	6	0	5	6
R_100_A01	12	0	13	3	5
R_100_A05	18	0	7	4	5
R_100_A09	25	0	0	5	6

Tabela 5.2: Conjunto de instâncias utilizadas para escolha do α.

devido ao fato da primeira trabalhar diretamente com os dados de entrada do problema, ao passo que a heurística de *pricing* necessitava de valores duais provenientes da execução de um Problema Mestre Restrito (PMR).

A execução dos testes ocorreu em uma máquina com processador Intel i3 3.3Ghz com 8GB de memória RAM e sistema operacional Linux. Todas as instâncias selecionadas foram testadas com 11 valores de α, variando entre 0,0 e 1,0, e com max_iter parametrizado em 1000 iterações. Para cada instância testada extraiu-se a melhor solução encontrada, conforme pode ser visualizado na Tabela 5.3.

Após a extração das melhores soluções de cada instância, foi realizado uma comparação dos melhores resultados com os encontrados em cada variação do α. A Tabela 5.4 sumariza este comparativo.

Na Tabela 5.4, a coluna *Proximidade(%)* é referente à distância do valor da solução em comparação ao melhor resultado encontrado na Tabela 5.3, já a coluna *Melhor(%)* diz respeito à porcentagem de soluções encontradas pelo respectivo valor de α que são iguais ao melhor resultado. Por fim, a coluna *Média(s)* é referente ao tempo médio, em segundos, de execução das instâncias.

Conforme apresentado na Tabela 5.4, os experimentos com um conjunto reduzido de instâncias indicam que os melhores resultados são obtidos quando o valor de α é $0,6$. Optou-se por utilizar max_iter igual a 1000 iterações por conta do pouco tempo consumido pela heurística que, no pior dos casos, apresentou um tempo médio de $0,03s$.

Observando as semelhanças na forma de utilização das constantes α e

Instância	CF0_QT1	CF0_QT66	CFMED_QT1	CFMED_QT66
R_000_A01	242	242	242	242
R_000_A05	340	339	340	345
R_000_A09	405	405	407	403
R_025_A01	229	229	242	242
R_025_A05	329	329	346	346
R_025_A09	362	358	372	357
R_050_A01	233	233	258	258
R_050_A05	327	318	346	345
R_050_A09	364	320	385	322
R_075_A01	226	223	242	242
R_075_A05	324	314	354	352
R_075_A09	341	286	390	307
R_100_A01	180	139	242	198
R_100_A05	276	258	339	339
R_100_A09	322	220	380	288

Tabela 5.3: Melhor resultado da heurística primal em cada instância.

Alpha	Proximidade(%)	Melhor(%)	Média(s)
0,6	1,1%	52%	0,03
0,7	1,3%	47%	0,03
0,5	1,4%	35%	0,03
0,4	1,6%	40%	0,03
0,3	1,6%	35%	0,03
0,8	1,7%	45%	0,03
1,0	2,1%	42%	0,03
0,9	2,1%	35%	0,03
0,2	3,2%	18%	0,02
0,1	4,6%	12%	0,02
0,0	18,9%	0%	0,02

Tabela 5.4: Desempenho do α em 1000 iterações.

max_iter, presentes tanto na heurística primal quanto heurística de *pricing*, e considerando os resultados obtidos nos experimentos com o conjunto reduzido de instâncias, decidiu-se por utilizar os mesmos valores de α e *max_iter* para as duas heurísticas desenvolvidas.

Na heurística de *pricing* optou-se por utilizar o mesmo valor nas duas constantes, α e β, que correspondem ao critério de variabilidade do algoritmo.

5.3 Resultados dos Testes

Os experimentos foram realizados com o objetivo de obter a solução ótima do problema.

A implementação foi feita em linguagem C com o framework SCIP, descrito em Achterberg (2009), e utilizando o resolvedor CPLEX de Mittelmann (2007). A heurística primal desenvolvida foi utilizada para fornecer uma base inicial ao algoritmo e um limitante superior, enquanto que a heurística de *pricing* busca agilizar a criação de colunas de custo reduzido negativo.

Todas as instâncias foram executadas em uma máquina com processador Intel i7 3.6Ghz com 8GB de memória e sistema operacional Linux. Foram desenvolvidos cinco programas que combinam os modelos matemáticos (MCF) e (SPF), além das heurísticas primal e de *pricing*. As características dos programas podem ser visualizadas na Tabela 5.5.

Programa	**SPF-**	**SPF**	**SPF***	**MCF-**	**MCF**
Modelo	(SPF)	(SPF)	(SPF)	(MCF)	(MCF)
Algoritmo	*B&P*	*B&P*	*B&P*	*B&B*	*B&B*
Heurística	-	Primal	Primal + *Pricing*	-	Primal

Tabela 5.5: Características dos programas.

O detalhamento das características de cada programa da Tabela 5.5 pode ser conferido abaixo:

- **SPF-:** Este programa utiliza o modelo matemático (SPF) e é um algoritmo de *branch-and-price* que faz uso somente da técnica de *pricing* exato, não utiliza heurísticas para povoar a base inicial e não possui uma solução inicial;

- **SPF:** Este programa utiliza o modelo matemático (SPF) e é um algoritmo de *branch-and-price* que faz uso somente da técnica de *pricing* exato, utiliza a heurística primal para povoar a base e fornecer uma solução inicial;

- **SPF*:** Este programa utiliza o modelo matemático (SPF) e é um algoritmo de *branch-and-price* que faz uso do *pricing* exato somente quando a heurística de *pricing* falha em prover uma solução de custo reduzido negativo, utiliza a heurística primal para povoar a base e fornecer uma solução inicial;

- **MCF-:** Este programa utiliza o modelo matemático (MCF) e é um algoritmo de *branch-and-bound*, não possui uma solução inicial;

- **MCF:** Este programa utiliza o modelo matemático (MCF) e é um algoritmo de *branch-and-bound*, utiliza a heurística primal para fornecer uma solução inicial.

Com o objetivo de avaliar a qualidade do limitante inferior no nó raiz encontrado pelas formulações (MCF) e (SPF) foi realizada a comparação do *GAP* encontrado no nó raiz em cada instância nos 4 grupos de instâncias. O *GAP* de cada formulação é dado por:

$$(bestUB - rootLB)/bestUB, \tag{5.1}$$

sendo $rootLB$ o limitante inferior encontrado no nó raiz pelos programas baseados na respectiva formulação e $bestUB$ o menor limitante superior encontrado na execução dos cinco programas. Os resultados podem ser visualizados na Tabela 5.6.

	CF0_QT1		CF0_QT66		CFMED_QT1		CFMED_QT66	
Instância	(SPF)	(MCF)	(SPF)	(MCF)	(SPF)	(MCF)	(SPF)	(MCF)
R_000_A01	2,15	21,49	2,15	21,49	2,15	21,49	2,15	21,49
R_000_A02	0	24,30	0	24,30	0	24,30	0	24,30
R_000_A03	0	31,76	0	31,76	0	31,76	0	31,76
R_000_A04	2,86	21,43	*	*	*	*	2,86	21,43
R_000_A05	0,65	30,18	0,94	30,38	0,65	30,18	0,65	30,18
R_000_A06	1,73	36,83	1,73	36,83	1,73	36,83	1,73	36,83
R_000_A07	*	*	*	*	*	*	*	*
R_000_A08	3,68	24,38	4,71	25,19	3,41	24,17	3,15	23,97
R_000_A09	5,84	29,42	6,08	29,60	5,61	29,24	6,08	29,60
R_025_A01	3,28	20,52	*	*	5,41	23,88	*	*
R_025_A02	0	25,42	1,04	25,42	0,92	27,51	1,65	25,72
R_025_A03	1,26	31,41	0	29,10	1,08	31,41	0	29,10
R_025_A04	*	*	*	*	*	*	*	*
R_025_A05	2,13	31,25	5,47	31,55	3,32	33,96	5,23	33,96
R_025_A06	4,66	37,89	0	32,73	2,88	39,01	0,57	35,03
R_025_A07	*	*	*	*	*	*	*	*
R_025_A08	*	*	*	*	*	*	*	*
R_025_A09	0,69	27,70	*	*	0	27,97	0	25,55
R_050_A01	*	*	*	*	14,34	33,76	0	16,63

continua na próxima página.

Tabela 5.6 – continuação.

Instância	CF0_QT1 (SPF)	CF0_QT1 (MCF)	CF0_QT66 (SPF)	CF0_QT66 (MCF)	CFMED_QT1 (SPF)	CFMED_QT1 (MCF)	CFMED_QT66 (SPF)	CFMED_QT66 (MCF)
R_050_A02	0,99	27,31	0	22,90	4,54	34,26	1,38	25,14
R_050_A03	7,28	38,54	0	25,71	6,18	42,76	0	25,14
R_050_A04	*	*	*	*	*	*	*	*
R_050_A05	*	*	*	*	4,70	36,84	*	*
R_050_A06	4,66	40,17	1,18	34,47	6,05	45,97	0	37,56
R_050_A07	*	*	*	*	*	*	*	*
R_050_A08	*	*	*	*	*	*	*	*
R_050_A09	*	*	3,59	23,94	0	32,63	3,73	23,39
R_075_A01	*	*	*	*	*	*	*	*
R_075_A02	*	*	*	*	*	*	*	*
R_075_A03	5,24	37,78	0	22,09	3,01	41,69	0	23,15
R_075_A04	*	*	*	*	*	*	*	*
R_075_A05	*	*	*	*	*	*	*	*
R_075_A06	8,19	47,66	2,29	39,48	5,97	55,04	*	*
R_075_A07	*	*	*	*	*	*	*	*
R_075_A08	*	*	*	*	*	*	*	*
R_075_A09	*	*	*	*	*	*	*	*
R_100_A01	*	*	*	*	*	*	*	*
R_100_A02	3,89	52,44	0	42,67	2,41	63,88	*	*
R_100_A03	8,27	61,09	0	44,32	4,14	67,82	0	53,09
R_100_A04	*	*	*	*	*	*	*	*
R_100_A05	*	*	*	*	*	*	*	*
R_100_A06	11,02	61,37	1,79	57,46	7,68	68,31	*	*
R_100_A07	*	*	*	*	*	*	*	*
R_100_A08	*	*	*	*	*	*	*	*
R_100_A09	*	*	13,76	47,86	*	*	*	*
Média	3,57	34,56	2,13	32,35	3,59	37,69	1,46	28,64

Tabela 5.6: *GAP* do LB no nó raiz das formulações (SPF) e (MCF).

Para a construção Tabela 5.6 foram considerados somente *GAP*s encontrados, no nó raiz, em ambas formulações para uma mesma instância. As colunas (SPF) dizem respeito ao menor *GAP* encontrado após a execução dos programas SPF-, SPF e SPF*. As colunas (MCF) são referentes ao *GAP* encontrado após a execução dos programas MCF- e MCF. Instâncias que não obtiveram *GAP* em ambas colunas foram marcadas com *.

Verificando a Tabela 5.6 é possível observar que a média do *GAP* no nó raiz dos programas baseados no modelo (SPF) foi de $2,69\%$ contra $33,31\%$ dos programas baseados no modelo (MCF), além disso os programas baseados no modelo (SPF) foram capazes de encontrar a solução ótima para 25 das 87 instâncias comparadas.

Os resultados obtidos com a execução dos cinco programas nos grupos de instâncias CF0_QT1, CF0_QT66, CFMED_QT1 e CFMED_QT66 são apresentados, respectivamente, na Tabela 5.7, Tabela 5.8, Tabela 5.9 e

Tabela 5.10. Tais tabelas apresentam o mesmo padrão de organização, de forma que:

- As colunas **rootLB** dizem respeito ao valor do limitante inferior obtido no nó raiz por cada programa;
- As colunas **LB** são relativas ao valor do limitante inferior final obtido por cada programa;
- As colunas **UB** correspondem ao valor do limitante superior final obtido por cada programa;
- As colunas **Tempo** referem-se ao tempo total utilizado por cada programa para encontrar a solução ótima. Quando o programa não encontra a solução ótima no tempo limite de 1800 segundos, a coluna tempo é assinalada com -;
- O símbolo * é usado para indicar que o valor da coluna não pode ser encontrado no limite de tempo de 1800 segundos.

Instância	rootLB					LB					UB					Tempo				
	SPF-	SPF	SPF*	MCF-	MCF	SPF-	SPF	SPF*	MCF-	MCF	SPF-	SPF	SPF*	MCF-	MCF	SPF-	SPF	SPF*	MCF-	MCF
R_000_A01	*	236,8	236,8	190,0	190,0	*	236,8	237,4	220,0	215,4	*	242	242	242	242	-	-	-	-	-
R_000_A02	251,0	251,0	251,0	190,0	190,0	251,0	251,0	251,0	219,7	223,7	251	251	251	251	251	1332,7	203,9	246,7	-	-
R_000_A03	279,0	279,0	279,0	190,4	190,4	279,0	279,0	279,0	226,6	229,8	279	279	279	279	279	386,8	180,5	68,6	-	-
R_000_A04	*	*	292,4	236,5	236,5	*	*	292,4	253,5	254,3	*	301	301	357	301	-	-	-	-	-
R_000_A05	*	336,8	336,8	236,7	236,7	*	339,0	338,0	258,2	263,2	*	339	339	477	350	-	737,6	-	-	-
R_000_A06	*	368,5	368,5	236,9	236,9	*	372,0	370,0	257,6	260,9	*	375	375	510	388	-	-	-	-	-
R_000_A07	*	*	*	279,0	279,0	*	*	*	292,6	296,4	*	328	328	353	329	-	-	-	-	-
R_000_A08	*	*	356,4	279,8	279,8	*	*	356,4	284,5	290,9	*	370	370	498	374	-	-	-	-	-
R_000_A09	*	373,8	373,8	280,2	280,2	*	374,8	375,3	295,1	296,5	*	397	405	488	416	-	-	-	-	-
R_025_A01	*	*	221,5	182,0	182,0	*	*	221,5	199,5	204,0	270	229	229	229	229	-	-	-	-	-
R_025_A02	*	240,0	240,0	179,0	179,0	*	240,0	240,0	205,3	204,9	240	240	240	240	240	-	1080,5	1113,0	-	-
R_025_A03	273,5	273,5	273,5	190,0	190,0	274,0	274,0	275,0	222,9	226,3	277	292	289	277	277	-	-	-	-	-
R_025_A04	*	*	*	221,0	221,0	*	*	*	240,8	241,5	*	289	289	301	289	-	-	-	-	-
R_025_A05	*	321,0	321,0	225,5	225,5	*	321,0	323,5	248,0	245,5	*	331	328	418	329	-	-	-	-	-
R_025_A06	*	348,0	348,0	226,7	226,7	*	348,0	348,0	256,5	251,1	*	365	371	408	371	-	-	-	-	-
R_025_A07	*	*	*	259,5	259,5	*	*	*	268,2	272,8	*	315	317	323	312	-	-	-	-	-
R_025_A08	*	*	*	255,3	255,3	*	*	*	259,1	271,9	*	340	340	*	345	-	-	-	-	-
R_025_A09	*	358,5	358,5	261,0	261,0	*	358,5	358,5	268,5	272,4	*	361	361	*	369	-	-	-	-	-
R_050_A01	*	*	*	167,8	167,8	*	*	*	188,7	191,7	*	233	233	219	214	-	-	-	-	-
R_050_A02	*	*	220,8	162,1	162,1	*	*	220,8	189,2	192,4	*	223	223	227	223	-	-	-	-	-
R_050_A03	*	235,5	235,5	156,1	156,1	*	235,5	235,5	184,2	188,3	384	254	254	279	254	-	-	-	-	-
R_050_A04	*	*	*	198,0	198,0	*	*	*	215,5	213,5	*	274	274	302	274	-	-	-	-	-
R_050_A05	*	*	*	215,2	215,2	*	*	*	225,8	226,7	*	325	323	365	327	-	-	-	-	-
R_050_A06	*	*	327,0	205,2	205,2	*	*	327,0	221,1	224,3	343	356	376	414	359	-	-	-	-	-
R_050_A07	*	*	*	238,6	238,6	*	*	*	248,4	251,1	*	290	290	316	290	-	-	-	-	-
R_050_A08	*	*	*	226,0	226,0	*	*	*	234,7	239,7	*	315	318	*	318	-	-	-	-	-
R_050_A09	*	*	*	244,6	244,6	*	*	*	250,9	257,8	*	364	369	*	366	-	-	-	-	-
R_075_A01	*	*	*	158,3	158,3	*	*	*	168,7	173,8	*	226	226	215	214	-	-	-	-	-
R_075_A02	*	*	*	143,4	143,4	*	*	*	158,6	157,5	*	211	211	232	211	-	-	-	-	-
R_075_A03	*	235,0	235,0	154,3	154,3	*	235,0	235,0	174,8	174,1	381	267	256	248	276	-	-	-	-	-
R_075_A04	*	*	*	175,2	175,2	*	*	*	182,9	182,8	*	248	263	271	263	-	-	-	-	-
R_075_A05	*	*	*	185,9	185,9	*	*	*	197,2	197,1	*	326	324	327	328	-	-	-	-	-
R_075_A06	*	*	294,7	168,0	168,0	*	*	294,7	173,6	177,4	*	321	336	388	332	-	-	-	-	-
R_075_A07	*	*	*	215,8	215,8	*	*	*	221,2	224,2	*	297	295	*	296	-	-	-	-	-
R_075_A08	*	*	*	212,3	212,3	*	*	*	213,8	215,6	*	340	332	*	338	-	-	-	-	-
R_075_A09	*	*	*	206,1	206,1	*	*	*	206,3	209,6	*	335	334	*	342	-	-	-	-	-
R_100_A01	*	*	*	86,3	86,3	*	*	*	100,1	98,4	*	185	180	195	187	-	-	-	-	-
R_100_A02	*	173,0	173,0	85,6	85,6	*	173,0	173,0	97,9	97,9	*	180	180	190	190	-	-	-	-	-
R_100_A03	*	201,8	201,8	85,6	85,6	*	201,8	201,8	97,7	97,7	269	220	220	231	231	-	-	-	-	-
R_100_A04	*	*	*	123,3	123,3	*	*	*	131,0	132,7	*	225	225	*	241	-	-	-	-	-
R_100_A05	*	*	*	121,8	121,8	*	*	*	130,0	130,8	*	292	286	*	288	-	-	-	-	-
R_100_A06	*	280,3	280,3	121,7	121,7	*	280,3	280,3	128,4	130,4	458	320	315	*	331	-	-	-	-	-
R_100_A07	*	*	*	159,4	159,4	*	*	*	166,3	163,6	*	275	262	*	276	-	-	-	-	-
R_100_A08	*	*	*	156,4	156,4	*	*	*	164,2	165,0	*	313	317	*	333	-	-	-	-	-
R_100_A09	*	*	*	155,0	155,0	*	*	*	163,1	163,7	*	343	337	*	328	-	-	-	-	-

Tabela 5.7: Resultados do grupo de instâncias CF0_QT1.

	rootLB					LB					UB					Tempo				
Instância	SPF-	SPF	SPF*	MCF-	MCF	SPF-	SPF	SPF*	MCF-	MCF	SPF-	SPF	SPF*	MCF-	MCF	SPF-	SPF	SPF*	MCF-	MCF
R_000_A01	*	236,8	236,8	190,0	190,0	*	236,8	237,4	219,8	219,8	*	242	242	242	242	-	-	-	-	-
R_000_A02	251,0	251,0	251,0	190,0	190,0	251,0	251,0	251,0	214,3	223,9	251	251	251	251	251	1319,1	210,0	251,3	-	-
R_000_A03	279,0	279,0	279,0	190,4	190,4	279,0	279,0	279,0	228,1	227,1	279	279	279	279	279	387,1	74,1	50,8	-	-
R_000_A04	*	*	*	236,5	236,5	*	*	*	255,9	261,1	*	301	301	312	301	-	-	-	-	-
R_000_A05	*	336,8	336,8	236,7	236,7	*	337,8	338,2	262,3	258,9	*	340	345	409	345	-	-	-	-	-
R_000_A06	*	368,5	368,5	236,9	236,9	*	370,0	372,0	258,3	264,9	*	375	375	437	389	-	-	-	-	-
R_000_A07	*	*	*	279,0	279,0	*	*	*	290,4	295,6	*	328	328	399	332	-	-	-	-	-
R_000_A08	*	*	356,4	279,8	279,8	*	*	356,4	286,0	297,1	*	374	374	413	375	-	-	-	-	-
R_000_A09	*	373,8	373,8	280,2	280,2	*	375,3	375,3	288,1	295,3	*	411	398	511	419	-	-	-	-	-
R_025_A01	*	*	*	182,0	182,0	*	*	*	200,3	199,0	*	229	229	228	229	-	-	-	-	-
R_025_A02	*	237,5	237,5	179,0	179,0	*	237,5	237,5	207,9	206,5	241	268	243	240	240	-	-	-	-	-
R_025_A03	268,0	268,0	268,0	190,0	190,0	268,0	268,0	268,0	216,5	226,3	268	268	268	277	268	799,9	472,2	301,7	-	-
R_025_A04	*	*	*	221,0	221,0	*	*	*	242,3	241,4	*	259	259	259	259	-	-	-	-	-
R_025_A05	*	311,0	*	225,2	225,2	*	311,0	*	250,4	248,1	*	329	329	433	329	-	-	-	-	-
R_025_A06	*	337,0	337,0	226,7	226,7	*	337,0	337,0	252,1	247,3	*	337	337	462	362	-	681,2	258,9	-	-
R_025_A07	*	*	*	259,0	259,0	*	*	*	270,1	275,0	*	315	311	389	314	-	-	-	-	-
R_025_A08	*	*	*	255,3	255,3	*	*	*	273,5	271,4	*	341	349	437	344	-	-	-	-	-
R_025_A09	*	*	*	261,0	261,0	*	*	*	277,6	278,9	505	357	357	440	358	-	-	-	-	-
R_050_A01	*	*	*	167,8	167,8	*	*	*	191,8	186,9	288	233	205	205	205	-	-	-	-	-
R_050_A02	*	*	210,0	161,9	161,9	*	*	210,0	187,2	196,6	*	210	210	225	210	-	-	1033,6	-	-
R_050_A03	210,0	210,0	*	156,0	156,0	210,0	210,0	*	189,9	210,0	210	210	210	210	210	1192,9	161,7	-	-	1167,2
R_050_A04	*	*	*	197,0	197,0	*	*	*	211,7	213,1	*	239	236	298	240	-	-	-	-	-
R_050_A05	*	*	*	215,2	215,2	*	*	*	235,2	236,5	*	318	318	355	318	-	-	-	-	-
R_050_A06	*	*	309,3	205,1	205,1	*	*	309,3	213,2	225,1	313	327	314	380	360	-	-	-	-	-
R_050_A07	*	*	*	237,7	237,7	*	*	*	247,4	251,0	*	285	269	322	285	-	-	-	-	-
R_050_A08	*	*	*	225,5	225,5	*	*	*	236,5	241,0	*	307	308	388	308	-	-	-	-	-
R_050_A09	*	308,5	308,5	243,4	243,4	*	308,5	309,0	250,2	263,3	*	320	320	*	323	-	-	-	-	-
R_075_A01	*	*	*	157,0	157,0	*	*	*	171,9	171,8	*	201	191	190	190	-	-	-	-	-
R_075_A02	*	*	*	141,6	141,6	*	*	*	157,2	157,2	*	176	188	176	176	-	-	-	-	-
R_075_A03	*	196,0	196,0	152,7	152,7	*	196,0	196,0	174,3	174,5	407	196	196	196	197	-	838,1	690,9	-	-
R_075_A04	*	*	*	173,0	173,0	*	*	*	184,1	184,1	*	249	197	192	192	-	-	-	-	-
R_075_A05	*	*	*	184,5	184,5	*	*	*	198,5	197,4	*	318	317	293	319	-	-	-	-	-
R_075_A06	*	*	264,8	164,0	164,0	*	*	264,8	175,6	178,5	*	271	278	424	293	-	-	-	-	-
R_075_A07	*	*	*	213,5	213,5	*	*	*	220,4	220,8	*	263	263	249	242	-	-	-	-	-
R_075_A08	*	*	*	201,8	201,8	*	*	*	209,2	211,6	*	244	259	273	281	-	-	-	-	-
R_075_A09	*	*	*	197,4	197,4	*	*	*	205,5	205,4	*	288	287	321	290	-	-	-	-	-
R_100_A01	*	*	*	77,4	77,4	*	*	*	87,7	88,5	193	138	137	158	140	-	-	-	-	-
R_100_A02	*	135,0	135,0	77,4	77,4	*	135,0	135,0	87,5	87,5	175	135	135	140	140	-	369,5	383,0	-	-
R_100_A03	139,0	139,0	139,0	77,4	77,4	139,0	139,0	139,0	87,0	86,1	139	139	139	145	152	1449,0	125,1	149,3	-	-
R_100_A04	*	*	*	98,9	98,9	*	*	*	103,9	103,9	*	165	163	180	169	-	-	-	-	-
R_100_A05	*	*	*	107,2	107,2	*	*	*	112,1	111,9	*	260	266	278	276	-	-	-	-	-
R_100_A06	*	247,5	247,5	107,2	107,2	*	247,5	247,5	111,4	112,1	261	252	257	292	273	-	-	-	-	-
R_100_A07	*	*	*	116,8	116,8	*	*	*	120,8	120,2	*	188	185	194	191	-	-	-	-	-
R_100_A08	*	*	*	119,4	119,4	*	*	*	122,2	122,3	*	224	248	236	238	-	-	-	-	-
R_100_A09	*	197,5	*	119,4	119,4	*	197,5	*	123,2	123,6	*	229	240	236	245	-	-	-	-	-

Tabela 5.8: Resultados do grupo de instâncias CF0_QT66.

Instância	rootLB					LB					UB					Tempo				
	SPF-	SPF	SPF*	MCF-	MCF	SPF-	SPF	SPF*	MCF-	MCF	SPF-	SPF	SPF*	MCF-	MCF	SPF-	SPF	SPF*	MCF-	MCF
R_000_A01	*	236,8	236,8	190,0	190,0	*	237,4	237,4	214,5	215,8	*	242	242	242	242	-	-	-	-	-
R_000_A02	251,0	251,0	251,0	190,0	190,0	251,0	251,0	251,0	218,8	218,6	251	251	251	251	251	1323,7	153,1	248,3	-	-
R_000_A03	279,0	279,0	279,0	190,4	190,4	279,0	279,0	279,0	226,8	226,8	279	279	279	279	279	387,8	66,4	51,3	-	-
R_000_A04	*	*	*	236,5	236,5	*	*	*	252,0	257,5	*	301	301	379	301	-	-	-	-	-
R_000_A05	*	336,8	336,8	236,7	236,7	*	337,8	337,6	252,0	258,2	*	339	349	466	347	-	-	-	-	-
R_000_A06	*	368,5	368,5	236,9	236,9	*	372,0	369,5	258,3	265,6	*	375	377	462	387	-	-	-	-	-
R_000_A07	*	*	*	279,0	279,0	*	*	*	293,0	295,0	*	336	336	468	328	-	-	-	-	-
R_000_A08	*	356,4	356,4	279,8	279,8	*	356,4	356,4	284,4	294,2	*	374	369	*	371	-	-	-	-	-
R_000_A09	*	373,8	373,8	280,2	280,2	*	374,8	373,8	295,0	302,0	*	396	413	540	413	-	-	-	-	-
R_025_A01	*	*	228,9	184,2	184,2	*	*	228,9	202,3	199,5	*	242	242	242	242	-	-	-	-	-
R_025_A02	246,7	246,7	246,7	180,5	180,5	247,0	247,0	249,0	208,2	210,6	251	251	249	257	256	-	-	1647,7	-	-
R_025_A03	274,0	274,0	274,0	190,0	190,0	274,5	274,5	274,5	211,9	211,9	283	277	292	277	277	-	-	-	-	-
R_025_A04	*	*	*	223,4	223,4	*	*	*	239,4	241,9	*	301	301	465	301	-	-	-	-	-
R_025_A05	*	*	334,5	228,5	228,5	*	*	334,5	251,8	250,6	*	346	346	415	346	-	-	-	-	-
R_025_A06	*	364,2	364,2	228,7	228,7	*	365,3	365,3	248,6	251,2	*	375	375	433	375	-	-	-	-	-
R_025_A07	*	*	*	263,1	263,1	*	*	*	266,2	275,2	*	329	329	*	329	-	-	-	-	-
R_025_A08	*	*	*	260,0	260,0	*	*	*	275,3	271,7	*	369	376	482	380	-	-	-	-	-
R_025_A09	*	369,0	369,0	265,8	265,8	*	369,0	369,0	277,6	279,8	369	369	369	470	389	-	1716,5	1117,9	-	-
R_050_A01	*	*	221,0	170,9	170,9	*	*	221,0	188,0	189,9	*	258	258	285	258	-	-	-	-	-
R_050_A02	*	*	237,7	163,7	163,7	*	*	237,7	191,1	188,9	*	249	261	273	259	-	-	-	-	-
R_050_A03	258,0	258,0	258,0	157,4	157,4	258,0	258,0	258,5	184,3	189,0	*	278	275	373	278	-	-	-	-	-
R_050_A04	*	*	*	199,5	199,5	*	*	*	211,9	217,4	*	301	301	440	301	-	-	-	-	-
R_050_A05	*	*	328,8	217,9	217,9	*	*	328,8	230,7	236,2	*	345	345	*	353	-	-	-	-	-
R_050_A06	*	*	361,7	208,0	208,0	*	*	363,3	221,4	220,7	*	385	385	497	398	-	-	-	-	-
R_050_A07	*	*	*	242,3	242,3	*	*	*	252,0	254,3	*	328	328	*	328	-	-	-	-	-
R_050_A08	*	*	*	232,1	232,1	*	*	*	244,4	247,0	*	369	369	*	369	-	-	-	-	-
R_050_A09	*	369,0	369,0	248,6	248,6	*	369,0	369,0	253,3	261,6	*	369	369	*	404	-	1678,9	1064,4	-	-
R_075_A01	*	*	*	158,3	158,3	*	*	*	170,0	169,8	*	242	244	407	244	-	-	-	-	-
R_075_A02	*	*	*	144,1	144,1	*	*	*	158,3	160,3	*	237	237	350	249	-	-	-	-	-
R_075_A03	258,0	258,0	258,0	155,1	155,1	258,5	258,0	258,5	172,4	174,9	266	285	266	582	288	-	-	-	-	-
R_075_A04	*	*	*	177,4	177,4	*	*	*	184,1	182,7	*	303	303	*	303	-	-	-	-	-
R_075_A05	*	*	*	186,7	186,7	*	*	*	196,4	199,6	*	355	352	*	355	-	-	-	-	-
R_075_A06	*	*	354,5	169,5	169,5	*	*	354,5	177,9	181,8	*	387	390	*	377	-	-	-	-	-
R_075_A07	*	*	*	219,2	219,2	*	*	*	221,0	224,5	*	335	335	*	344	-	-	-	-	-
R_075_A08	*	*	*	216,6	216,6	*	*	*	218,9	222,7	*	365	365	*	382	-	-	-	-	-
R_075_A09	*	*	*	209,4	209,4	*	*	*	213,4	213,0	369	369	390	*	397	-	-	-	-	-
R_100_A01	*	*	*	86,3	86,3	*	*	*	101,1	102,5	*	242	242	451	242	-	-	-	-	-
R_100_A02	*	231,3	231,3	85,6	85,6	*	231,3	231,3	94,3	96,7	*	237	237	463	237	-	-	-	-	-
R_100_A03	*	255,0	255,0	85,6	85,6	*	255,0	255,0	95,5	96,5	*	268	266	493	266	-	-	-	-	-
R_100_A04	*	*	*	123,3	123,3	*	*	*	131,8	130,6	*	301	301	*	301	-	-	-	-	-
R_100_A05	*	*	*	121,8	121,8	*	*	*	130,4	130,5	*	350	346	*	345	-	-	-	-	-
R_100_A06	*	354,5	354,5	121,7	121,7	*	354,5	354,5	129,7	130,2	464	385	384	*	387	-	-	-	-	-
R_100_A07	*	*	*	159,4	159,4	*	*	*	167,3	169,2	*	328	328	*	328	-	-	-	-	-
R_100_A08	*	*	*	156,4	156,4	*	*	*	165,1	165,8	*	362	377	*	381	-	-	-	-	-
R_100_A09	*	*	*	155,0	155,0	*	*	*	163,1	163,2	*	368	369	*	393	-	-	-	-	-

Tabela 5.9: Resultados do grupo de instâncias CFMED_QT1.

	rootLB					LB					UB					Tempo				
Instância	SPF-	SPF	SPF*	MCF-	MCF	SPF-	SPF	SPF*	MCF-	MCF	SPF-	SPF	SPF*	MCF-	MCF	SPF-	SPF	SPF*	MCF-	MCF
R_000_A01	*	236,8	236,8	190,0	190,0	*	237,4	237,4	217,3	219,7	*	242	242	242	242	-	-	-	-	-
R_000_A02	251,0	251,0	251,0	190,0	190,0	251,0	251,0	251,0	218,4	222,9	251	251	251	251	251	1318,7	382,5	124,5	-	-
R_000_A03	279,0	279,0	279,0	190,4	190,4	279,0	279,0	279,0	228,0	221,3	279	279	279	279	279	388,9	67,7	179,9	-	-
R_000_A04	*	*	292,4	236,5	236,5	*	*	292,4	257,0	253,1	*	301	301	347	301	-	-	-	-	-
R_000_A05	*	336,8	336,8	236,7	236,7	*	339,0	338,0	261,1	256,8	*	339	339	423	340	-	1482,1	-	-	-
R_000_A06	*	368,5	368,5	236,9	236,9	*	370,0	370,0	263,0	268,2	*	375	383	466	388	-	-	-	-	-
R_000_A07	*	*	*	279,0	279,0	*	*	*	290,7	296,7	*	333	328	349	328	-	-	-	-	-
R_000_A08	*	356,4	356,4	279,8	279,8	*	356,4	356,4	289,7	300,2	*	368	370	493	373	-	-	-	-	-
R_000_A09	*	*	373,8	280,2	280,2	*	*	374,8	282,7	297,7	*	399	398	*	417	-	-	-	-	-
R_025_A01	*	*	*	184,2	184,2	*	*	*	202,0	205,3	230	227	225	225	225	-	-	-	-	-
R_025_A02	239,0	239,0	239,0	180,5	180,5	243,0	239,0	243,0	206,5	207,1	243	243	243	243	243	1789,5	-	911,9	-	-
R_025_A03	268,0	268,0	268,0	190,0	190,0	268,0	268,0	268,0	217,8	227,1	268	268	268	277	268	1043,8	647,5	140,4	-	-
R_025_A04	*	*	*	223,4	223,4	*	*	*	239,5	247,7	*	259	259	274	259	-	-	-	-	-
R_025_A05	*	327,9	327,9	228,5	228,5	*	327,9	327,9	245,5	252,1	*	346	346	478	346	-	-	-	-	-
R_025_A06	350,0	*	350,0	228,7	228,7	350,0	*	352,0	247,0	250,4	355	352	352	536	375	-	-	749,2	-	-
R_025_A07	*	*	*	262,9	262,9	*	*	*	272,2	276,7	*	329	304	488	329	-	-	-	-	-
R_025_A08	*	*	*	260,0	260,0	*	*	*	267,0	279,8	*	349	349	516	376	-	-	-	-	-
R_025_A09	*	*	357,0	265,8	265,8	*	*	357,0	266,2	284,2	*	357	357	*	357	-	-	1156,2	-	-
R_050_A01	*	*	205,0	170,9	170,9	*	*	205,0	192,2	205,0	*	211	205	205	205	-	-	1788,2	-	1150,8
R_050_A02	*	215,0	215,0	163,2	163,2	*	215,0	215,0	190,2	191,5	*	219	218	224	223	-	-	-	-	-
R_050_A03	210,0	210,0	210,0	157,2	157,2	210,0	210,0	210,0	210,0	210,0	210	210	210	210	210	800,2	173,0	327,5	1721,1	1706,0
R_050_A04	*	*	*	199,2	199,2	*	*	*	212,0	215,1	*	242	239	277	301	-	-	-	-	-
R_050_A05	*	*	*	217,6	217,6	*	*	*	225,6	235,7	*	345	345	485	352	-	-	-	-	-
R_050_A06	*	*	332,0	207,3	207,3	*	*	332,0	227,8	228,3	*	332	332	384	398	-	-	1361,5	-	-
R_050_A07	*	*	*	241,9	241,9	*	*	*	251,7	253,0	*	280	298	347	328	-	-	-	-	-
R_050_A08	*	*	*	230,2	230,2	*	*	*	235,4	243,3	*	369	369	*	369	-	-	-	-	-
R_050_A09	*	310,0	310,0	246,7	246,7	*	310,0	310,5	253,7	262,8	*	322	322	547	324	-	-	-	-	-
R_075_A01	*	*	*	157,1	157,1	*	*	*	169,8	169,8	*	205	205	224	224	-	-	-	-	-
R_075_A02	*	*	*	142,2	142,2	*	*	*	158,4	158,4	*	204	192	216	226	-	-	-	-	-
R_075_A03	200,0	200,0	200,0	153,7	153,7	200,0	200,0	200,0	174,1	174,6	200	200	200	215	206	1384,3	500,8	532,8	-	-
R_075_A04	*	*	*	175,2	175,2	*	*	*	182,0	178,7	*	243	239	323	278	-	-	-	-	-
R_075_A05	*	*	*	185,2	185,2	*	*	*	193,3	192,0	*	357	354	574	352	-	-	-	-	-
R_075_A06	*	*	*	165,0	165,0	*	*	*	176,1	177,2	*	344	339	475	350	-	-	-	-	-
R_075_A07	*	*	*	216,5	216,5	*	*	*	220,4	222,9	*	335	335	426	345	-	-	-	-	-
R_075_A08	*	*	*	206,3	206,3	*	*	*	212,3	212,3	*	323	365	*	382	-	-	-	-	-
R_075_A09	*	*	*	200,9	200,9	*	*	*	206,7	213,2	*	306	312	445	312	-	-	-	-	-
R_100_A01	*	*	*	77,4	77,4	*	*	*	86,8	87,6	*	198	198	282	208	-	-	-	-	-
R_100_A02	*	*	*	77,4	77,4	*	*	*	85,9	87,5	*	189	189	226	189	-	-	-	-	-
R_100_A03	165,0	165,0	165,0	77,4	77,4	165,0	165,0	165,0	86,0	86,2	165	165	165	235	165	1681,1	661,7	645,5	-	-
R_100_A04	*	*	*	98,9	98,9	*	*	*	103,0	102,7	*	241	239	368	241	-	-	-	-	-
R_100_A05	*	*	*	107,2	107,2	*	*	*	112,4	112,2	*	339	346	*	345	-	-	-	-	-
R_100_A06	*	*	*	107,2	107,2	*	*	*	111,2	111,7	*	332	332	*	353	-	-	-	-	-
R_100_A07	*	*	*	116,8	116,8	*	*	*	120,9	122,4	*	266	279	322	268	-	-	-	-	-
R_100_A08	*	*	*	119,4	119,4	*	*	*	123,6	123,4	*	307	309	*	332	-	-	-	-	-
R_100_A09	*	*	*	119,4	119,4	*	*	*	123,2	123,8	*	297	299	335	296	-	-	-	-	-

Tabela 5.10: Resultados do grupo de instâncias CFMED_QT66.

Com o objetivo de facilitar a análise dos resultados da Tabela 5.7, Tabela 5.8, Tabela 5.9 e Tabela 5.10, foi realizado o resumo dos dados, que podem ser visualizados, respectivamente, na Tabela 5.11, Tabela 5.12, Tabela 5.13 e Tabela 5.14.

Situação	SPF-	SPF	SPF*	MCF-	MCF
Ótimo (%)	4,40	8,90	6,70	0,00	0,00
GAP médio (%)	0,03	1,78	2,56	36,52	47,85
Com *GAP* (%)	6,70	35,60	48,90	71,10	100,00
Tempo (s)	859,75	550,63	476,10	*	*
GAP RootLB	3,57	3,57	3,57	34,00	34,00

Tabela 5.11: Resumo dos resultados do grupo de instâncias CF0_QT1.

Situação	SPF-	SPF	SPF*	MCF-	MCF
Ótimo (%)	11,10	17,80	17,80	0,00	2,20
GAP médio (%)	0,00	1,45	0,86	48,98	37,78
Com *GAP* (%)	11,10	37,80	40,00	97,80	100,00
Tempo (s)	1029,60	366,49	389,94	*	1167,20
GAP RootLB	2,13	2,13	2,13	28,75	28,75

Tabela 5.12: Resumo dos resultados do grupo de instâncias CF0_QT66.

Situação	SPF-	SPF	SPF*	MCF-	MCF
Ótimo (%)	4,40	8,90	11,10	0,00	0,00
GAP médio (%)	0,18	1,30	2,78	65,32	63,67
Com *GAP* (%)	11,10	37,80	53,30	60,00	100,00
Tempo (s)	855,75	903,73	825,92	*	*
GAP RootLB	3,59	3,59	3,59	38,85	38,85

Tabela 5.13: Resumo dos resultados do grupo de instâncias CFMED_QT1.

Nessas tabelas, as colunas SPF-, SPF, SPF*, MCF- e MCF dizem respeito aos cinco programas executados para resolver o grupo de instâncias e cada linha das tabelas sintetizam os resultados conforme descrito abaixo:

- *Ótimo (%)*: Representa a porcentagem de instâncias que foram resolvidas em sua otimalidade;
- *GAP (%)*: Referente à média do *GAP* das instâncias que tiveram alguma solução encontrada, não necessariamente ótima. O *GAP* de uma solução é dado pela fórmula $((UB - LB)/LB) \times 100$;
- *Com GAP (%)*: Corresponde à porcentagem de instâncias que apresentaram algum *GAP*;

Situação	SPF-	SPF	SPF*	MCF-	MCF
Ótimo (%)	15,60	15,60	24,40	2,20	4,40
GAP médio (%)	0,04	0,51	0,86	69,59	59,54
Com *GAP* (%)	17,80	31,10	44,40	84,40	100,00
Tempo (s)	1200,93	559,33	719,78	1721,10	1428,40
GAP RootLB	1,46	1,46	1,46	34,13	34,13

Tabela 5.14: Resumo dos resultados do grupo de instâncias CFMED_QT66.

- *Tempo (s)*: Diz respeito ao tempo médio consumido, em segundos, para resolver as instâncias de forma ótima;
- *GAP RootLB*: Referente à média do *GAP* encontrado no nó raiz das instâncias. A fórmula do *GAP* é a mesma apresentada na Equação 5.1.

Observando as Tabelas 5.11, 5.12, 5.13 e 5.14 podemos verificar que o programa MCF foi capaz de obter soluções com *GAP* para 100% das instâncias em todos os quatro grupos, porém apresentou uma média dos *GAP*s superior a 37,78% e resolveu de forma ótima menos de 5% das instâncias. Por outro lado, o programa SPF* obteve, na maioria, os melhores resultados, resolvendo na otimalidade de 6% a quase 25% das instâncias, além disso, apresentou uma média dos *GAP*s inferior a 3%.

Com o objetivo de avaliar o desempenho da heurística primal na formulação (SPF), comparou-se os resultados obtidos pelos programas SPF- e SPF. A comparação pode ser verificada na Tabela 5.15.

Instâncias	Com *GAP* SPF-/SPF(=)	*GAP* SPF-	*GAP* SPF	Ótimos SPF-/SPF(=)	Tempo SPF-	Tempo SPF
CF0_QT1	3 / 16 (3)	0,37	2,20	2 / 4 (2)	859,8	192,2
CF0_QT66	5 / 17 (5)	0,00	0,00	5 / 8 (5)	1029,6	208,6
CFMED_QT1	5 / 17 (5)	1,52	2,60	2 / 4 (2)	855,8	109,8
CFMED_QT66	8 / 14 (7)	0,00	0,24	7 / 7 (6)	1102,8	405,5
Soma	21 / 64 (20)			16 / 23 (15)		
Média		0,47	1,26		962,0	229,0

Tabela 5.15: Desempenho SPF- $\times$ SPF.

Na Tabela 5.15, cada entrada da coluna *Com GAP SPF-/SPF(=)* é apresentada na forma $|X|/|Y|(|Z|)$, sendo X o conjunto de instâncias com *GAP* no programa *SPF-*, Y o conjunto de instâncias com *GAP* no programa *SPF* e Z o conjunto de instâncias com *GAP* nos dois programas. As colunas *GAP SPF-* e *GAP SPF* apresentam a média do *GAP*, em porcentagem, dos resultados de $(X \cap Z)$ e $(Y \cap Z)$, respectivamente. A coluna *Ótimos SPF-/SPF(=)* é apresentada na forma $|A|/|B|(|C|)$, sendo A o conjunto de instâncias resolvidas de forma ótima pelo programa *SPF-*, B o conjunto de instâncias resolvidas de forma

ótima pelo programa *SPF* e C o conjunto de instâncias resolvidas de forma ótima pelos dois programas. As colunas *Tempo SPF-* e *Tempo SPF* apresentam a média do tempo, em segundos, utilizado pelos programas para resolver as instâncias em $(A \cap C)$ e $(B \cap C)$, respectivamente.

Observando a Tabela 5.15 é possível constatar que a implementação da heurística primal promoveu um aumento de $204,76\%$ no número de resultados com *GAP* encontrados. É perceptível também um aumento de $43,75\%$ no número de soluções ótimas encontradas e a redução em $76,19\%$ do tempo de solução de 15 instâncias, resolvidas de forma ótima pelos programas SPF- e SPF. É importante ressaltar também um pequeno aumento de $0,79\%$ no *GAP* de 20 instâncias passíveis de comparação.

Almejando avaliar o desempenho da heurística primal na formulação (MCF), comparou-se os resultados obtidos pelos programas MCF- e MCF. A comparação pode ser verificada na Tabela 5.16.

Instâncias	Com *GAP* MCF-/MCF(=)	*GAP* MCF-	*GAP* MCF	Ótimos MCF-/MCF(=)	Tempo MCF-	Tempo SPF
CF0_QT1	32 / 45 (32)	51,35	38,68	0 / 0 (0)	*	*
CF0_QT66	44 / 45 (44)	50,09	37,26	0 / 1 (0)	*	*
CFMED_QT1	27 / 45 (27)	108,86	48,82	0 / 0 (0)	*	*
CFMED_QT66	38 / 45 (38)	80,57	46,59	1 / 2 (1)	1721,1	1706,0
Soma	141 / 180 (141)			1 / 3 (1)		
Média		72,72	42,84		1721,1	1706,0

Tabela 5.16: Desempenho MCF- $\times$ MCF.

A Tabela 5.16 resume os resultados obtidos pelos programas MCF- e MCF. A disposição das colunas segue o mesmo padrão da Tabela 5.15, porém aplicada aos programas MCF- e MCF.

Verificando a Tabela 5.16 é possível perceber que a utilização da heurística primal no programa MCF aumentou em $27,66\%$ a quantidade de resultados com *GAP*, atingindo 100% das instâncias. É possível ainda constatar uma redução de $29,98\%$ no *GAP* obtido com a aplicação da heurística.

Aspirando avaliar o desempenho da heurística de *pricing*, aplicada à formulação (SPF), comparou-se os resultados obtidos pelos programas SPF e SPF*. A comparação pode ser visualizada na Tabela 5.17.

A Tabela 5.17 sumariza os resultados obtidos pelos programas SPF e SPF*. A disposição das colunas segue o mesmo padrão da Tabela 5.15, porém aplicada aos programas SPF e SPF*.

Verificando a Tabela 5.17, observamos que a aplicação da heurística de *pricing* foi capaz de aumentar em $31,25\%$ a quantidade de resultados com *GAP* encontrados, além de reduzir em $0,26\%$ o *GAP* de 61 instâncias comparadas. É possível constatar também um aumento de $17,39\%$ na quantidade de soluções

Instâncias	Com *GAP* SPF/SPF*(=)	*GAP* SPF	*GAP* SPF*	Ótimos SPF/SPF*(=)	Tempo SPF	Tempo SPF*
CF0_QT1	16 / 22 (16)	4,55	4,22	4 / 3 (3)	488,3	476,1
CF0_QT66	17 / 18 (14)	2,29	1,46	8 / 8 (7)	395,7	298,0
CFMED_QT1	17 / 24 (17)	3,12	3,21	4 / 5 (4)	903,7	620,5
CFMED_QT66	14 / 20 (14)	1,39	1,44	7 / 11 (6)	405,5	325,1
Soma	64 / 84 (61)			23 / 27 (20)		
Média		2,84	2,58		548,3	429,9

Tabela 5.17: Desempenho SPF $\times$ SPF*.

ótimas encontradas, além da redução de $21,59\%$ no tempo de solução ótimo de 20 instâncias.

Com o objetivo de avaliar o desempenho dos programas baseados nos dois modelos matemáticos, comparou-se os resultados obtidos pelos programas MCF e SPF. A comparação pode ser visualizada na Tabela 5.18.

Instâncias	Com *GAP* MCF/SPF(=)	*GAP* MCF	*GAP* SPF	Ótimos MCF/SPF(=)	Tempo MCF	Tempo SPF
CF0_QT1	45 / 16 (16)	50,16	4,55	0 / 4 (0)	*	*
CF0_QT66	45 / 17 (17)	40,86	3,16	1 / 8 (1)	1167,2	161,7
CFMED_QT1	45 / 17 (17)	59,84	3,12	0 / 4 (0)	*	*
CFMED_QT66	45 / 14 (14)	26,59	1,39	2 / 7 (1)	1706,0	173,0
Soma	180 / 64 (64)			3 / 23 (2)		
Média		44,36	3,06		1436,6	167,4

Tabela 5.18: Desempenho MCF $\times$ SPF.

Na Tabela 5.18, a disposição das colunas segue o mesmo padrão da Tabela 5.15, porém aplicada aos programas MCF e SPF.

Observando a Tabela 5.18 é possível constatar que o programa MCF, mesmo sendo capaz de obter resultados com *GAP* para 100% das instâncias, obteve somente 3 resultados ótimos. Em contrapartida, o programa SPF obteve resultados com *GAP* para $35,56\%$ das instâncias, sendo, aproximadamente, $1/3$ destes resultados ótimos. Verifica-se também que o *GAP* dos resultados obtidos pelo programa SPF é de somente $3,06\%$, um valor $41,3\%$ menor do que o encontrado pelo programa MCF. Por fim, é importante ressaltar a redução de $88,35\%$ no tempo de solução das instâncias.

Visando avaliar a qualidade da solução inicial, foi realizada a comparação do melhor UB, fornecido pela heurística primal nos programas SPF, SPF* e MCF, com o melhor UB encontrado em todos os cinco programas executados em cada instância. Os resultados foram resumidos e podem ser obervados na Tabela 5.19.

A coluna *Melhor UB* da Tabela 5.19 é referente à porcentagem de soluções

Instâncias	Melhor UB	Ótimo	*GAP* UB
CF0_QT1	48,89	4,44	4,54
CF0_QT66	35,56	8,89	8,33
CFMED_QT1	62,22	6,67	4,74
CFMED_QT66	42,22	15,56	9,96
Média	47,22	8,89	6,89

Tabela 5.19: Solução inicial.

iniciais que são iguais ao melhor UB de cada instância, a coluna *Ótimo* diz respeito à porcentagem de soluções iniciais que são iguais ao valor ótimo da solução, por fim a coluna *GAP UB* representa uma média do *GAP*, de cada instância, o qual é dado pela fórmula $(FirstSol - BestUB)/BestUB$ sendo $FirstSol$ o valor da solução inicial fornecida pela heurística primal e $BestUB$ o melhor limitante superior (UB) da instância encontrado após a execução dos programas SPF-, SPF, SPF*, MCF- e MCF.

Observando a Tabela 5.19 é possível verificar que $47,22\%$ das soluções iniciais, provenientes da heurística primal, são iguais ao melhor limitante superior e que $8,89\%$ das soluções iniciais são iguais ao ótimo. Para as soluções que não são iguais ao melhor limitante superior, o *GAP* é de apenas $6,89\%$. Podemos concluir então, que heurística primal fornece bons limitantes superiores (UB), o que reforça a melhoria no desempenho dos programas MCF e SPF, conforme mostrado na Tabela 5.15 e Tabela 5.16.

Tendo em vista que tanto a heurística primal quanto a heurística de *pricing* fornecem pseudo-arborescências que podem compor a solução final (a que fornece o melhor UB), um novo experimento foi realizado para avaliar a origem das pseudo-arborescências que compõem as soluções finais, obtidas pelo programa SPF* nos quatro grupos de instâncias do CPAFLP. O resultado pode ser visualizado na Tabela 5.20.

Instância	m	Primal	Exato	*Pricing*
R_000_A01	3	100%	-	-
R_000_A02	4	100%	-	-
R_000_A03	5	100%	-	-
R_000_A04	3	100%	-	-
R_000_A05	4	100%	-	-
R_000_A06	5	100%	-	-
R_000_A07	3	100%	-	-
R_000_A08	4	100%	-	-
R_000_A09	5	100%	-	-
R_025_A01	3	83%	8%	8%
R_025_A02	4	33%	33%	33%
R_025_A03	5	74%	-	26%
R_025_A04	3	83%	17%	-

continua na próxima página.

Tabela 5.20 – continuação.

Instância	m	Primal	Exato	*Pricing*
R_025_A05	4	88%	6%	6%
R_025_A06	5	85%	5%	10%
R_025_A07	3	92%	8%	-
R_025_A08	4	75%	13%	13%
R_025_A09	5	100%	-	-
R_050_A01	3	67%	17%	17%
R_050_A02	4	69%	13%	19%
R_050_A03	5	90%	-	10%
R_050_A04	3	75%	25%	-
R_050_A05	4	81%	6%	13%
R_050_A06	5	60%	5%	35%
R_050_A07	3	83%	-	17%
R_050_A08	4	100%	-	-
R_050_A09	5	95%	-	5%
R_075_A01	3	75%	17%	8%
R_075_A02	4	75%	-	25%
R_075_A03	5	55%	5%	40%
R_075_A04	3	67%	17%	17%
R_075_A05	4	100%	-	-
R_075_A06	5	75%	5%	20%
R_075_A07	3	100%	-	-
R_075_A08	4	94%	-	6%
R_075_A09	5	90%	10%	-
R_100_A01	3	92%	8%	-
R_100_A02	4	93%	7%	-
R_100_A03	5	89%	11%	-
R_100_A04	3	83%	17%	-
R_100_A05	4	100%	-	-
R_100_A06	5	95%	5%	-
R_100_A07	3	100%	-	-
R_100_A08	4	94%	6%	-
R_100_A09	5	100%	-	-
Média		87%	5%	8%

Tabela 5.20: Origem das pseudo-arborescências.

A coluna *m* da Tabela 5.20 representa o limite de pseudo-arborescências da instância, as colunas *Primal*, *Exato* e *Pricing* dizem respeito à porcentagem de pseudo-arborescências da solução final com origem, respectivamente, na heurística primal, no *pricing* exato e na heurística de *pricing*.

Analisando a Tabela 5.20, é possível observar que a heurística primal contribui com 87% das pseudo-arborescências que compõem as soluções finais, seguido por 8% da heurística de *pricing* e por 5% obtidos pelo método exato do SPF*. Assim, podemos concluir que a base inicial oferecida pela heurística primal é composta por pseudo-arborescências com custo vantajoso, pois estas participam massivamente das soluções finais.

Cada programa gera, ao final da computação, um arquivo de saída em linguagem DOT, explicado em Koutsofios et al. (1991), que através do pacote Graphviz, descrito em Ellson et al. (2001), pode ser convertido em imagens. A Figura 5.1 é um exemplo dessa saída convertida.

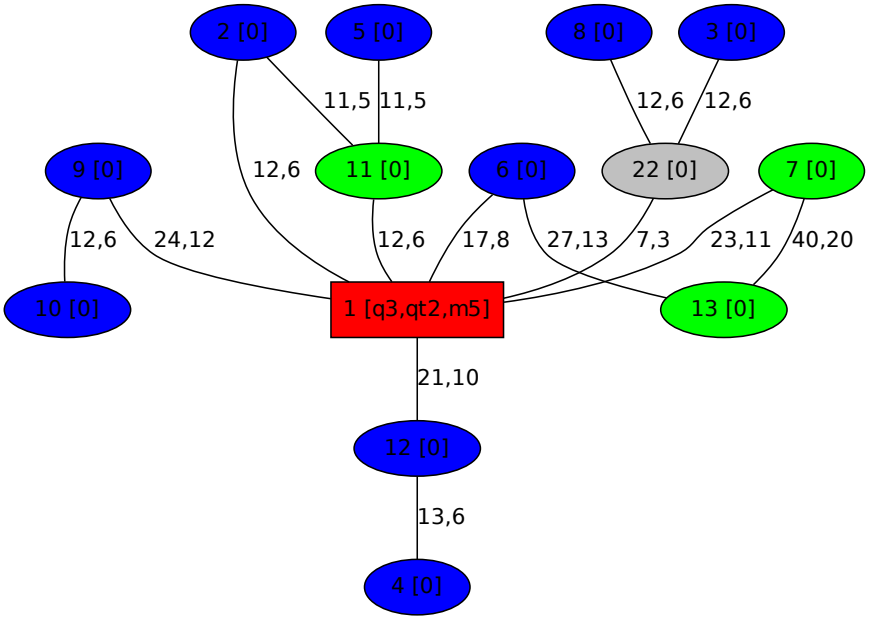

Figura 5.1: Solução da instância R_075_A03 do grupo CF0_QT66.

A Figura 5.1 é referente ao grafo solução da instância R_075_A03 do conjunto de testes CF0_QT66. A solução apresentada é ótima e possui 5 pseudo-arborescências de custo total 196. A instância foi resolvida pelos programas SPF e SPF*, no tempo de 838 e 691 segundos, respectivamente. Cada aresta da Figura 5.1 possui dois valores que representam, respectivamente, o custo de utilização do arco em um ciclo e em uma arborescência. Cada vértice possui um número que o identifica dentro da instância e um valor entre colchetes que corresponde ao custo de instalação de facilidade no mesmo.

5.4 Considerações sobre os Resultados

A criação dos 4 grupos de instâncias de testes teve como objetivo principal manter as características das instâncias originalmente propostas para o *CRTP*. Considerou-se também a utilização de instâncias com 26 vértices e denotadas como fáceis por Hill (2012). Tais escolhas resultaram em 180 instâncias de características distintas, um número que consideramos suficiente para avaliar os modelos propostos e algoritmos.

Os experimentos realizados para a definição dos parâmetros α e *max_iter* foram realizados com o objetivo de minimizar a quantidade de testes necessários. A escolha de um número reduzido de instâncias, 45 do total, foi positiva pois possibilitou a realização de testes detalhados envolvendo 11 valores de α. Nesta etapa optou-se por não realizar testes com variações de *max_iter* uma vez que os testes, com *max_iter* igual a 1000, não extrapolaram a casa dos décimos de segundo.

As implementações dos cinco programas foram realizadas para permitir diferentes comparações, com respeito aos modelos e heurísticas propostas.

Considerando os experimentos realizados podemos concluir que a formulação (SPF) fornece limitantes inferiores muito melhores que (MCF), tendo em vista que a média do *GAP* nó raiz da formulação (SPF) foi de apenas $2,69\%$, um valor $30,62\%$ menor que o encontrado na formulação (MCF). O *GAP* da formulação (SPF) se mostrou tão apertado que $28,74\%$ das 87 instâncias comparadas foram resolvidas na otimalidade ainda no nó raiz, conforme apresentado na Tabela 5.6.

Analisando os experimentos realizados com os programas SPF- $\times$ SPF e MCF- $\times$ MCF, mostrados nas Tabelas 5.15 e 5.16, é possível concluir que a aplicação da heurística primal contribuiu na melhoria do desempenho de ambas as formulações. No programa SPF, a aplicação da heurística primal triplicou a quantidade de instâncias em que foram possíveis de ser obter um *GAP* e aumentou em $43,75\%$ a quantidade de soluções ótimas encontradas. Já na formulação (MCF), a aplicação da heurística permitiu ao programa MCF encontrar soluções com *GAP* para 100% das instâncias além de reduzir o *GAP* em, aproximadamente, 30%.

Considerando os resultados obtidos com a aplicação da heurística de *pricing*, conforme mostrado pela Tabela 5.17, podemos concluir que a heurística de *pricing* colabora com melhoria dos resultados, uma vez que sua implementação aumenta a quantidade de instâncias em que foram possíveis de se obter um *GAP*, aumenta a quantidade de soluções ótimas encontradas, diminui o *GAP* dos resultados obtidos e reduz o tempo necessário para a obtenção da solução ótima das instâncias.

CAPÍTULO

6

Conclusões

Neste trabalho apresentamos o problema das pseudo-arborescências capacitadas com localização de facilidades, um problema de otimização combinatória novo na literatura e que faz referência a problemas envolvendo a entrega de bens e serviços. Devido à inexistência de bibliografia, e consequentemente de instâncias de testes para o problema, este trabalho dedicou-se também à criação do primeiro conjunto de instâncias para o CPAFLP, criado a partir da adaptação das instâncias do CRTP propostas em Hill (2012).

Apresentamos também dois modelos matemáticos para o problema, sendo o primeiro um modelo *multi-commodity flow* (MCF), o qual foi resolvido por um algoritmo exato que utilizou o método de *branch-and-bound*, e o segundo um modelo de partição de conjuntos (SPF) que foi resolvido por um algoritmo exato baseado no método de *branch-and-price*.

Além dos modelos e algoritmos exatos, duas heurísticas também foram propostas com o objetivo de acelerar a execução dos algoritmos. A heurística primal teve como objetivo prover uma base inicial para o método de geração de colunas e fornecer uma solução inicial, enquanto a heurística de *pricing* teve como objetivo gerar pseudo-arborescências de custo reduzido negativo.

A partir dos modelos matemáticos e heurísticas desenvolvidas, foram implementados cinco programas nominados por MCF-, MCF, SPF-, SPF e SPF*. Cada programa possui uma combinação diferente de modelo e heurística, e foi executado sob as mesmas condições e nos mesmos grupos de instâncias. Ao todo foram realizados 900 testes.

Através dos experimentos foi possível concluir que o modelo (SPF) fornece limitantes duais muito mais apertados que o modelo (MCF), porém a execução

dos programas baseados neste modelo exigiram mais tempo de processamento e não foram capazes de encontrar resultados para todas as instâncias em 1800 segundos.

De maneira geral a implementação da heurística primal melhorou substancialmente o desempenho tanto do algoritmo de *branch-and-price* como do algoritmo de *branch-and-bound*.

Os experimentos também mostraram que a heurística de *pricing* colabora para a melhoria do desempenho do algoritmo de *branch-and-price*.

Os resultados do trabalho foram apresentados no *VIII Latin-American Algorithms, Graphs and Optimization Symposium*, e publicados sob o título **The Ring Tree Facility Location Problem** na revista *Electronic Notes in Discrete Mathematics* (ENDM), Abe et al. (2015).

Após o desenvolvimento desse estudo, novos trabalhos podem ser ramificados, ficando estes como sugestões de trabalhos futuros:

- Avaliar o uso de relaxações para o problema de *pricing*;
- Avaliar o comportamento dos programas em uma variedade maior de instâncias, aplicando outros valores para os custos de instalação de facilidades e quantidade de clientes em cada arborescência; e
- acelerar o algoritmo de *branch-and-price* através da utilização de outras buscas locais.

Referências Bibliográficas

Abe, F. H. N., Hoshino, E. A., e Hill, A. (2015). The ring tree facility location problem. *Electronic Notes in Discrete Mathematics*, 50:331–336. Citado na página 72.

Achterberg, T. (2009). Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41. http://mpc.zib.de/index.php/MPC/article/view/4. Citado na página 54.

Baldacci, R., Dell'Amico, M., e González, J. S. (2007). The capacitated m-ring-star problem. *Operations Research*, 55(6):1147–1162. Citado nas páginas 8 e 9.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., e Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329. Citado na página 21.

Christofides, N., Mingozzi, A., e Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical programming*, 20(1):255–282. Citado na página 8.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812. Citado na página 43.

Dantzig, G. B. e Ramser, J. H. (1959). The truck dispatching problem. *Management science*, 6(1):80–91. Citado na página 8.

Dell'Amico, M., Maffioli, F., e Värbrand, P. (1995). On prize-collecting tours and the asymmetric travelling salesman problem. *International Transactions in Operational Research*, 2(3):297–308. Citado na página 29.

Desaulniers, G., Desrosiers, J., e Solomon, M. M. (2005). *Column generation*, volume 5. Springer Science & Business Media. Citado na página 21.

Du Merle, O., Villeneuve, D., Desrosiers, J., e Hansen, P. (1999). Stabilized column generation. *Discrete Mathematics*, 194(1):229–237. Citado na página 21.

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240. Citado na página 46.

Ellson, J., Gansner, E., Koutsofios, L., North, S. C., e Woodhull, G. (2001). Graphviz - open source graph drawing tools. In *Graph Drawing*, páginas 483–484. Springer. Citado na página 68.

Feillet, D., Dejax, P., e Gendreau, M. (2005). Traveling salesman problems with profits. *Transportation Science*, 39(2):188–205. Citado na página 29.

Feo, T. A. e Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2):67–71. Citado na página 23.

Feo, T. A. e Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133. Citado na página 23.

Fukasawa, R., Longo, H., Lysgaard, J., de Aragão, M. P., Reis, M., Uchoa, E., e Werneck, R. (2006). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106(3):491–511. Citado na página 8.

Glover, F. e Kochenberger, G. A. (2003). *Handbook of metaheuristics*. Springer. Citado nas páginas 23 e 25.

Gondzio, J. e Sarkissian, R. (1996). Column generation with a primal-dual method. *Logilab, HEC Geneva, Section of Management Sciences, University of Geneva*, 102. Citado na página 21.

Hill, A. (2012). Modeling techniques in tree and ring structure based locational network design. In *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2012 4th International Congress on*, páginas 322–327. Citado nas páginas 1, 49, 68, e 71.

Hill, A. e Voß, S. (2014). Optimal capacitated ring trees. *EURO Journal on Computational Optimization*, páginas 1–30. Citado na página 27.

Hoshino, E. A. e De Souza, C. C. (2012). A branch-and-cut-and-price approach for the capacitated m-ring-star problem. *Discrete Applied Mathematics*, 160(18):2728–2741. Citado na página 9.

Hoshino, E. A. e Hill, A. (2014). Column generation approach for the capacitated ring tree problem. *ALIO/EURO*. Citado nas páginas 3 e 7.

Johnson, D. S. (1973). Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, páginas 38–49. ACM. Citado na página 11.

Koutsofios, E., North, S., et al. (1991). Drawing graphs with dot. Relatório técnico, Technical Report 910904-59113-08TM, AT&T Bell Laboratories, Murray Hill, NJ. Citado na página 68.

Land, A. H. e Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, páginas 497–520. Citado na página 18.

Lübbecke, M. E. e Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, 53(6):1007–1023. Citado na página 21.

Luna, H. P. e Goldbarg, M. C. (2000). Otimização combinatória e programação linear. *Rio de Janeiro: Campus*. Citado nas páginas 13, 14, e 15.

Miller, C., Tucker, A., e Zemlin, R. (1960). Integer programming formulations and traveling salesman problems. *Journal of ACM*, 7:326–329. Citado na página 32.

Mittelmann, H. (2007). Recent benchmarks of optimization software. In *22nd Euorpean Conference on Operational Research*. Citado na página 54.

Naji-Azimi, Z., Salari, M., e Toth, P. (2010). A heuristic procedure for the capacitated m-ring-star problem. *European Journal of Operational Research*, 207(3):1227–1234. Citado na página 9.

Puccini, A. d. L. (1972). *Introdução à programação linear*. Livro Tecnico. Citado na página 13.

Rousseau, L.-M., Gendreau, M., e Feillet, D. (2007). Interior point stabilization for column generation. *Operations Research Letters*, 35(5):660–668. Citado na página 21.

Ryan, D. M. e Foster, B. A. (1981). An integer programming approach to scheduling. *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, páginas 269–280. Citado na página 22.

Toth, P. e Vigo, D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1):487–512. Citado na página 8.

Vance, P. H., Barnhart, C., Johnson, E. L., e Nemhauser, G. L. (1994). Solving binary cutting stock problems by column generation and branch-and-bound. *Computational optimization and applications*, 3(2):111–130. Citado na página 21.

Vanderbeck, F. e Wolsey, L. A. (1996). An exact algorithm for ip column generation. *Operations research letters*, 19(4):151–159. Citado na página 21.

Wolsey, L. A. (1998). *Integer programming*. Wiley, New York. Citado nas páginas 11, 12, 16, e 20.