COPPE UFRJ
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

ADIÇÃO, COMPLEMENTO E REVERSÃO DE ARCOS PARA SATISFAZER DEMANDAS DE CONECTIVIDADE EM DIGRAFOS

Matheus Abreu da Costa Corrêa

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

Orientador: Abilio Lucena de Pereira Filho

Rio de Janeiro
Maio de 2022

ADIÇÃO, COMPLEMENTO E REVERSÃO DE ARCOS PARA SATISFAZER DEMANDAS DE CONECTIVIDADE EM DIGRAFOS

Matheus Abreu da Costa Corrêa

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

Orientador: Abilio Lucena de Pereira Filho

Aprovada por: Prof. Abilio Lucena de Pereira Filho
Prof. Laura Silvia Bahiense da Silva Leite
Prof. Adria Ramos de Lyra

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 2022

Corrêa, Matheus Abreu da Costa

Adição, complemento e reversão de arcos para satisfazer demandas de conectividade em digrafos/Matheus Abreu da Costa Corrêa. – Rio de Janeiro: UFRJ/COPPE, 2022.

XII, 60 p.: il.; 29, 7cm.

Orientador: Abilio Lucena de Pereira Filho

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2022.

Referências Bibliográficas: p. 56 – 60.

1. conectividade em digrafos. 2. adição, complemento e reversão de arcos. 3. formulações matemáticas. 4. soluções exatas. I. Pereira Filho, Abilio Lucena de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Ao Deus Trino

Agradecimentos

Agradeço à minha mãe e irmã por me apoiarem desde quando estava na minha graduação.

Quero agradecer a Lemobs, a empresa em que trabalho desde o meu estágio, pelo apoio prestado a mais uma fase da minha vida acadêmica que acabo de concluir.

Sou grato ao meu orientador, prof. Abilio Lucena, por toda sugestão valiosa à escrita deste texto. A Felipe Crispim agradeço imensamente por haver me ajudado nas muitas dúvidas que tive, mesmo que a maioria tenha sido no tema anterior, o qual não chegou a se concretizar num trabalho de dissertação.

Meu sincero obrigado a todos os professores com que já tive aulas, desde o primário até o presente momento. Grande parte desta conquista se deve a vocês.

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

ADIÇÃO, COMPLEMENTO E REVERSÃO DE ARCOS PARA SATISFAZER DEMANDAS DE CONECTIVIDADE EM DIGRAFOS

Matheus Abreu da Costa Corrêa

Maio/2022

Orientador: Abilio Lucena de Pereira Filho

Programa: Engenharia de Sistemas e Computação

Considere um digrafo com custos associados a seus arcos e sujeito a operações de adição, complemento ou reversão de arcos. Quando aplicadas, as operações devem ser todas de um mesmo tipo, predefinido, e têm por objetivo obter um digrafo que atenda a certos requisitos de conectividade. Além disso, o custo total das operações efetuadas deve ser sempre o menor possível. Problemas desse tipo, muitas vezes NP-difíceis, são normalmente associados ao desenho de redes de telecomunicações ou de transportes. Nessas aplicações, o nível de conectividade imposto aos digrafos se reflete no grau de resiliência, flexibilidade ou eficiência de funcionamento que se quer obter para as redes. Formulações matemáticas e algoritmos exatos são aqui propostos para alguns desses problemas, em alguns casos pela primeira vez na literatura. Resultados computacionais comprovam que as formulações propostas são fortes e isto se reflete no bom desempenho dos algoritmos utilizados para resolvê-las.

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

ARC ADDITION, COMPLEMENT AND REVERSAL TO ATTAIN CONNECTIVITY REQUIREMENTS IN DIGRAPHS

Matheus Abreu da Costa Corrêa

May/2022

Advisor: Abilio Lucena de Pereira Filho

Department: Systems Engineering and Computer Science

Consider a digraph with costs associated with its arcs and subject to operations of addition, complement or reversal of arcs. When applied, these operations must be of a single, predefined type, aimed at obtaining a digraph that satisfies certain connectivity requirements. Moreover, the total cost of the operations must be minimum. Problems such as these are mostly NP-hard and are commonly associated with the design of particular telecommunication or transport networks. Quite frequently, the connectivity level imposed on the digraphs is reflected by the reseliency, flexibility or operation efficiency required by the underlying networks. Mathematical formulations and exact algorithms are proposed here for some of these problems, in some cases for the very first time in the literature. Computational results indicate that the proposed formulations are strong, as reflected by the good performance of their accompanying algorithms.

Sumário

Lista de Figuras **x**

Lista de Tabelas **xi**

Lista de Abreviaturas **xii**

1 Introdução **1**

1.1 Revisão da Literatura 3

1.2 Organização dos Capítulos Seguintes 6

2 Formulações Matemáticas **7**

2.1 Uma Formulação para o Problema de Reversão de Arcos 9

2.1.1 Uma avaliação empírica da formulação (2.5) 11

2.1.2 Uma formulação para o 1-PRSS 16

2.1.3 Uma formulação para o 1-PRsT 18

2.2 Formulações para Problemas de Complemento de Arcos 19

2.2.1 Uma formulação para o 1-PCSS 19

2.2.2 Uma formulação para o 1-PCsT 20

2.3 Formulações de Problemas de Adição de Arcos 21

2.3.1 Formulações para o 1-PASS e o 1-PAsT 22

2.3.2 Formulações para o 1-PAVV, 2-PAVV e o 3-PAVV 23

3 Resolução Exata das Formulações **24**

3.1 Imposições algorítmicas de nossas formulações 25

3.2 Algoritmos Branch-and-Bound 26

3.3 Algoritmos Branch-and-Cut 27

3.4 Algoritmos de planos de corte 30

3.4.1 Separação de soluções não inteiras 31

3.4.2 Separação de soluções inteiras 32

3.5 Experimentos adicionais para o 1-PRVV 34

3.6 Identificação de todos os nossos algoritmos BC 36

3.6.1 Algoritmos BC e suas eventuais implementações 37

4 Experimentos Computacionais **39**
4.1 Experimentos computacionais para o 1-PRSS 40
4.2 Experimentos computacionais para o 1-PRsT 43
4.3 Experimentos computacionais para o 1-PCSS 45
4.4 Experimentos computacionais para o 1-PCsT 46
4.5 Experimentos computacionais para o 1-PASS 47
4.6 Experimentos computacionais para o 1-PAsT 48
4.7 Experimentos computacionais para o k-PAVV 49

5 Conclusão e Trabalhos Futuros **52**

Referências Bibliográficas **56**

Lista de Figuras

1.1 Exemplo de execução das operações sobre arcos 2

2.1 Cortes de arcos $\delta(X)^-$ e $\delta(Y)^+$ 8
2.2 $D = (V, A)$ com fonte s e sumidouro t 8
2.3 Identificando desigualdades violadas 12
2.4 Transformação de um grafo $G = (V, E)$ 14

Lista de Tabelas

2.1 Resultados para a relaxação linear da formulação para o 1-PRVV . . 16

3.1 Resultados para a relaxação linear da formulação para o 1-PRVV com separação de soluções inteiras 35

4.1 Resultados computacionais para os algoritmos **BC**, **LC** e **BC-LC** . . 41
4.2 Resultados computacionais para o algoritmo **BC-LC-RI** 41
4.3 Resultados computacionais dos algoritmos **BC**, **LC** e **BC-LC** 43
4.4 Resultados computacionais para o algoritmo **BC-LC-RI** 44
4.5 Resultados computacionais dos algoritmos **BC**, **LC** e **BC-LC** 45
4.6 Resultados computacionais para o algoritmo **BC-LC-RI** 46
4.7 Resultados computacionais dos algoritmos **BC**, **LC** e **BC-LC** 47
4.8 Resultados computacionais para o algoritmo **BC-LC-RI** 47
4.9 Resultados computacionais para os algoritmos **BC**, **LC** e **BC-LC** . . 48
4.10 Resultados computacionais para o algoritmo **BC-LC-RI** 48
4.11 Resultados computacionais para os algoritmos **BC**, **LC** e **BC-LC** . . 49
4.12 Resultados computacionais para o algoritmo **BC-LC-RI** 49
4.13 Resultados computacionais para os algoritmos **BC**, **LC** e **BC-LC** . . 51
4.14 Resultados computacionais para o algoritmo **BC-LC-RI** 51

Lista de Abreviaturas

BB	Branch-and-Bound, p. 24
BC	Branch-and-Cut, p. 10
K-PA	Problema de Adição de Arcos, p. 2
K-PC	Problema de Complemento de Arcos, p. 2
K-PR	Problema de Reversão de Arcos, p. 2
LC	Lazy Constraints, p. 29
PAG	Problema do Acréscimo em Grafos, p. 4
PAIRU	Problema do Agendamento de Interrupções em Redes Urbanas, p. 5
PCC	Problema de Cobertura de Código, p. 13
PFM	Problema de Fluxo Máximo, p. 5
PLI	Programação Linear Inteira, p. 5
PRSI	Problema de Redes Sujeitas a Interrupções, p. 5
PSG	Problema de Steiner em Grafos, p. 13
PVA	Projetos de Sistemas de Veículos Autômatos, p. 5
TFC	Teorema do Fluxo Máximo - Corte Mínimo, p. 3

Capítulo 1

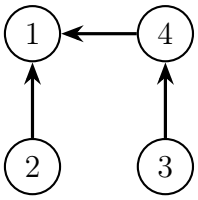
Introdução

Seja $D = (V, A)$ um digrafo definido por um conjunto de vértices V e um conjunto de arcos A tais que $n = |V|$ e $m = |A|$. Dados um *vértice de origem* s e um *vértice de destino* t, onde $s, t \in V$, a *arco-conectividade-st* de D corresponde ao número máximo de *caminhos orientados elementares* arco-disjuntos que levam de s a t, para $s \neq t$, em que caminhos elementares referem-se àqueles sem repetição de vértices. Se D possui arco-conectividade-s-t maior ou igual a $k \in \mathbb{N}$ para todo par orientado de vértices de V, ele é dito fortemente *k-arco-conexo*.

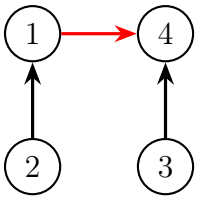
Nesta dissertação introduzimos formulações matemáticas e algoritmos de solução exata para vários problemas de arco-conectividade investigados por ARKIN *et al.* [1]. Para cada um deles, um digrafo $D = (V, A)$ é dado e são definidos subconjuntos não nulos de vértices de origem, $S \subset V$, e vértices de destino, $T \subset V$. Além disso, são permitidas operações de *adição*, *reversão* ou *complemento* de arcos de forma a atender a certos requisitos de arco-conectividade-$\{st\}$ impostos a todo par $s \in S$ e $t \in T$.

Operações de adição de arcos correspondem a incluir em D *arcos ausentes* do mesmo. Arcos ausentes de D são aqueles que o tornariam um digrafo completo. Por sua vez, reverter um arco $(i, j) \in A$ corresponde a substitui-lo por $(j, i) \notin A$. Finalmente, operações de complemento de arcos correspondem a uma forma restrita de adição de arcos em que um arco $(j, i) \notin A$ só pode ser adicionado a D quando $(i, j) \in A$. Para cada tipo de operação e para cada arco ao qual ela se aplique é associado um custo de execução da operação. Na figura 1.1, há a exposição das três operações anteriores. A figura 1.1a) mostra o digrafo D sem nenhuma modificação, enquanto as três seguintes apresentam a execução de cada operação sobre ele.

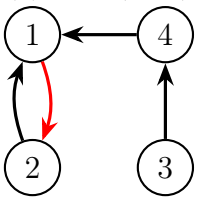
Apenas um único tipo de operação, definida a priori, é permitida em qualquer um dos problemas investigados por ARKIN *et al.* [1]. Estes são quase sempre $\mathcal{NP}$-Difíceis (é o caso de todos os que selecionamos para investigar) e o que se deseja é transformar $D = (V, A)$ em um digrafo $D' = (V, A')$ que atenda (ao menor custo total possível) a certos requisitos de conectividade predefinidos. Problemas desse tipo



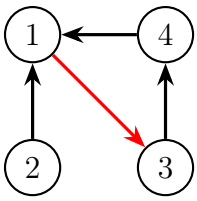
(a) $D = (V, A)$



(b) Reversão do arco $a = (4, 1)$



(c) Complemento do arco $a = (2, 1)$



(d) Adição do arco $a = (1, 3)$

Figura 1.1: Exemplo de execução das operações sobre arcos

encontram aplicações no desenho de redes de telecomunicações ou de transportes. A conectividade imposta à D' se refletindo no grau de resiliência, flexibilidade ou eficiência de funcionamento que se deseja impor à rede física modelada por D. Esses problemas podem também afetar a sociedade positivamente nos âmbitos social, econômico e ambiental quando bem resolvidos HUANG *et al.* [2]. Isto porque, se os digrafos estiverem modelando vias de um centro urbano, um menor custo oriundo da execução das operações sobre arcos significa menores transformações sobre essas vias, ou seja, um número menor de inversões do sentido delas e também de obras para a construção e manutenção das mesmas. Isso resulta em menos gastos de recursos públicos, assim como em uma menor interferência no comportamento dos motoristas daquela região, visto que eles precisam habituar-se às mudanças impostas pelas alterações decorrentes da execução das operações anteriores. Se, além da minimização do custo das operações sobre os arcos, levarmos também em conta a minimização do tamanho dos menores caminhos entre todos os pontos de um centro urbano, o tempo de viagem dos motoristas será diminuído, resultando em menos emissão de CO_2 e gastos com combustíveis.

Os problemas investigados em [1] são caracterizados pelo único tipo de operação que permitem e pelos conjuntos de vértices S e T sobre os quais requisitos de k-arco-conectividade são impostos. Vamos denotá-los genericamente como Problema de Adição de Arcos (k-PA), Problema de Reversão de Arcos (k-PR) e Problema de Complemento de Arcos (k-PC). Levando-se em conta os subconjuntos de vértices S e T previamente definidos, chega-se então ao k-PAST, ao k-PRST e ao k-PCST, respectivamente. Sendo ainda mais específico, quando $S = T$ se aplica, os problemas são denotados simplesmente por k-PASS, k-PRSS e k-PCSS. Finalmente, tomando o k-PAST como exemplo, quando $S = \{s\}$ ou $T = \{t\}$, passamos a ter o k-PAsT, o k-PAst, o k-PASt ou o k-PAss, dependendo da situação envolvida.

Nesta dissertação, vamos investigar os seguintes problemas: k-PAVV (para $k \in \{1, 2, 3\}$), 1-PASS, 1-PAsT, 1-PRSS, 1-PRsT, 1-PCSS e o 1-PCsT.

É importante ressaltar que, além de arco-conectividade, seu conceito análogo em termos de vértices, vértice-conectividade, é também investigado na literatura. Neste caso, são considerados caminhos orientados elementares *internamente-vértice-disjuntos* que levam de s a t (vértices internos correspondendo, em qualquer um desses caminhos, aos vértices distintos de s e t). Mais especificamente, um digrafo $D = (V, A)$ é dito fortemente *k-vértice-conexo* quando o número de caminhos internamente-vértice-disjuntos que levam de s para t, para todo $s, t \in V$, $s \neq t$, é maior ou igual a k. Ambas as definições de conectividade resultam do Teorema de Menger (veja DIESTEL [3] para detalhes). Este é apresentado na literatura de formas distintas e, dependendo do ponto de partida, arco-conectividade, por exemplo, um corolário do teorema (veja [3]) levaria à definição de vértice-conectividade que acabamos de apresentar. Da mesma forma, corolários adicionais (veja [3], mais uma vez) levariam também às definições de conectividade em grafos não direcionados que apresentaremos a seguir.

Dado um grafo não direcionado, $G = (V, E)$, definido por um conjunto de vértices V e um conjunto de arestas E, este é dito *k-aresta-conexo* quando contém k ou mais caminhos elementares aresta-disjuntos entre qualquer par de vértices $s, t \in V$, com $s \neq t$. Por sua vez, G é dito *k-vértice-conexo* quando os caminhos são vértice-disjuntos.

Além do Teorema de Menger, as contribuições de ROBBINS [4] e o Teorema do Fluxo Máximo - Corte Mínimo (TFC) (veja, por exemplo, [3]) são subsídios fundamentais para o estudo de conectividade em grafos, orientados ou não. Em mais detalhes, o Teorema de Menger nos permite formular matematicamente os problemas aqui investigados e o TFC, que será enunciado no Capítulo 2, nos permite resolver tais formulações.

Em outro resultado fundamental para o estudo de conectividade em grafos, ROBBINS [4] identificou que um grafo não orientado $G = (V, E)$ é 2-aresta-conexo se, e somente se, existir uma orientação $D = (V, A)$ para suas arestas tal que, para cada par de vértices distintos $i, j \in V$, deve existir um caminho orientado elementar em D indo de i para j e outro indo de j para i. Orientações de grafos levam a problemas conceitualmente próximos aos que vamos aqui investigar.

1.1 Revisão da Literatura

Baseados nos resultados teóricos introduzidos por ROBBINS [4], algoritmos para orientar as arestas de um grafo $G = (V, E)$ em um digrafo fortemente arco-conexo, $D = (V, A)$, foram propostos inicialmente por ROBERTS [5]. Orientar as arestas

de G significando associar à cada $e = \{i, j\} \in E$ um único arco (i, j) ou (j, i) em D. Posteriormente, CHVÁTAL e THOMASSEN [6] definiram métricas para avaliar e comparar diferentes orientações de G, sob diferentes condições de arco-conectividade forte impostas a D.

Mais recentemente, MARTINS *et al.* [7] generalizaram o problema de orientação de arestas, associando *pesos* aos arcos de $D = (V, A)$ e exigindo que a soma dos pesos dos caminhos de peso mínimo entre todos os pares de vértices distintos $i, j \in V$ seja a menor possível. Uma formulação matemática para o problema, assim como um algoritmo de geração de colunas para resolvê-lo, são propostos em [7].

O Problema do Acréscimo em Grafos (PAG), introduzido por ESWARAN e TARJAN [8], é definido sobre um grafo não orientado, $G = (V, E)$, com custos associados a suas *arestas ausentes*, $\{e = \{i, j\} : i, j \in V, e \notin E\}$. Seu objetivo é identificar um conjunto de arestas ausentes de menor custo total que, quando adicionadas a $G = (V, E)$, fazem com que o grafo resultante, $G' = (V, E')$, satisfaça a certas condições de conectividade predefinidas. Os autores demostram que, quando todas as arestas ausentes têm o mesmo custo, o problema pode ser resolvido em tempo polinomial. No entanto, sua versão para custos gerais é demonstrada ser $\mathcal{NP}$-Difícil [8]. O PAG, vale notar, pode ser facilmente estendido a digrafos, considerando-se, neste caso, arcos ausentes em vez das arestas ausentes. Uma especialização da forma padrão do problema para grafos planares foi investigada por KANT [9]. Por sua vez, aplicações do PAG na área de segurança de dados foram investigadas por KAO [10] e LIU *et al.* [11].

Ainda em relação ao PAG, FRANK [12] investigou uma versão k-{aresta, arco}-conexa do mesmo ($k \geq 2$ para grafos não direcionados e $k \geq 1$ para grafos direcionados). Tal versão não associa custos à arestas ou arcos. Para o caso 2-aresta-conexo, LJUBIC *et al.* [13] propõem uma meta-heurística baseada em Algoritmos Genéticos. Esta mesma versão do problema foi também investigada por KHULLER *et al.* [14], juntamente com sua extensão a digrafos fortemente 1-arco-conexos. Em particular, [14] propõe algoritmos de aproximação de fator 2 para ambas as versões do problema.

Concentrando-se agora em problemas diretamente relacionados àqueles que vamos aqui investigar, ARKIN *et al.* [1] fazem uma síntese de diversos problemas de conectividade em digrafos, caracterizados por operações de adição, reversão ou complemento de arcos. A complexidade computacional de cada problema é identificada, tanto para o caso geral quanto para algumas classes específicas de digrafos. Da mesma forma, [1] identifica também a complexidade computacional de alguns casos particulares desses problemas. Mais especificamente, investigam casos que envolvem subconjuntos de vértices de origem, S, e de destino, T, específicos e/ou estruturas de custo particulares para os arcos envolvidos. Dentre os problemas estudados em

[1], optamos por investigar aqui apenas aqueles que são $\mathcal{NP}$-Difíceis.

Um resultado importante, também próximo ao nosso tema de investigação, diz respeito ao 1-PRVV (vide a nomenclatura de problemas que definimos anteriormente). GABOW [15] e FRANK [16] demonstram que o mesmo pode ser resolvido em tempo polinomial. É interessante observar que, apesar do 1-PRVV ser um problema de fácil resolução, não conseguimos encontrar na literatura nenhuma formulação linear contínua para resolvê-lo (ou seja, que leve a soluções ótimas naturalmente inteiras para o problema).

O 1-PRVV é central para nossa investigação e vamos introduzir (no próximo capítulo) uma formulação de Programação Linear Inteira (PLI) para o mesmo. Nos testes computacionais que efetuamos, a relaxação linear dessa formulação obteve (para todas as instâncias de teste consideradas) soluções ótimas naturalmente inteiras e, consequentemente, ótimas para o 1-PRVV. No entanto, não temos uma garantia teórica (dada, por exemplo, por Unimodularidade Total [17]) de que isso venha a acontecer sempre. De qualquer forma, nossos resultados empíricos são importantes pois a formulação em questão é a estrutura básica que utilizamos para formular todos os problemas $\mathcal{NP}$-Difíceis que vamos aqui investigar e esta, como indicamos, se mostrou convenientemente forte.

Um problema de reversão de arcos, com aplicação direta na gestão de vias de tráfego urbano de veículos automotores, é investigado por COCO *et al.* [18]. Ele se denomina Problema do Agendamento de Interrupções em Redes Urbanas (PAIRU) e tem, como dados de entrada, as interrupções previstas para alguns arcos de um certo digrafo (malha viária de algum centro urbano), num dado instante de tempo. Considerados os bloqueios de tráfego previstos e a consequente eliminação de seus arcos correspondentes, o problema exige, por meio de eventuais reversões de arcos, arco-conectividade forte para o digrafo resultante. O critério para a escolha dos arcos a reverter é definido pela minimização de uma função linear do comprimento total dos caminhos mínimos par-a-par que se venha a obter.

Outro problema com forte apelo prático e também próximo aos que aqui investigamos foi estudado por REBENNACK *et al.* [19]. Ele envolve a reversão de arcos de digrafos que definem instâncias do Problema de Fluxo Máximo (PFM) (vide AHUJA *et al.* [20], para detalhes do PFM). A ideia sendo a de aumentar (tanto quanto permitido) o fluxo máximo entre pares específicos de vértices predefinidos. Este mesmo problema também encontra aplicações em Projetos de Veículos Autômatos (PVA) (vide GASKINS e TANCHOCO [21], para detalhes) nos quais a ênfase é, dentre outras, evitar colisões entre os veículos e minimizar custos de deslocamentos.

Finalmente, fechando nossa revisão de literatura, HUANG *et al.* [2], investigaram o Problema de Redes Sujeitas a Interrupções (PRSI). No PRSI um digrafo fortemente conexo perde arcos (possivelmente perdendo também uma propriedade

específica de conectividade que antes possuía) e deve-se inverter o sentido de um subconjunto dos arcos que restam, de forma a recuperar a propriedade perdida. Os autores propõem uma formulação de programação inteira multiobjetivo para o problema e utilizam o *solver* CPLEX [22] para resolvê-la de forma exata. Esse procedimento de solução se mostra viável apenas para instâncias de pequeno porte. Para instâncias de maior porte, os autores sugerem e utilizam meta-heurísticas.

1.2 Organização dos Capítulos Seguintes

No Capítulo 2 sugerimos uma formulação para o 1-PRVV. Ela serve de base para formularmos, no mesmo capítulo, os demais problemas investigados nesta dissertação. No Capítulo 3 descrevemos todos os algoritmos implementados para resolver as formulações anteriores e, em sequência, no Capítulo 4, fazemos uma avaliação empírica dos mesmos, a partir de experimentos computacionais. Finalmente, no Capítulo 5, concluímos a dissertação, com uma avaliação dos resultados obtidos e sugestões para trabalhos futuros.

Capítulo 2

Formulações Matemáticas

Uma formulação matemática para o 1-PRVV é aqui introduzida e será utilizada como ponto de partida para formularmos, neste mesmo capítulo, todos os problemas $\mathcal{NP}$-Difíceis que vamos investigar. Entretanto, antes de apresentarmos as formulações, vamos definir algumas notações para as mesmas.

Dado um grafo não orientado $G = (V, E)$ e conjuntos de vértices não nulos $S \subset V$ e $V \setminus S$, um *corte* de G é definido pelo conjunto de arestas $\delta(S) = \delta(V \setminus S) = \{\{i, j\} \in E : i \in S \wedge j \in V \setminus S\}$. Nesse caso, S é denominado o conjunto de vértices que *induz (gera)* o corte $\delta(S)$.

Em particular, quando G é conexo, a remoção de todas as arestas de $\delta(S)$ (para $S \subset V$ e $V \setminus S$ não vazios) induz duas componentes conexas do mesmo. Estas são denotadas respectivamente por $(S, E(S))$ e $(V \setminus S, E(V \setminus S))$, em que $E(S)$ (resp. $E(V \setminus S)$) corresponde à todas as arestas de G com ambas as extremidades em S (resp. $V \setminus S$).

Como indicado anteriormente, um grafo G é k-aresta-conexo se, para qualquer par de vértices distintos $i, j \in V$, existem pelo menos k caminhos elementares aresta-disjuntos entre eles. De forma análoga, como descrito por BONDY e MURTY [23], basta que G não possua um corte S com $|\delta(S)| < k$ e ele será k-aresta-conexo.

Para um digrafo $D = (V, A)$, cortes são definidos de forma análoga à indicada acima. Ou seja, para conjuntos de vértices não nulos $S \subset V$ e $V \setminus S$, definimos os seguintes cortes orientados: $\delta(S)^+ = \delta(V \setminus S)^- = \{(i, j) \in A : i \in S \wedge j \in V \setminus S\}$ e $\delta(S)^- = \delta(V \setminus S)^+ = \{(j, i) \in A : i \in S \wedge j \in V \setminus S\}$. Nesse caso, S (resp. $V \setminus S$) é o conjunto de vértices que induz os cortes orientados $\delta(S)^+$ e $\delta(S)^-$ (resp. $\delta(V \setminus S)^-$ e $\delta(V \setminus S)^+$). Quando vértices $s \in S$ e $t \in V \setminus S$ são escolhidos para representar um corte $\delta(S)^+$, podemos especializar nossa notação e chamá-lo de *corte-s-t*. A figura 2.1 apresenta exemplos de cortes de arcos para os conjuntos de vértices $X = \{6\}$ e $Y = \{1, 2\}$.

Finalmente, voltando às definições de [23], um digrafo $D = (V, A)$ é denominado fortemente k-arco-conexo quando não possui cortes $\delta(S)^+$ e $\delta(S)^-$ de cardinalidades

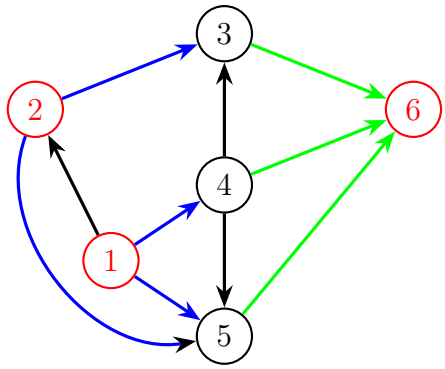


Figura 2.1: Cortes de arcos $\delta(X)^-$ e $\delta(Y)^+$

menores do que k, para qualquer par de conjuntos de vértices não vazios $S \subset V$ e $V \setminus S$.

Quando *capacidades* $\{h_{ij} \geq 0 : (i,j) \in A\}$ são associadas aos arcos de $D = (V, A)$ e dois vértices distintos $s, t \in V$ são identificados e denominados respectivamente *vértice de origem* e *vértice de destino*, D é chamado de *rede*. Neste caso, qualquer corte-s-t terá uma capacidade igual à soma das capacidades dos arcos do corte. A figura 2.2 apresenta uma rede de exemplo com capacidades h associadas aos seus arcos. Note que a capacidade do corte de arcos $\delta(\{s\})^+$ é igual a 11.

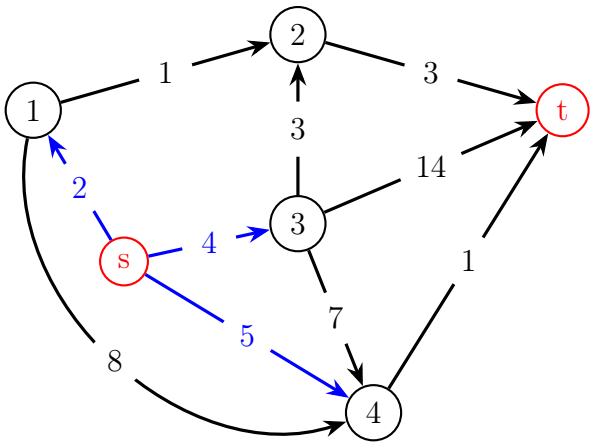


Figura 2.2: $D = (V, A)$ com fonte s e sumidouro t

Considerando uma rede, como acabamos de definir, o TFC (veja DIESTEL [3]) estabelece que o maior fluxo que se pode enviar de s para t corresponde à menor capacidade definida por um corte-s-t. Como veremos mais adiante, o TFC nos oferece a fundamentação teórica para resolver cada uma das formulações matemáticas que vamos apresentar a seguir.

2.1 Uma Formulação para o Problema de Reversão de Arcos

O 1-PRVV (na notação introduzida no capítulo 1), utiliza operações de reversão de arcos para tentar transformar um digrafo $D = (V, A)$ num digrafo $D' = (V, A')$ fortemente conexo. Isso deve ser feito ao menor custo total de reversões e, dependendo do grafo de entrada, D, o problema pode eventualmente ser inviável.

Assuma que A é particionado em subconjuntos $A^1 = \{(i,j) \in A : (j,i) \in A\}$ e $A^2 = \{(i,j) \in A : (j,i) \notin A\}$. Consequentemente, arcos candidatos à reversão devem necessariamente fazer parte do conjunto A^2, e $c_{ji} \in \mathbb{R}$ é então associado à cada $(i,j) \in A^2$, quando sua direção é invertida.

Para formular o 1-PRVV, associamos variáveis $\{x_{ij} \in [0,1] : (i,j) \in A\}$ aos arcos de A e variáveis $\{y_{ij} \in [0,1] : (j,i) \in A^2\}$ aos arcos de A que são candidatos à reversão. Consideramos então uma região poliédrica, $\mathcal{R}_0$, definida pela interseção das seguintes restrições:

$$x_{ij} + y_{ji} = 1, \ \forall (i,j) \in A^2 \tag{2.1}$$

$$\sum_{\{(i,j)\in A:\ i\in S,\ j\in V\setminus S\}} x_{ij} + \sum_{\{(j,i)\in A^2:\ i\in S,\ j\in V\setminus S\}} y_{ij} \geq 1,$$
$$\forall \{k,l\} \in V,\ k \neq l,\ \forall S \subset V,\ k \in S,\ l \in V \setminus S \tag{2.2}$$

$$0 \leq x_{ij} \leq 1, \ (i,j) \in A \tag{2.3}$$

$$0 \leq y_{ij} \leq 1, \ (j,i) \in A^2. \tag{2.4}$$

Uma formulação para o 1-PRVV é então dada por

$$v(\mathcal{R}_0) = \text{minimize } \{ \sum_{(j,i)\in A^2} c_{ij} y_{ij} : \ (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_0 \cap \mathbf{Z}^{|A|+|A^2|} \} \tag{2.5}$$

e sua relaxação linear correspondente é definida como

$$\overline{v}(\mathcal{R}_0) = \text{minimize } \{ \sum_{(j,i)\in A^2} c_{ij} y_{ij} : \ (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_0 \}, \tag{2.6}$$

com um *gap de dualidade* relativo ao valor da solução ótima dado por

$$g(\mathcal{R}_0) = \frac{v(\mathcal{R}_0) - \overline{v}(\mathcal{R}_0)}{v(\mathcal{R}_0)}. \tag{2.7}$$

Este, como se pode verificar, indica a proporção de $v(\mathcal{R}_0)$ alcançada por $\overline{v}(\mathcal{R}_0)$.

As restrições (2.1) garantem que não podemos ter simultaneamente um arco de A e a sua reversão numa mesma solução. As restrições (2.2) correspondem a todas as caracterizações de cortes viáveis para o digrafo $D' = (V, A')$ que se quer obter. Note que alguns desses cortes podem envolver arcos em suas orientações originais ou em orientações invertidas. Além disso, de acordo com [23], essas restrições são suficientes para garantir que o digrafo D' resultante seja fortemente 1-arco-conexo. Finalmente, a função objetivo em (2.5) garante que eventuais reversões serão efetuadas ao menor custo total possível.

Sendo o 1-PRVV um problema com complexidade de resolução polinomial, deve existir para ele, em tese, uma formulação em que o valor da relaxação linear seja sempre igual ao valor da solução ótima. Nessa situação, a formulação deve necessariamente apresentar certas propriedades (unimodularidade total [17], por exemplo) que lhe garantam *gaps* de dualidade nulos. Vale notar que 2.5 não parece ter propriedades desse tipo e torna-se então importante verificar empiricamente quão próximos de $v(\mathcal{R}_0)$ são os valores $\overline{v}(\mathcal{R}_0)$ que ela obtém. Para isso, vamos recorrer a experimentos computacionais envolvendo digrafos $D = (V, A)$ que serão definidos mais adiante.

A formulação (2.5) contém um número exponencial de inequações do tipo (2.2). Portanto, num algoritmo *Branch-and-Cut* (BC) (veja PADBERG e RINALDI [24]), essas restrições são inicialmente desconsideradas e utilizadas apenas à medida em que são violadas pela relaxação linear resultante. Isso é feito porque seria impraticável, à menos de casos triviais, a enumeração exaustiva de todas elas. Da mesma forma, seria impraticável resolver diretamente um prolema de Programação Linear envolvendo um número gigantesco de restrições (nesse caso, o procedimento a seguir seria utilizar a técnica de geração de colunas BERTSIMAS e TSITSIKLIS [25]). Um algoritmo específico para *separar* as desigualdades em questão (ou seja, identificá-las) será descrito em breve. No entanto, para tornar mais rápida a sua convergência, é importante ressaltar que algumas restrições de (2.2) podem ser utilizadas desde o início, como uma forma de fortalecer as relaxações lineares iniciais de (2.5). Para o 1-PRVV, as seguintes desigualdades são utilizadas com essa finalidade:

$$\sum_{(i,j)\in A} x_{ij} + \sum_{(j,i)\in A^2} y_{ij} \geq 1, \ \forall i \in V \tag{2.8}$$

$$\sum_{(j,i)\in A} x_{ji} + \sum_{(i,j)\in A^2} y_{ji} \geq 1, \ \forall i \in V. \tag{2.9}$$

Estas impõem que, em qualquer solução viável a ser obtida, todo vértice do digrafo deve possuir ao menos um arco apontando para fora do mesmo e ao menos um arco

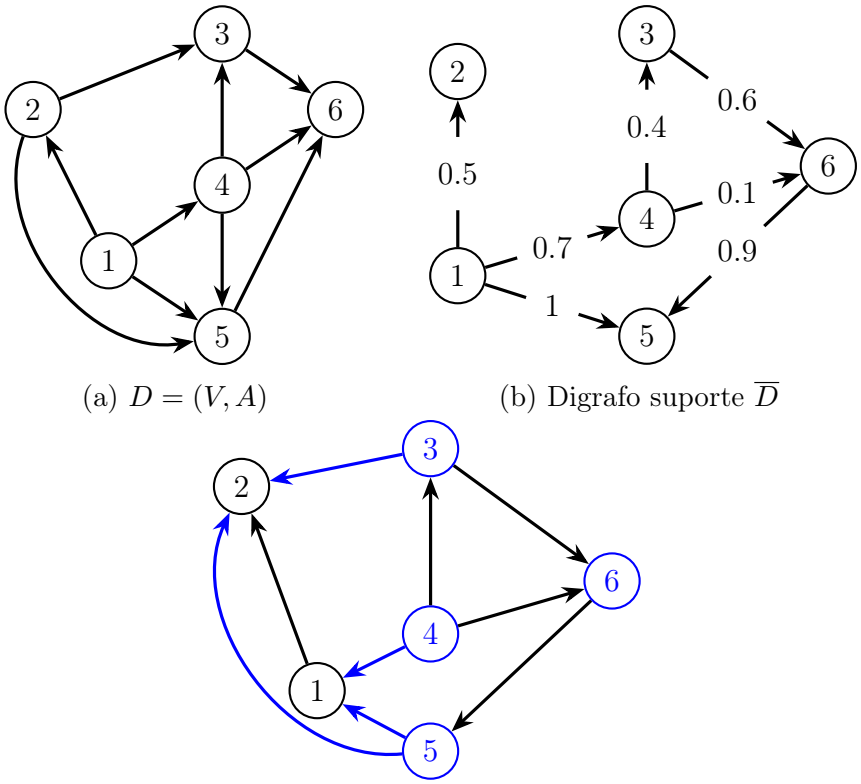
apontando para ele. Tais desigualdades são poucas e tendem a contribuir significativamente não apenas para o fortalecimento das relaxações lineares iniciais, mas também para uma convergência mais rápida para uma solução ótima definida por (2.6). Note que, no esquema proposto, apenas uma pequena fração das desigualdades definidas por (2.2) deverá ser considerada explicitamente ao longo da execução do algoritmo de planos de corte (consulte MARCHAND *et al.* [26] para maiores detalhes) que vamos propor.

Finalmente, devemos ressaltar que uma formulação multifluxos foi proposta para o 1-PRVV por HUANG *et al.* [2]. No entanto, relaxações lineares de formulações multifluxos (para problemas de conectividade em grafos) tendem a demandar um esforço computacional excessivo para a sua resolução. Em função disso, resolvemos propor a formulação alternativa introduzida acima. Formulações como as que sugerimos (para problemas de conectividade em grafos), tipicamente demandam tempos de CPU mais aceitáveis, o que é confirmado pelos experimentos computacionais que vamos apresentar a seguir. Tais experimentos têm por objetivo determinar empiricamente o quão forte é nossa formulação do 1-PRVV.

2.1.1 Uma avaliação empírica da formulação (2.5)

Iniciamos nossa avaliação empírica de (2.5) com a descrição do algoritmo utilizado para resolver sua relaxação linear (2.6) (e, por analogia, para obter relaxações lineares para eventuais subproblemas em qualquer nó de uma árvore de enumeração Branch-and-Cut (BC) [24]). Consideramos inicialmente a relaxação de (2.6) definida pela interseção das restrições (2.1), (2.3), (2.4), (2.8) e (2.9). Assumindo que $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ é uma solução ótima para ela, desigualdades de corte (2.2) violadas por $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ devem ser identificadas e acrescentadas à nossa relaxação corrente de (2.6), num procedimento de planos de corte que nos levará, eventualmente, à $\overline{v}(\mathcal{R}_0)$.

Seja $\overline{D} = (V, \overline{A})$ o *digrafo suporte* associado à $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, em que $\overline{A} = \{\{(i,j) : \overline{x}_{ij} > 0, (i,j) \in A\} \cup \{(j,i) \notin A : \overline{y}_{ij} > 0\}\}$. Para cada arco $(i,j) \in \overline{A}$ associamos uma capacidade h_{ij} igual à $\overline{x}_{ij}$, quando $(i,j) \in A$, e igual à $\overline{y}_{ij}$, quando $(i,j) \notin A$. Para cada par de vértices $s, t \in V$, calculamos o fluxo máximo de s a t, com base nas capacidades h. Se este resultar num valor menor do que 1, faz-se necessária a adição da desigualdade de corte (associada ao fluxo máximo identificado) à relaxação corrente de (2.6). Assumindo que $S \subset V$ é o conjunto de vértices que induz o corte-s-t, $\delta(S)^+$, o mesmo está então caracterizado. Após testar todos os pares de vértices possíveis, se alguma desigualdade de corte violada for identificada, a relaxação linear corrente de (2.6) deve ser reforçada com essas desigualdades e resolvida, mais uma vez. Caso contrário, na inexistência de desigualdades de cortes violadas, o procedimento é terminado. Neste estudo, para calcular o fluxo máximo entre dois



(a) $D = (V, A)$

(b) Digrafo suporte $\overline{D}$

(c) Corte de arcos $\delta(S)^+$ violado

Figura 2.3: Identificando desigualdades violadas

vértices de uma rede, utilizou-se o algoritmo de Dinic (leia DINITZ [27]).

A figura 2.3 apresenta os passos necessários à identificação de uma desigualdade de corte de arcos sendo violada. 2.3b expõe um digrafo suporte $\overline{D}$ associado a uma solução $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ qualquer. Há uma capacidade h atribuída a cada um dos seus arcos de acordo com os valores arbitrários associados às variáveis da solução hipotética em questão. Em $\overline{D}$, não é possível enviar ao menos uma unidade de fluxo do vértice 4 ao vértice 1, mas apenas a 3, 5 e 6. Isso nos concede um conjunto de vértices $S = \{3, 4, 5, 6\}$ cujo corte de arcos $\delta(S)^+$ está sendo violado. Observe que os arcos (em azul) pertencentes a tal corte não pertencem ao digrafo D originalmente, mas são eles fruto de reversões de arcos.

Para testar computacionalmente nossas formulações matemáticas para problemas de conectividade em digrafos, precisamos selecionar, inicialmente, instâncias de teste adequadas. Infelizmente, não encontramos na literatura nenhum conjunto de digrafos que cumprisse esta função. Na realidade, até mesmo para a formulação multifluxo do 1-PRVV, proposta anteriormente, inexistem instâncias sugeridas para testes. O mesmo se aplica aos demais problemas que vamos aqui investigar.

Como exemplo, para os problemas de reversão restrita de arcos investigados em [18], conceitualmente próximos aos problemas que aqui consideramos, são utilizados digrafos de teste de dimensão muito reduzida. Optou-se então pela utilização de

grafos já disponíveis em repositórios na Internet, propostos anteriormente para testar problemas distintos dos nossos. Em particular, grafos não direcionados utilizados para testar formulações para o Problema de Steiner em Grafos (PSG) HAKIMI [28]. Tais grafos são aqui transformados em digrafos, mediante regras bem definidas, gerando assim instâncias de teste, de fácil acesso, para nossos problemas.

Especificamente, utilizamos as famílas de grafos não direcionados B, C e PUC (acesse a OR-LIB[1], BEASLEY [29], para obtê-las). Os primeiros dois grupos de instâncias B BEASLEY [30] e C BEASLEY [29] representam grafos esparsos com custos aleatórios definidos sobre suas arestas, os quais variam de 1 a 10. Já o último grupo, a saber, PUC ROSSETI *et al.* [31], contém grafos bipartidos, hipercubos e aqueles produzidos a partir de instâncias do Problema de Cobertura de Código (PCC) VAN LINT [32]. As arestas desses grafos possuem custos variados, desde unitários até aqueles definidos aleatoriamente dentro de intervalos específicos de valores mínimo e máximo. Tais instâncias são especialmente adequadas para nós porque definem, de antemão, subconjuntos $T \subset V$ de vértices *terminais*. Estes cumprem adequadamente a função prevista para os subconjuntos de vértices S e T, nos problemas 1-PRSS e 1-PRsT, por exemplo. Além disso, como já informado, tais grafos possuem custos associados a suas arestas, os quais podem ser reutilizados em nossos problemas. O leitor poderá pontuar, com razão, que há muitas outras instâncias existentes para o PSG na literatura com estas mesmas características. Por isso, é importante citar que as instâncias de B e C são bem resolvidas na literatura até por algoritmos ingênuos, ao passo que as do grupo PUC revelam ser mais complexas, algumas ainda sem prova de otimalidade para o PSG. Essas últimas análises nos possibilitam então escolher grupos de instâncias que, em teoria, produzirão instâncias fáceis e mais complexas para os nossos problemas, trazendo maior diversidade aos nossos experimentos.

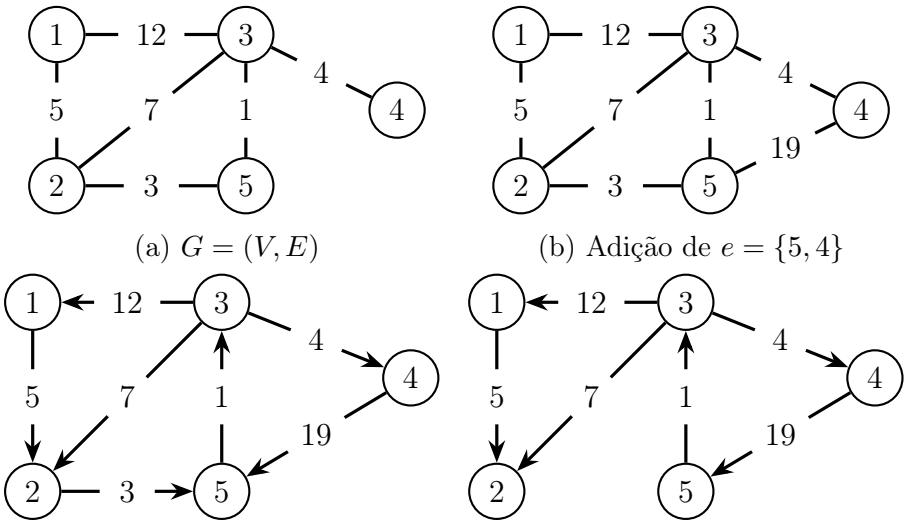
Para transformar em digrafos os grafos não direcionados, $G = (V, E)$, indicados acima, identificamos inicialmente todas as eventuais *pontes* de G. Pontes são definidas como arestas individuais de G que, se removidas do mesmo, o tornam desconexo. Identificada uma ponte, consideramos então o corte $\delta(S)$ que esta impõe a G. Feito isso, geramos aleatoriamente um certo número de arestas com uma extremidade em S e a outra em $V \setminus S$, e as adicionamos ao grafo. Assim procedendo, chegamos então, eventualmente, a um grafo que não contém pontes, ou seja, um grafo 2-aresta-conexo. Estes, como indicado por ROBBINS [4], são os que permitem uma orientação fortemente conexa de suas arestas, nos levando assim a um grafo minimamente adequado aos nossos propósitos.

Um algoritmo para transformar uma instância do PSG em uma do 1-PRVV será descrito a seguir. Dado um grafo $G = (V, E)$ associado a qualquer uma das famílias

[1]http://people.brunel.ac.uk/ mastjjb/jeb/orlib/steininfo.html

de grafos B, C ou PUC, verifica-se se ele possui alguma ponte, o que pode ser feito em tempo polinomial com o auxílio de uma busca em profundidade [33]. Existindo pontes, procedemos como sugerido no parágrafo anterior, até que, eventualmente, o grafo se torne 2-aresta-conexo. Denomine por $G' = (V, E')$ o grafo resultante do passo anterior. Aplicando a $G' = (V, E')$ o procedimento descrito em HUANG *et al.* [2], obtemos então um digrafo fortemente conexo $D = (V, A)$, como desejamos. Finalmente, de forma aleatória, arcos de D são removidos para que ele perca a arco-conectividade que possui (do contrário não haveria a necessidade de reversão de seus arcos). Esta eliminação de arcos é realizada de forma a maximizar as chances de que ainda seja possível a obtenção de um digrafo $D' = (V, A')$ fortemente conexo por meio de operações de reversão de arcos. Finalmente, para o digrafo $D = (V, A)$ eventualmente obtido, associamos aos seus arcos candidatos à reversão (logo, $(i, j) \in A^2$), os custos das arestas $\{i, j\}$ de G que lhes deram origem. No entanto, se um arco candidato à reversão foi formado a partir de uma aresta criada de forma aleatória com fins de eliminação de pontes do grafo G original, seu custo é um valor inteiro escolhido aleatoriamente entre 0 e o maior custo existente dentre as arestas de E. Após a execução do procedimento descrito, obtém-se uma nova instância para o 1-PRVV.

A figura 2.4 contém uma ilustração de cada passo do algoritmo examinado acima. 2.4b) ilustra a adição da aresta $e = \{5, 4\}$ para a remoção da ponte $p = \{3, 4\}$ em G. Em sequência, ocorre a orientação das arestas do grafo G' produzido pelo passo predecessor. Finalmente, o arco $a = (2, 5)$ é removido. O leitor sem dificuldades conseguirá perceber que a reversão dos arcos $b = (1, 2)$ e $c = (3, 1)$ leva a uma solução viável para o 1-PRVV.



(a) $G = (V, E)$ (b) Adição de $e = \{5, 4\}$

(c) Orientando as arestas de G' (d) Removendo o arco $a = (2, 5)$

Figura 2.4: Transformação de um grafo $G = (V, E)$

Utilizamos em nossos experimentos um computador equipado com um processador AMD Ryzen™3 3200G com 4 *cores* e frequência básica de 3,6GHz, 16GB de memória RAM com frequência de 2666MHz e operando sob o sistema operacional Xubuntu 20.04, uma distribuição Linux. Nossos algoritmos foram implementados na linguagem de programação C++ e compilados pelo *gcc*, em sua versão 9.3.0, sob a diretiva de compilação O3. Para a resolução da formulação foi utilizado o *solver* proprietário Gurobi na versão 9.1.2, GUROBI OPTIMIZATION, LLC [34], configurado na sua forma padrão, ou seja, sem a imposição de qualquer orientação adicional para a sua execução.

Na tabela 2.1 apresentamos os resultados obtidos com a resolução da relaxação linear (2.6), correspondente à nossa formulação (2.5) do 1-PRVV, assim como o valor ótimo de (2.5). Adiantamos que não apresentamos os resultados computacionais para todas as instâncias que geramos (por volta de 80). Se isso fosse feito, o fluxo de leitura do texto seria imensamente prejudicado. Desta forma, o leitor é convidado a consultar o apêndice deste documento, disponibilizado online em `https://github.com/pl225/instancias-dissertacao`, para obter informações sobre as demais instâncias. Tal abordagem será aqui aplicada sempre que resultados computacionais forem apresentados.

A tabela 2.1 contém 6 colunas, a primeira identificando a instância de teste, a segunda indicando seu número de vértices e a terceira sua quantidade de arcos. Em seguida, ela indica o valor da solução ótima de (2.5), sua relaxação linear (2.6) e, por último, o tempo de CPU despendido, em segundos, até o término do procedimento de solução.

Nome	$\|V\|$	$\|A\|$	$v(\mathcal{R}_0)$	$\overline{v}(\mathcal{R}_0)$	t(s)
b01	50	79	8	8	0,221
b16	100	186	16	16	1,298
c04	500	735	96	96	260,432
c13	500	1751	146	146	360,603
c16	500	2500	158	158	140,452
bip42p	1200	1967	296	296	2459,361
bipe2u	550	1003	234	234	238,264
cc3-10u	1000	2025	161	161	3630,713
cc3-11p	1331	2995	147	147	4886,271
cc3-5u	125	150	9	9	4,139
cc5-3p	243	304	69	69	25,329
cc6-3u	729	1092	170	170	537,537
cc7-3u	2187	3062	558	558	41350,221
hc10u	1024	1280	279	279	4336,245

Nome	$\|V\|$	$\|A\|$	$v(\mathcal{R}_0)$	$\overline{v}(\mathcal{R}_0)$	t(s)
hc6p	64	77	23	23	0,57
hc7u	128	157	35	35	2,307
hc8p	256	359	63	63	40,187
hc9u	512	576	75	75	236,015

Tabela 2.1: Resultados para a relaxação linear da formulação para o 1-PRVV

Observa-se, para todas as nossas instâncias de teste, que uma solução naturalmente inteira foi encontrada para a relaxação linear indicada por (2.6). Consequentemente, *gaps* de dualidade nulos, $g(\mathcal{R}_0) = 0$, resultaram para todas elas. Ou seja, embora a formulação (2.5) não tenha uma garantia teórica de levar a relaxações lineares naturalmente inteiras, isto acabou acontecendo para todas as nossas instâncias de teste. Visto que o 1-PRVV é um problema com complexidade de resolução polinomial e que nossa formulação para o mesmo serve como base para todas as outras que apresentaremos a seguir, é significativo obter, empiricamente, *gaps* de dualidade nulos para ela.

Vale observar que os tempos de CPU indicados na tabela 2.1 nos trazem um sinal de alerta. Embora nossa formulação do 1-PRVV tenha sido capaz de obter relaxações lineares invariavelmente inteiras para todas as nossas instâncias de teste, os tempos de CPU exigidos para isso foram, em alguns casos, excessivos. Isso não se deve à formulação em si, mas aos algoritmos de separação de desigualdades de cortes violados que implementamos. Estes exigem a identificação exaustiva de todas as desigualdades violadas, em todas as execuções de nossos algoritmos de separação.

No próximo capítulo, para a separação de desigualdades violadas por soluções fracionárias, apresentaremos alternativas que visam reduzir o tempo de CPU necessário para a obtenção da relaxação linear ótima, (2.6). Não obstante, adiantamos que, em algum estágio de nossos algoritmos de planos de corte, será necessário efetuar uma separação exaustiva de desigualdades de corte. Mesmo diante das diferentes propostas que investigamos para mitigar seus efeitos (a serem descritas mais adiante), este ponto ainda assim se qualifica como um dos temas que vamos propor para investigações futuras.

2.1.2 Uma formulação para o 1-PRSS

O 1-PRSS é um problema de reversão de arcos que demanda a obtenção de um digrafo fortemente conexo, $D' = (V, A')$, para cada par de vértices de um conjunto de entrada, $S \subset V$. Considerando-se as variáveis $(\mathbf{x}, \mathbf{y})$ definidas anteriormente, associe a elas uma região poliédrica $\mathcal{R}_1$, definida pela interseção das restrições (2.1), (2.3) e (2.4) com:

$$\sum_{\{(i,j)\in A:\ i\in B,\ j\in V\setminus B\}} x_{ij} + \sum_{\{(j,i)\in A^2:\ i\in B,\ j\in V\setminus B\}} y_{ij} \geq 1,$$
$$\forall \{k,l\} \in S,\ k \neq l,\ \forall B \subset V,\ k \in B,\ l \in V \setminus B. \quad (2.10)$$

Uma formulação para o 1-PRSS é então definida como

$$v(\mathcal{R}_1) = \text{minimize } \{ \sum_{(j,i)\in A^2} c_{ij} y_{ij} :\ (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_1 \cap \mathbf{Z}^{|A|+|A^2|} \} \quad (2.11)$$

e sua relaxação linear correspondente é dada por

$$\overline{v}(\mathcal{R}_1) = \text{minimize } \{ \sum_{(j,i)\in A^2} c_{ij} y_{ij} :\ (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_1 \}, \quad (2.12)$$

o que leva a um *gap* de dualidade

$$g(\mathcal{R}_1) = \frac{v(\mathcal{R}_1) - \overline{v}(\mathcal{R}_1)}{v(\mathcal{R}_1)} \quad (2.13)$$

para a formulação.

Como pode ser observado, a diferença entre as restrições (2.2) e (2.10) diz respeito apenas aos subconjuntos de vértices que dão origem a estas desigualdades de corte. Para o 1-PRVV, estes são definidos por V. Para o 1-PRSS, $S \subset V$ é utilizado. Isso demostra o grau de adaptação da formulação original para cobrir diferentes requisitos de conectividade.

Para inicializar o procedimento de obtenção de (2.12), seguimos, de forma análoga, o que sugerimos para obter (2.6). Isso se traduz aqui nas seguintes desigualdades de corte:

$$\sum_{(i,j)\in A} x_{ij} + \sum_{(j,i)\in A^2} y_{ij} \geq 1,\ \forall i \in S \quad (2.14)$$

$$\sum_{(j,i)\in A} x_{ji} + \sum_{(i,j)\in A^2} y_{ji} \geq 1,\ \forall i \in S \quad (2.15)$$

As desigualdades acima garantem que toda solução viável para o 1-PRSS deve conter pelo menos um arco apontando para fora de (resp. um arco apontando para) cada vértice $i \in S$.

2.1.3 Uma formulação para o 1-PRsT

O 1-PRsT tem por objetivo conectar um vértice $s \in V$ a um conjunto de vértices $T \subset V \setminus \{s\}$ por meio de operações de reversão de arcos. Como fizemos para o 1-PRSS, apenas algumas modificações marginais são necessárias para adaptar nossa formulação do 1-PRVV para o 1-PRsT. Continuamos a utilizar variáveis $(\mathbf{x}, \mathbf{y})$ e as elas associamos uma região poliédrica, $\mathcal{R}_2$, definida pela interseção das restrições (2.1), (2.3) e (2.4) com:

$$\sum_{\{(i,j)\in A:\ i\in B,\ j\in V\setminus B\}} x_{ij} + \sum_{\{(j,i)\in A^2:\ i\in B,\ j\in V\setminus B\}} y_{ij} \geq 1,$$
$$\forall k \in T,\ \forall B \subset V,\ s \in B,\ k \in V \setminus B. \quad (2.16)$$

Uma formulação para o 1-PRsT é então definida como

$$v(\mathcal{R}_2) = \text{minimize } \{ \sum_{(j,i)\in A^2} c_{ij} y_{ij} :\ (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_2 \cap \mathbf{Z}^{|A|+|A^2|} \}, \quad (2.17)$$

e sua relaxação linear correspondente é dada por

$$\overline{v}(\mathcal{R}_2) = \text{minimize } \{ \sum_{(j,i)\in A^2} c_{ij} y_{ij} :\ (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_2 \}, \quad (2.18)$$

o que nos leva a um *gap* de dualidade

$$g(\mathcal{R}_2) = \frac{v(\mathcal{R}_2) - \overline{v}(\mathcal{R}_2)}{v(\mathcal{R}_2)} \quad (2.19)$$

para a formulação.

As restrições (2.16) impõem a existência de pelo menos um arco em qualquer conjunto de corte separando o vértice s de qualquer vértice de T. A exemplo do que fizemos anteriormente, vamos manter sempre em nossas relaxações lineares de (2.17) algumas poucas desigualdades de corte que nos ajudam a convergir mais rapidamente para o valor ótimo de (2.18). São elas:

$$\sum_{(s,j)\in A} x_{sj} + \sum_{(j,s)\in A^2} y_{sj} \geq 1 \quad (2.20)$$

$$\sum_{(j,i)\in A} x_{ji} + \sum_{(i,j)\in A^2} y_{ji} \geq 1,\ \forall i \in T \quad (2.21)$$

A desigualdade (2.20) impõe a existência de pelo menos um arco apontando para fora de s. A desigualdade (2.21), por sua vez, impõe a existência de pelo menos um

arco entrando em cada vértice de T. Como se pode observar, as duas são casos particulares das desigualdades (2.16).

2.2 Formulações para Problemas de Complemento de Arcos

Os problemas desta seção podem ser formulados introduzindo-se alterações pontuais nas formulações que sugerimos anteriormente para os problemas k-PR. Para cada novo problema, vamos seguir o mesmo roteiro utilizado para descrever as formulações propostas anteriormente. Inicialmente, variáveis serão definidas. A seguir, uma região poliédrica a elas associada será descrita. Feito isso, uma formulação do problema será proposta, baseada naquela região poliédrica. Finalmente, concluindo o roteiro, uma relaxação linear da formulação será então apresentada, seguida da descrição do *gap* de dualidade que a mesma implica.

2.2.1 Uma formulação para o 1-PCSS

O 1-PCSS é um problema de complemento de arcos que tem por objetivo a obtenção de um digrafo fortemente conexo, $D' = (V, A')$, para cada par de vértices distintos de um conjunto de entrada $S \subset V$.

Para atender às operações de complemento de arcos, convém definir um conjunto de arcos, $A^2 = \{(i,j) : (j,i) \in A, (i,j) \notin A\}$, candidatos à inserção em D. Estes, vale notar, não pertencem a A e poderão ser inseridos em D se e, somente se, seu arco reverso pertence a A.

Para cada arco $(i,j) \in A^2$ é associado a um custo $c_{ij} \in \mathbb{R}$. Sejam $\{x_{ij} \in [0,1] : (i,j) \in A\}$ e $\{y_{ij} \in [0,1] : (i,j) \in A^2\}$ variáveis associadas aos arcos dos conjuntos A e A^2, respectivamente. Seja $\mathcal{R}_3$ uma região poliédrica associada a $(\mathbf{x}, \mathbf{y})$ e formada pela interseção das restrições (2.3) e (2.4) com:

$$x_{ij} = 1, \, \forall \, (i,j) \in A \tag{2.22}$$

$$\sum_{\{(i,j) \in A:\, i \in B,\, j \in V \setminus B\}} x_{ij} + \sum_{\{(i,j) \in A^2:\, i \in B,\, j \in V \setminus B\}} y_{ij} \geq 1,$$
$$\forall \{k,l\} \in S, \; k \neq l, \; \forall B \subset V, \; k \in B, \; l \in V \setminus B. \tag{2.23}$$

Uma formulação para o 1-PCSS é então definida como

$$v(\mathcal{R}_3) = \text{minimize} \, \{ \sum_{(i,j) \in A^2} c_{ij} y_{ij} : \; (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_3 \cap \mathbf{Z}^{|A| + |A^2|}, \} \tag{2.24}$$

e sua relaxação linear é dada por

$$\overline{v}(\mathcal{R}_3) = \text{minimize } \{ \sum_{(i,j)\in A^2} c_{ij} y_{ij} : \ (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_3 \}, \tag{2.25}$$

o que nos leva a um *gap* de dualidade

$$g(\mathcal{R}_3) = \frac{v(\mathcal{R}_3) - \overline{v}(\mathcal{R}_3)}{v(\mathcal{R}_3)} \tag{2.26}$$

para a formulação.

As restrições do tipo (2.22) indicam que todos os arcos de A devem fazer parte da solução. A função objetivo descrita por (2.24) assegura que as complementações que venham a ocorrer deverão ser efetuadas ao menor custo total possível. Note que a única diferença que pode ser apontada entre as inequações (2.23) e a (2.10) é a orientação utilizada nos arcos candidatos à complementação. Nos problemas k-PR utiliza-se a orientação (j, i) enquanto nos problemas k-PC utiliza-se (i, j), quando $(i, j) \in A^2$. No que tange às desigualdades iniciais de corte que vamos utilizar, procedemos de forma semelhante ao que foi sugerido para a obtenção de (2.12):

$$\sum_{(i,j)\in A} x_{ij} + \sum_{(i,j)\in A^2} y_{ij} \geq 1, \ \forall i \in S \tag{2.27}$$

e

$$\sum_{(j,i)\in A} x_{ji} + \sum_{(j,i)\in A^2} y_{ji} \geq 1, \ \forall i \in S. \tag{2.28}$$

A ressaltar aqui, apenas a orientação distinta dada aos arcos candidatos à complementação, quando comparado ao que impomos aos problemas de reversão de arcos, k-PR.

2.2.2 Uma formulação para o 1-PCsT

Utilizando operações de complemento de arcos, o 1-PCsT tem por objetivo conectar (ao menor custo total possível) um vértice $s \in V$ a um conjunto de vértices $T \subset V \setminus \{s\}$. A formulação que propomos para este problema é muito semelhante àquela sugerida para o 1-PRsT, na seção 2.1.3. Pequenas diferenças ocorrem, quais sejam a utilização de um conjunto A^2 e o uso distinto de sentido de orientação para os arcos candidatos à complementação, nos mesmos moldes descritos para o 1-PCSS na seção 2.2.1.

Utilizamos aqui as mesmas variáveis $(\mathbf{x}, \mathbf{y})$ definidas anteriormente e as associamos agora a uma região poliédrica $\mathcal{R}_4$ definida pela interseção das restrições (2.3), (2.4) e (2.22) com

$$\sum_{\{(i,j)\in A:\ i\in B,\ j\in V\setminus B\}} x_{ij} + \sum_{\{(i,j)\in A^2:\ i\in B,\ j\in V\setminus B\}} y_{ij} \geq 1,$$

$$\forall k \in T,\ \forall B \subset V,\ s \in B,\ k \in V \setminus B. \quad (2.29)$$

Uma formulação para o 1-PCsT é então definida como

$$v(\mathcal{R}_4) = \text{minimize } \{ \sum_{(i,j)\in A^2} c_{ij} y_{ij} :\ (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_4 \cap \mathbf{Z}^{|A|+|A^2|} \} \quad (2.30)$$

e sua relaxação linear correspondente é dada por

$$\overline{v}(\mathcal{R}_4) = \text{minimize } \{ \sum_{(i,j)\in A^2} c_{ij} y_{ij} :\ (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_4 \}, \quad (2.31)$$

o que nos leva a um *gap* de dualidade

$$g(\mathcal{R}_4) = \frac{v(\mathcal{R}_4) - \overline{v}(\mathcal{R}_4)}{v(\mathcal{R}_4)}. \quad (2.32)$$

Para inicializar o procedimento de solução para a formulação, utilizamos as seguintes desigualdades de corte:

$$\sum_{(s,j)\in A} x_{sj} + \sum_{(s,j)\in A^2} y_{sj} \geq 1 \quad (2.33)$$

e

$$\sum_{(j,i)\in A} x_{ji} + \sum_{(j,i)\in A^2} y_{ji} \geq 1,\ \forall i \in T. \quad (2.34)$$

2.3 Formulações de Problemas de Adição de Arcos

Formulações de problemas que permitem apenas operações de adição de arcos são sugeridas a seguir. Como antes, continuamos a tomar como base para essas formulações aquela sugerida para o 1-PRVV. Iniciamos a seção com formulações para o 1-PASS e o 1-PAsT, seguidas de formulações para o k-PAVV, $k \in \{1, 2, 3\}$. Antes de apresentá-las, porém, salientamos que o conjunto de arcos A^2 (aqueles passíveis de se adicionar ao grafo de entrada, $D = (V, A)$) é definido sem restrições adicionais e corresponde ao complemento dos arcos de D (ou seja, aqueles que o transformam um digrafo completo).

2.3.1 Formulações para o 1-PASS e o 1-PAsT

A única diferença entre os problemas de adição de arcos 1-PASS e 1-PAsT e os de complemento de arcos 1-PCSS e 1-PCsT diz respeito à definição do conjunto A^2 (ou seja, o de arcos candidatos à adição). Redefinido A^2 (como sugerido no início dessa seção), as formulações se tornam basicamente as mesmas.

Para formular o 1-PASS, consideremos as mesmas variáveis utilizadas em nossas formulações dos problemas k-PC e associemos a elas uma região poliédrica, $\mathcal{R}_5$, definida pela interseção das restrições (2.3), (2.4), (2.22) e (2.23). Feito isso, o 1-PASS pode ser formulado como

$$v(\mathcal{R}_5) = \text{minimize } \{ \sum_{(i,j)\in A^2} c_{ij} y_{ij} : (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_5 \cap \mathbf{Z}^{|A|+|A^2|} \} \tag{2.35}$$

e a sua relaxação linear correspondente é dada por

$$\overline{v}(\mathcal{R}_5) = \text{minimize } \{ \sum_{(i,j)\in A^2} c_{ij} y_{ij} : (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_5 \}, \tag{2.36}$$

o que leva a um *gap* de dualidade

$$g(\mathcal{R}_5) = \frac{v(\mathcal{R}_5) - \overline{v}(\mathcal{R}_5)}{v(\mathcal{R}_5)}. \tag{2.37}$$

Como desigualdades de corte para inicializar nosso procedimento de solução para o 1-PASS, utilizamos (2.27) e (2.28).

Concentrando-nos agora numa formulação do 1-PAsT, consideramos uma região poliédrica, $\mathcal{R}_6$, definida pela interseção das restrições (2.3), (2.4), (2.22) e (2.29). Uma formulação para o problema é então dada por

$$v(\mathcal{R}_6) = \text{minimize } \{ \sum_{(i,j)\in A^2} c_{ij} y_{ij} : (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_6 \cap \mathbf{Z}^{|A|+|A^2|} \} \tag{2.38}$$

e sua relaxação linear é definida como

$$\overline{v}(\mathcal{R}_6) = \text{minimize } \{ \sum_{(i,j)\in A^2} c_{ij} y_{ij} : (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_6 \}, \tag{2.39}$$

o que leva a um *gap* de dualidade

$$g(\mathcal{R}_6) = \frac{v(\mathcal{R}_6) - \overline{v}(\mathcal{R}_6)}{v(\mathcal{R}_6)}. \tag{2.40}$$

Para inicializar nosso procedimento de solução utilizamos as desigualdades (2.33) e (2.34).

2.3.2 Formulações para o 1-PAVV, 2-PAVV e o 3-PAVV

Como o leitor poderá observar, nossas formulações para os problemas k-PAVV, $k \in \{1, 2, 3\}$, são muito semelhantes àquela que introduzimos para o 1-PRVV (à menos, obviamente, das definições utilizadas para obter seus respectivos conjuntos A^2).

Utilizando as mesmas variáveis associadas a nossas duas formulações anteriores, considere uma região poliédrica, $\mathcal{R}_7$, definida pela interseção das restrições (2.3), (2.4) e (2.22) com

$$\sum_{\{(i,j)\in A:\ i\in S,\ j\in V\setminus S\}} x_{ij} + \sum_{\{(i,j)\in A^2:\ i\in S,\ j\in V\setminus S\}} y_{ij} \geq k,$$
$$\forall \{p,q\} \in V,\ p \neq q,\ \forall S \subset V,\ p \in S,\ q \in V \setminus S. \quad (2.41)$$

Uma formulação para o k-PAVV é então dada por

$$v(\mathcal{R}_7) = \text{minimize } \{ \sum_{(i,j)\in A^2} c_{ij} y_{ij} :\ (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_7 \cap \mathbf{Z}^{|A|+|A^2|} \} \quad (2.42)$$

e sua relaxação linear é definida como

$$\overline{v}(\mathcal{R}_7) = \text{minimize } \{ \sum_{(i,j)\in A^2} c_{ij} y_{ij} :\ (\mathbf{x}, \mathbf{y}) \in \mathcal{R}_7 \}, \quad (2.43)$$

o que leva a um *gap* de dualidade

$$g(\mathcal{R}_7) = \frac{v(\mathcal{R}_7) - \overline{v}(\mathcal{R}_7)}{v(\mathcal{R}_7)}. \quad (2.44)$$

Para inicializar o procedimento de solução, utilizamos as seguintes desigualdades de corte:

$$\sum_{(i,j)\in A} x_{ij} + \sum_{(i,j)\in A^2} y_{ij} \geq k,\ \forall i \in V \quad (2.45)$$

e

$$\sum_{(j,i)\in A} x_{ji} + \sum_{(j,i)\in A^2} y_{ji} \geq k,\ \forall i \in V. \quad (2.46)$$

Estas impõem que ao menos k arcos devem apontar para fora de (respectivamente, apontar para) cada vértice $i \in V$.

Capítulo 3

Resolução Exata das Formulações

O capítulo se inicia com uma caracterização conceitual dos algoritmos que desenvolvemos para resolver, de forma exata, as formulações propostas nesta dissertação. Logo a seguir, essa caracterização é tornada mais específica através da descrição de um algoritmo genérico do tipo Branch-and-Bound (BB) e sua posterior especialização em diferentes tipos de algoritmos Branch-and-Cut (BC) (vide [35] para mais informações sobre algoritmos BB e suas variantes BC). Algoritmos BC, deve-se ressaltar, constituem-se no procedimento mais efetivo atualmente existente para a resolução exata de problemas de Otimização Inteira NP-Difíceis.

Numa etapa seguinte, detalhamos de forma ainda mais específica os diferentes tipos de algoritmos BC que implementamos. Em cada caso, algoritmos de planos de corte (vide [36] para mais informações sobre esse tipo de algoritmo) são utilizados para resolver os diferentes tipos de problemas lineares contínuos encontrados em nossas enumerações BC. Sem exceção, todos eles se baseiam na separação de desigualdades de corte em digrafos (vide o preâmbulo do Capítulo 2 para uma definição dessas desigualdades e o restante do mesmo para sua especialização à cada problema considerado).

Os diferentes tipos de cortes em digrafos que consideramos podem ser entendidos como especializações daqueles associados à nossa formulação do 1-PRVV (vide formulação 2.5, Seção 2.1). Dessa maneira, o algoritmo de planos de corte que introduzimos na Subseção 2.1.1 (para resolver a relaxação linear 2.6, da formulação 2.5 do 1-PRVV) nos serve como modelo para todos os demais (cada um deles integrados, nesse caso, a diferentes algoritmos BC). No entanto, o desempenho computacional daquele algoritmo se mostrou bastante limitado (vide os resultados computacionais apresentados na Subseção 2.1.1) e procedimentos adicionais serão aqui descritos para melhorar seu desempenho. Tais benefícios são também estendidos aos algoritmos de planos de corte propostos para os demais problemas.

Concluímos o capítulo identificando de forma precisa os diferentes algoritmos BC que especificamos e a forma como o Gurobi eventualmente procedeu à execução dos

mesmos. Como vamos discutir, o Gurobi (assim como outros *softwares* comerciais de Otimização Inteira disponíveis) não necessariamente segue o que lhe foi pedido fazer. Em função disso, nos vimos compelidos a procurar alternativas (dentro das opções oferecidas pelo Gurobi) para implementar algoritmos mais próximos do que desejávamos. Como consequência, tais algoritmos acabam por incorrer num custo computacional maior que o necessário, já que o Gurobi não permite uma implementação otimizada de algoritmos que fogem ao que considera padrão.

3.1 Imposições algorítmicas de nossas formulações

Assumindo uma visão mais restritiva, todos os algoritmos de enumeração implícita que vamos aqui descrever poderiam ser classificados como sendo do tipo BB, em seu formato mais tradicional. Isso porque se baseiam diretamente na resolução das relaxações lineares das formulações propostas. Mais especificamente, não envolvem nenhum reforço dessas relaxações, em qualquer nó da árvore de enumeração BB (através, por exemplo, da agregação de eventuais desigualdades válidas, adicionais àquelas encontradas nas formulações propostas). Esse reforço de relaxações lineares, vale notar, é um dos fatores que identifica as variantes BC de um algoritmo BB (vide [35] para mais detalhes sobre algoritmos BB e suas variantes BC).

De fato, como os nossos experimentos computacionais vão demonstrar, não se faz necessário reforçar as formulações aqui propostas, pelo menos para as instâncias de teste que consideramos. Para estas, como vamos observar, as relaxações lineares de nossas formulações são quase invariavelmente inteiras e, consequentemente, ótimas para os problemas inteiros a que estão associadas. Ou seja, reforçá-las com desigualdades válidas adicionais não seria, pelo menos no momento, uma questão relevante.

Entretanto, um fator que dificulta atribuir simplesmente a denominação BB aos algoritmos de enumeração implícita que vamos descrever é o fato de todas as nossas formulações apresentarem um número exponencialmente alto de desigualdades. Mais especificamente, desigualdades de corte em digrafos, como as desigualdades (2.10), encontradas em nossa formulação do 1-PRSS. Ou seja, para resolver os problemas lineares contínuos encontrados em qualquer nó de nossas árvores de enumeração BB, tais desigualdades devem ser tratadas implicitamente, através de algoritmos de planos de corte. Torna-se então muito mais natural classificar nossos algoritmos simplesmente como algoritmos do tipo BC, já que todas as questões fundamentais associadas ao desenho, implementação e desempenho de um algoritmo BC (ou seja, a separação de desigualdades violadas, adicionais ou não à formulação original, e a forma como isso é conduzido) fazem parte de nossos algoritmos BB.

Vale aqui destacar o caso específico do 1-PRVV, problema que pode ser resol-

vido em tempo polinomial, o que de fato ocorreu nos experimentos computacionais descritos na Subseção 2.1.1. Ou seja, em tese, o 1-PRVV não demanda enumeração BB para a sua resolução (embora nossa formulação do problema não garanta isso) e poderia ser resolvido simplesmente através da relaxação linear de uma formulação adequada (que aparentemente ainda não está disponível na literatura).

Levando em conta os fatores que acabamos de mencionar e visando facilitar a descrição de nossos algoritmos de enumeração implícita, vamos assumir inicialmente que, de alguma maneira, temos acesso direto às soluções dos problemas lineares contínuos que precisamos resolver. Ou seja, num primeiro momento, não vamos entrar em detalhes maiores sobre os algoritmos de planos de corte que nos dão acesso a essas soluções. Assim procedendo, torna-se fácil descrever a estrutura geral de nossos algoritmos que, nesse contexto, poderiam ser entendidos simplesmente como algoritmos BB tradicionais. A seguir, passando a considerar explicitamente os algoritmos de planos de corte, ressaltamos suas caracteristicas BC.

3.2 Algoritmos Branch-and-Bound

Vamos utilizar o 1-PRSS como um modelo inicial para descrever (num nível mais geral) nossos algoritmos exatos, entendendo-os agora simplesmente como algoritmos do tipo BB.

No caso específico do 1-PRSS, um algoritmo BB padrão, resolveria, em primeiro lugar, no nó 0 de sua árvore de enumeração implícita, a relaxação linear (2.12) associada à formulação (2.11). Caso a solução ótima de (2.12) seja inteira, ela é naturalmente ótima para (2.11) e o algoritmo teria então obtido um *certificado de otimalidade* global para a mesma. Consequentemente, poderia ser então interrompido, logo neste estágio inicial. Caso contrário, uma variável assumindo um valor fracionário seria então escolhida para abrir dois novos nós (além do nó 0, ou raiz, em que nos encontramos) na árvore de enumeração BB. Isso é feito por meio de um procedimento denominado *ramificação*. Assuma, por exemplo, que temos duas alternativas de escolha: uma variável x_a, $a \in A$ (assumindo um valor, digamos, $\overline{x}_a = 0,5$) e uma variável y_b, $b \in A^2$ (também assumindo um valor igual a 0,5, ou seja, $\overline{y}_b = 0,5$).

Assuma que a escolha acima recai sobre a variável x_a. As duas ramificações (subproblemas) a abrir seriam então definidos em relação à variável x_a: uma delas acrescentaria uma restrição $x_a = 0$ à (2.11) (ou seja, forçaria a interseção de $\mathcal{R}_1$ com aquela restrição) e a outra faria o mesmo em relação à restrição $x_a = 1$. Consequentemente, ao se impor essas duas ramificações, o espaço de soluções definido por $\mathcal{R}_1$ (no nó 0 da árvore de enumeração BB) seria particionado em duas regiões poliédricas onde, em cada uma delas, a variável x_a assume um valor inteiro viável

distinto, nesse caso, respectivamente 0 e 1.

Continuando o procedimento de enumeração BB iniciado acima, rotulamos como *nó 1* aquele que foi *aberto* pela agregação da restrição $x_a = 0$ (à nossa descrição de $\mathcal{R}_1$) e por *nó 2* aquele em que isso é feito em relação à restrição $x_a = 1$. Nos concentramos agora na resolução dos problemas lineares contínuos associados aos nós 1 e 2, os dois únicos nós ainda em *aberto* em nossa árvore BB (o nó 0 seria então considerado *já explorado*).

Optando (através da utilização de algum critério) por resolver primeiro o problema linear contínuo associado ao nó 1, assuma que sua solução ótima é naturalmente inteira. Nesse caso, tal solução seria viável para o 1-PRSS (e deveria ser guardada, pois pode eventualmente se mostrar ótima para o problema como um todo) e o processo de enumeração implícita não precisaria ser expandido na *direção* do nó 1 (já que nenhuma solução viável para o 1-PRSS, de melhor qualidade, seria assim obtida). Caso contrário, se a solução não for inteira, uma variável assumindo um valor fracionário (obviamente uma variável distinta de x_a) seria então escolhida e dois novos nós seriam consequentemente abertos na árvore BB.

O procedimento descrito acima, de abertura e exploração de novos nós na árvore BB, seria conduzido até que não mais existissem nós abertos a explorar. Consequentemente, a melhor solução viável inteira disponível (nesse caso, a de menor valor) ao longo do procedimento, seria ótima para o 1-PRSS.

Para efeito de simplificação, omitimos diversos detalhes importantes na descrição do algoritmo BB genérico apresentado acima. Em particular, não discutimos mais detalhadamente as *podas* (ou seja, eventuais interrupções do processo de enumeração em nós da árvore BB) passíveis de se implementar.

Além de podas por viabilidade inteira, como a discutida anteriormente para o nó 1 da árvore BB que apresentamos, podas podem ser feitas, dentre outras, por *valor* ou pela eventual inviabilidade do problema linear contínuo que se quer resolver. Na primeira alternativa, a poda é efetuada sempre que o algoritmo de planos de corte retorna um valor (para o problema linear que se está resolvendo) maior ou igual ao valor de uma solução viável disponível para o 1-PRSS.

Finalmente, omitimos também qualquer menção a testes de pré-processamento, muitas vezes fundamentais para acelerar a convergência de algoritmos BB. Para maiores informações sobre essas e outras questões relevantes, sugerimos, mais uma vez, uma consulta a [35].

3.3 Algoritmos Branch-and-Cut

Concluída a descrição BB genérica de nossos algoritmos de solução exata, passamos agora a descrevê-los sob um enfoque mais pontual. Uma visão que, como se po-

derá observar, ressalta seus aspectos BC. Em particular, vamos discutir formas de conduzir a separação das desigualdades de corte (2.10), tanto na resolução da relaxação linear (2.12) da formulação (2.11), quanto na resolução dos problemas lineares contínuos que dela se originam, ao longo da árvore BB.

Vale notar que o principal objetivo de um algoritmo BB é concluir seu procedimento de enumeração implícita o mais rápido possível. No caso específico de formulações que envolvem um número exponencialmente alto de desigualdades, como é o nosso caso, isto traz não apenas desafios mas também oportunidades.

Na descrição do algoritmo BB genérico da seção anterior, assumimos que as relaxações lineares de todos os subproblemas definidos ao longo da árvore BB eram sempre resolvidas de maneira ótima. Vamos agora discutir como isso é feito de fato e suas implicações algorítmicas.

No Capítulo 2 (em nossa investigação empírica da formulação (2.5), proposta para o 1-PRVV), resolvemos (2.5) através da resolução direta de sua relaxação linear, (2.6). Isso é feito pela aplicação direta de um algoritmo de planos de corte (que *separa* as desigualdades de corte (2.2)). O procedimento é inicializado com uma relaxação linear de (2.6) que inclui apenas as desigualdades de corte (2.8) e (2.9). Esta é então resolvida e desigualdades de corte adicionais (violadas pela solução encontrada) são então identificadas e acrescidas a ela, reforçando-a. A seguir, a relaxação linear (reforçada) assim obtida é então resolvida e o procedimento como um todo é conduzido, recursivamente, até que não mais se encontre desigualdades de corte violadas. Nesse estágio, o algoritmo de planos de corte é então interrompido, com uma solução ótima para (2.6). No caso específico da formulação (2.5), para todas as instâncias de teste que consideramos, a solução ótima de (2.6) foi sempre inteira e, consequentemente, ótima para (2.5) (embora essa garantia não nos seja oferecida de antemão pela formulação (2.5)). Vamos agora ajustar, para o 1-PRSS, o procedimento que acabamos de descrever para o 1-PRVV.

Em um contexto BB, um algoritmo de planos de corte como o que acabamos de descrever oferece uma grande flexibilidade em termos de utilização. Em particular, note que, ao contrário do que fizemos para o 1-PRVV, o processo de separação de desigualdades de corte não precisa ser conduzido à exaustão. Ele pode ser interrompido mais cedo, sem comprometer a otimalidade global do algoritmo BB. Nesse caso, em tese, estaríamos simplesmente trocando um número maior de *rodadas* de separação de desigualdades de corte por um número maior de nós na árvore de enumeração BB. Obviamente, o ponto de equilíbrio na utilização ideal desses dois recursos é difícil de estabelecer e é dependente tanto da formulação considerada quanto do tipo de instância que se está resolvendo.

Na implementação de um algoritmo BC, devemos considerar atentamente a complexidade computacional dos problemas de separação envolvidos. No caso específico

da separação de desigualdades de corte (entre dois vértices predefinidos), essa complexidade é igual a $O(|V|^2)$ (que é a complexidade associada ao problema de fluxo máximo entre dois vértices de uma rede contendo $|V|$ vértices). Dependendo do número total de pares de vértices (para os quais desigualdades de corte estão definidas), o tempo de CPU gasto nessa operação pode ser excessivo (como é o caso do 1-PRVV). Nesse caso, isso poderia ser um argumento em favor de começar a *enumerar mais cedo*, em contrapartida a *cortar até o limite*, tanto em relação à cada rodada de separação de cortes quanto em relação a interromper prematuramente o algoritmo de planos de corte, como um todo. Claramente um enorme leque de alternativas existe para o uso combinado desses dois recursos.

Um ingrediente adicional na discussão acima é a complexidade de casos particulares do problema de separação. Por exemplo, o problema de separação de desigualdades de corte tem sua complexidade reduzida quando a solução do problema linear contínuo considerado é inteira. Um tipo de algoritmo BC, em que desigualdades que fazem parte da formulação do problema são tratadas implicitamente como planos de cortes e separadas apenas para pontos inteiros, é denominado algoritmo de *Lazy Constraints* (LC). Definições distintas sobre esse assunto podem ser encontradas na literatura, como a dos autores TÓTH *et al.* [37] e LERCH e TRAUTMANN [38]. Eles o entendem como um procedimento em que determinadas desigualdades necessárias à formulação de um problema são levadas em consideração apenas no instante no qual é percebido que estão sendo violadas. Quando isso ocorre, elas são adicionadas à formulação para que esta seja resolvida novamente.

De acordo com a nossa definição de um algoritmo LC, sempre que a solução do problema linear contínuo que se tem em mãos é não inteira, o algoritmo executa um procedimento de ramificação. Tipicamente, um algoritmo LC executa poucas rodadas de separação de desigualdades de cortes (já que isso só é feito apenas para soluções inteiras). Além disso, quando faz a separação, essa operação é conduzida de forma eficiente (já que a separação de pontos inteiros tem menor complexidade que o caso geral). Por outro lado, um algoritmo LC tende a explorar um número elevado de nós de enumeração, já que tipicamente efetua muitas ramificações.

Note que, até o momento, descrevemos dois tipos de algoritmos BC. Um algoritmo BC genérico, que resolve problemas de separação para qualquer tipo de entrada, inteira ou não, de forma exaustiva ou não. O outro, do tipo LC, é uma especialização do anterior que só efetua separação quando os dados de entrada são inteiros. Temos também, dentre inúmeras outras alternativas, a possibilidade de trabalharmos com uma hibridização dos dois procedimentos indicados acima. Um exemplo de algoritmo desse tipo seria um que separa desigualdades de corte de formas distintas, em diferentes nós da árvore BC. Na alternativa que vamos aqui investigar, no nó 0 das árvores BC, nosso algoritmo híbrido separa exaustivamente

as desigualdades de corte (consequentemente, separa soluções inteiras ou não). Para os demais nós, a separação é feita apenas quando a solução é inteira (se não é, ramificações são abertas, como num algoritmo LC). Denominamos *BC-LC* a esse algoritmo híbrido.

Finalmente, levando em conta a baixa complexidade computacional do problema de separação de desigualdades de corte para dados de entrada inteiros, investigamos também um refinamento do algoritmo BC que tenta se beneficiar desse fato. Mais especificamente, ao resolvermos um problema linear contínuo, verificamos se uma solução obtida é *quase inteira*. Para isso, definimos um valor ϵ suficientemente pequeno e, caso todas as variáveis da formulação assumam valores menores ou iguais a ϵ ou maiores ou igual a $(1 - \epsilon)$, arredondamos os valores fracionários para 0, no primeiro caso, e para 1, no segundo. A seguir, resolvemos o problema de separação (de desigualdades de corte) para o ponto inteiro assim obtido. Caso uma desigualdade violada seja identificada, verificamos se esta permanece violada para a solução (não inteira) original e, caso isso ocorra, reforçamos nossa relaxação linear com a mesma. Tal algoritmo, deve-se notar, é um algoritmo BC que simplesmente altera a estratégia de separação de desigualdades válidas (em seu algoritmo de planos de corte), na expectativa de acelerar a convergência do algoritmo como um todo. Nos nossos experimentos, fixamos o valor de ϵ em 10^{-4}, o que pode ser entendido como uma possível correção de um erro numérico de aproximação, evitando desperdiçar tempo de CPU separando como fracionário um ponto que é de fato inteiro.

Além da utilização de estratégias para acelerar a convergência de algoritmos de planos corte, como a que acabamos de sugerir acima, algoritmos BC e BC-LC devem incluir procedimentos para minorar os impactos negativos de um fenômeno de origem numérica denominado *tailing off* [24]. Este também impacta na convergência empírica de algoritmos de planos de corte e, para mitigar seus efeitos, um procedimento padrão é descrito na literatura e utilizado por nós (vide subseção 3.6). No entanto, sua utilização plena é limitada por restrições impostas à *customização* dos algoritmos disponíveis nos softwares comerciais de Otimização Inteira (vide subseção 3.6).

3.4 Algoritmos de planos de corte

Algoritmos de planos de corte, baseados na separação de desigualdades de corte em digrafos (como aquele descrito para o 1-PRVV no Capítulo 2), são a principal componente de todos os algoritmos BC implementados nesta pesquisa. Vamos então investigá-los mais a fundo nesta seção.

Em particular, vamos discutir primeiro algoritmos que separam desigualdades de corte para pontos genéricos, inteiros ou não (como aquele descrito no Capítulo

2). Feito isso, passamos para aqueles, mais eficientes, que envolvem algoritmos distintos para a separação de pontos inteiros e não inteiros, respectivamente. Nos dois casos vamos (mais uma vez) utilizar o 1-PRVV como modelo. Deve-se notar que, dependendo do problema, as desigualdades de corte a separar podem variar ligeiramente. No entanto, sua estrutura básica se mantém essencialmente a mesma e o fator mais revelante envolvido nas operações de separação é, quase sempre, o número de pares de vértices para os quais estas se aplicam.

3.4.1 Separação de soluções não inteiras

A separação é feita de forma semelhante àquela descrita no Capítulo 2 para a separação de desigualdades de corte (2.2) (na resolução do problema de relaxação linear (2.6), associado à nossa formulação (2.5) do 1-PRVV). Note que, embora não tenha sido necessário utilizar enumeração BB para resolver qualquer uma das instâncias de teste que utilizamos para o problema, isso poderia ter sido feito sem prejuízos à identificação de uma solução ótima inteira para o problema. Na realidade, poderia até mesmo vir a reduzir o tempo de CPU gasto para obtê-la (tendo em vista a convergência lenta apresentada pelo algoritmo de planos de cortes). No que se segue vamos assumir que estamos implementando um algoritmo BC para resolver o 1-PRVV.

Seja $D = (V, A)$ o digrafo sobre o qual o 1-PRVV é definido e $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ uma solução ótima para o problema linear contínuo que acabamos de resolver (em um nó qualquer da árvore BC). A separação das desigualdades de corte (2.2) (em relação ao ponto $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, do espaço Euclidiano) é então efetuada definindo-se, em primeiro lugar, um *grafo suporte*, $\overline{D} = (V, \overline{A})$, onde $\overline{A} = \{\{(i,j) : \overline{x}_{ij} > 0, (i,j) \in A\} \cup \{(j,i) \notin A : \overline{y}_{ij} > 0\}\}$. Mais especificamente, para cada arco $(i,j) \in \overline{A}$ associamos uma capacidade h_{ij} igual à $\overline{x}_{ij}$, quando $(i,j) \in A$, e igual à $\overline{y}_{ij}$, quando $(i,j) \notin A$.

Para cada par de vértices $s, t \in V$ que se aplique, o que se deseja é identificar um corte de mínima capacidade em $\overline{D} = (V, \overline{A})$, separando s de t. Para o 1-PRVV, cortes s-t de mínima capacidade com valores inferiores a 1 identificam uma desigualdade de corte violada pela solução $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$. Em outros problemas, como o 2-PAVV, em vez de 1, a desigualdade de corte é violada quando o corte s-t de mínima capacidade tem valor menor que 2, sendo este valor o mínimo grau de conectividade s-t imposto ao problema. Note, de forma equivalente, que encontrar um corte s-t de mínima capacidade em D corresponde a identificar o fluxo máximo de s para t (na rede capacitada associada ao digrafo $\overline{D} = (V, \overline{A})$).

Como visto na Seção 2.1.1 (em nossa investigação empírica de um algoritmo de planos de corte para resolver a relaxação linear (2.6), da formulação (2.5) do 1-PRVV), a identificação exaustiva de todas as desigualdades de corte violadas, em

cada rodada do algoritmo de separação, é por demais custosa em termos de CPU. Experimentamos então, em cada uma dessas rodadas, identificar apenas um máximo de 10 desigualdades de corte violadas. Uma vez identificadas, tais desigualdades são agregadas ao problema linear contínuo que acabamos de resolver, reforçando-o. Feito isso, o problema linear contínuo reforçado é resolvido e uma nova rodada de separação de desigualdades de corte é conduzida sobre o grafo suporte definido por sua solução ótima, como descrito acima. Note que esse procedimento de separação continua sendo exaustivo pois, eventualmente, todos os pares de vértices aplicáveis deverão ser investigados, na rodada final de separação, quando nenhuma desigualdade de corte violada é encontrada.

3.4.2 Separação de soluções inteiras

Assumindo que o ponto a separar, $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, é inteiro, vamos descrever agora três algoritmos para efetuar essa operação. Ao fazer isso vamos também assumir que $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ corresponde à solução ótima de um problema linear contínuo que pode estar associado tanto ao 1-PRVV quanto a problemas contidos em cada um dos três grupos de problemas que vamos definir a seguir.

O primeiro grupo envolve problemas que demandam conectividade forte entre todos os pares de vértices de seus digrafos e é constituído pelos seguintes problemas: k-PAVV, para $k \in \{1, 2, 3\}$, e o 1-PRVV. Para eles, o procedimento de separação que vamos descrever envolve duas etapas, no máximo.

O processo é inicializado com o digrafo suporte $\overline{D} = (V, \overline{A})$, associado ao ponto $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ que se quer separar. A seguir, identificamos todas as componentes fortemente conexas de $\overline{D}$, o que pode ser feito em tempo polinomial por meio do algoritmo de Kosaraju (consulte AHO *et al.* [39]). Para cada componente $C = (V_C, A_C)$, $C \in \mathcal{C}$, obtida, as seguintes desigualdades de corte são então violadas por $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$:

$$\sum_{\{(i,j)\,:\, i \in V_C,\, j \in V \setminus V_C,\, (i,j) \in A\}} x_{ij} + \sum_{\{(i,j)\,:\, i \in V_C,\, j \in V \setminus V_C,\, (i,j) \in A^2\}} y_{ij} \geq k \tag{3.1}$$

$$\sum_{\{(j,i)\,:\, i \in V_C,\, j \in V \setminus V_C,\, (j,i) \in A\}} x_{ji} + \sum_{\{(j,i)\,:\, i \in V_C,\, j \in V \setminus V_C,\, (j,i) \in A^2\}} y_{ji} \geq k. \tag{3.2}$$

Uma vez identificado o conjunto de desigualdades violadas indicado acima, a etapa seguinte do procedimento de separação para os problemas k-PAVV, $k \in \{1, 2, 3\}$, se aplica apenas quando $k > 1$. Nesse caso, para cada componente $C = (V_C, A_C)$, $C \in \mathcal{C}$, é necessário garantir a existência de pelo menos k caminhos elementares orientados arco-disjuntos s-t, para todo par de vértices $s, t \in V_C$, $s \neq t$. Para o caso $k = 1$, note que esse procedimento suplementar é desnecessário, já que a garantia de 1-arco-conectividade forte resulta da definição de $C \in \mathcal{C}$.

Assumindo então que $k > 1$ se aplica, considere qualquer par de vértices $s, t \in V_C$, $s \neq t$, de qualquer componente fortemente conexa $C \in \mathcal{C}$, e identifique um corte s-t de capacidade mínima em $C = (V_C, A_C)$. Denote esse corte por $\delta(S)^+$, $s \in S$, $S \subset V_C$. Se $|\delta(S)^+| < k$ resulta, a desigualdade de corte

$$\sum_{\{(i,j)\,:\,i \in S,\, j \in V \setminus S,\, (i,j) \in A\}} x_{ij} + \sum_{\{(i,j) \in\,:\,i \in S,\, j \in V \setminus S,\, (i,j) \in A^2\}} y_{ij} \geq k \tag{3.3}$$

é violada por $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ e pode ser utilizada para reforçar o problema linear contínuo que se tem em mãos. Com a identificação de desigualdades do tipo (3.3), concluímos o procedimento de separação (de desigualdades de corte) para os problemas k-PAVV, $k \in \{1, 2, 3\}$, quando $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ é assumido ser inteiro.

Para efeito de separação de desigualdades de corte para pontos $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ inteiros, o segundo grupo de problemas que vamos considerar é constituído por problemas em que essas desigualdades são definidas apenas para pares de vértices contidos em subconjuntos estritos de V, denotados por $S \subset V$. Mais especificamente, esse grupo é formado por: 1-PASS, 1-PRSS e 1-PCSS.

Para os problemas indicados acima, nosso procedimento de separação também se divide em duas etapas, a exemplo do que ocorreu para os problemas do grupo anterior. Em particular, sua primeira etapa também se inicia com a identificação das componentes fortemente conexas contidas no digrafo suporte $\overline{D} = (V, \overline{A})$, associado à $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, e com a separação de todas as desigualdades de cortes que delas resultem. No entanto, componentes de interesse são agora restritas àquelas que contêm pelo menos um vértice de S, desconsiderando-se as demais. Vale ressaltar que existindo uma componente fortemente conexa, $C = (V_C, A_C)$, que contenha todos os vértices de S, o procedimento de separação se encerra logo nesta primeira etapa.

A segunda etapa concentra-se nos subgrafos de $\overline{D} = (V, \overline{A})$ que contêm algum vértice de S, sem definirem, entretanto, componentes fortemente conexas daquele grafo. Ou seja, está voltada para subgrafos que definem componentes fracamente conexas de $\overline{D}$. Seja $\mathcal{F}$ o conjunto formado por tais componentes e denote genericamente por $F = (V_F, A_F)$, $F \in \mathcal{F}$, qualquer uma delas. Para algum $v \in V_F \cap S$, seja $F_1 = (V_{F_1}, A_{F_1})$ o subgrafo contido em F formado pela união de todos os caminhos elementares orientados que terminam em v, e $F_2 = (V_{F_2}, A_{F_2})$ o subgrafo contido em F formado por todos os caminhos elementares orientados que se iniciam em v. As desigualdades de corte definidas abaixo são violadas por $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ e podem ser utilizadas para reforçar o problema linear contínuo que deu origem a esta solução:

$$\sum_{\{(j,i)\,:\,i \in V_{F_1},\, j \in V \setminus V_{F_1},\, (j,i) \in A\}} x_{ji} + \sum_{\{(j,i)\,:\,i \in V_{F_1},\, j \in V \setminus V_{F_1},\, (j,i) \in A^2\}} y_{ji} \geq 1 \tag{3.4}$$

$$\sum_{\{(i,j)\,:\,i\in V_{F_2},\,j\in V\setminus V_{F_2},\,(i,j)\in A\}} x_{ij} + \sum_{\{(i,j)\,:\,i\in V_{F_2},\,j\in V\setminus V_{F_2},\,(i,j)\in A^2\}} y_{ij} \geq 1. \tag{3.5}$$

Com as desigualdades (3.4) e (3.5), concluímos o procedimento de separação de desigualdades de corte para o 1-PRSS, o 1-PASS e o 1-PCSS, quando o ponto a separar, $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, é inteiro. No entanto, vale lembrar que o 1-PRSS usa a orientação inversa de um arco $(i, j) \in A^2$, a saber, (j, i).

Finalmente, passamos a considerar agora nosso terceiro grupo de problemas, composto por 1-PRsT, 1-PAsT e 1-PCsT. Os procedimentos de separação de pontos inteiros, $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, que implementamos para (desigualdades de corte associadas a) suas respectivas formulações, são muito simples e diretos. Mais uma vez, se iniciam com um grafo suporte, $\overline{D} = (V, \overline{A})$, associado à $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$.

Verificamos inicialmente se $\overline{D} = (V, \overline{A})$ contém caminhos que levam de s a todos os vértices de T. Se isso ocorre, nenhuma desigualdade de corte é violada por $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ e o procedimento de separação é concluído. Caso contrário, identifique todas as componentes fracamente conexas existentes em $\overline{D}$ que possuem algum vértice de T, denotando-as por $\mathcal{F}$, e seja $F = (V_F, A_F)$, $V_F \cap T \neq \emptyset$, $F \in \mathcal{F}$, qualquer uma delas. As desigualdades de corte definidas abaixo são violadas por $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ e podem ser utilizadas para reforçar o problema linear contínuo que deu origem a tal solução:

$$\sum_{\{(j,i)\,:\,i\in V_F,\,j\in V\setminus V_F,\,(j,i)\in A\}} x_{ji} + \sum_{\{(j,i)\,:\,i\in V_F,\,j\in V\setminus V_F,\,(j,i)\in A^2\}} y_{ji} \geq 1 \tag{3.6}$$

3.5 Experimentos adicionais para o 1-PRVV

Para demonstrar empiricamente as vantagens de nossas propostas de aceleração de convergência de algoritmos de planos de corte, vamos, mais uma vez, utilizar o 1-PRVV como modelo para testes.

No Capítulo 2, utilizamos um algoritmo de fluxo máximo para separar as desigualdades de corte (2.2) associadas à nossa formulação (2.5) do 1-PRVV. Esse procedimento, como ficou evidente pelos resultados computacionais obtidos, não se mostrou atraente, demandando tempos de CPU excessivos.

Em função desse fraco desempenho e levando em conta o baixo custo computacional associado à separação de desigualdades de corte para pontos inteiros, incluímos no procedimento anterior a opção de também separar pontos quase inteiros, quando disponíveis (veja as definições pertinentes, no início deste capítulo).

Dada uma solução ótima, $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, quase inteira, arrendondamos para 0 ou 1 seus valores fracionários e separamos, de forma eficiente a partir de busca em profundidade SZWARCFITER [40] e outras instruções contidas na primeira separação de

soluções inteiras descrita na seção 3.4.2, o ponto inteiro assim obtido. Para a desigualdade de corte identificada pelo algoritmo, verificamos se a mesma é violada por $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$. Caso isso ocorra, ela é então utilizada para reforçar o problema linear contínuo que acabamos de resolver. Por sua vez, se $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ não é quase inteira, sua separação é efetuada através de um algoritmo de fluxo máximo, como descrito anteriormente.

No caso específico do 1-PRVV, os problemas lineares contínuos que resolvemos são oriundos de uma relaxação linear de (2.6) que, por sua vez, é uma relaxação linear de (2.5). Mais especificamente, nosso problema linear inicial é definido por uma região poliédrica formada pela interseção de (2.1), (2.3), (2.4), (2.8) e (2.9).

Uma "amostra representativa" dos resultados computacionais obtidos para o 1-PRVV (com a aplicação do procedimento de separação de desigualdades de corte (2.2) descrito acima) é apresentada na Tabela 3.1. Para efeito de comparação, note que essa tabela envolve as mesmas instâncias consideradas na Tabela 2.1:

Nome	$\|V\|$	$\|A\|$	$v(\mathcal{R}_0)$	$\overline{v}(\mathcal{R}_0)$	t(s)
b01	50	79	8	8	0,007
b16	100	186	16	16	0,001
c04	500	735	96	96	0,223
c13	500	1751	146	146	0,227
c16	500	2500	158	158	0,006
bip42p	1200	1967	296	296	0,027
bipe2u	550	1003	234	234	0,008
cc3-10u	1000	2025	161	161	1,555
cc3-11p	1331	2995	147	147	0,035
cc3-5u	125	150	9	9	0,001
cc5-3p	243	304	69	69	0,056
cc6-3u	729	1092	170	170	0,012
cc7-3u	2187	3062	558	558	6,458
hc10u	1024	1280	279	279	1,312
hc6p	64	77	23	23	0,004
hc7u	128	157	35	35	0,003
hc8p	256	359	63	63	0,076
hc9u	512	576	75	75	0,007

Tabela 3.1: Resultados para a relaxação linear da formulação para o 1-PRVV com separação de soluções inteiras

Comparando os tempos de CPU descritos nas tabelas 2.1 e 3.1, torna-se evidente que a separação de desigualdades de corte para pontos quase inteiros acelerou de

forma impressionante a convergência do algoritmo de planos de corte.

Vale ressaltar que o desempenho superior do novo algoritmo (quando aplicado à nossas instâncias de teste) decorre, principamente, do fato das soluções encontradas serem frequentemente quase inteiras.

Outro ponto importante a destacar é o fato do 1-PRVV ser um problema com complexidade polinomial de resolução. Nesse sentido, nossa formulação para ele se mostrou extremamente forte, com sua relaxação linear, (2.6), levando a soluções ótimas naturalmente inteiras para todas as instâncias de teste utilizadas. Isso ocorreu mesmo sem que esta apresente propriedades teóricas que garantam esse resultado. Ou seja, ela é empiricamente forte e serve como uma boa plataforma para formularmos e resolvermos os demais problemas investigados nesta dissertação, todos eles NP-difíceis.

O desempenho do algoritmo de planos de corte descrito acima nos motivou a investigar procedimentos similares para os demais problemas aqui considerados, todos eles $\mathcal{NP}$-Difíceis. Note, no entanto, que a ocorrência de *gaps* de dualidade não nulos (para as relaxações lineares de suas formulações) deve, agora, se tornar mais comum, numa situação oposta àquela que prevaleceu para o 1-PRVV. Da mesma forma, tempos de CPU mais elevados devem ser também esperados na convergência de seus respectivos algoritmos de planos de corte.

Finalmente, outro ponto importante a considerar é a forma como esses algoritmos de planos de corte devem ser incorporados a um algoritmo BC, já que os problemas a resolver são agora NP-Difíceis.

3.6 Identificação de todos os nossos algoritmos BC

Softwares comerciais de otimização inteira, como o Gurobi (que utilizamos em nossos experimentos computacionais), tendem a restringir de forma considerável a implementação dos algoritmos BC que lhes são propostos pelos usuários. Dessa maneira, como iremos argumentar mais adiante, não temos nenhuma garantia de que de fato fazem o que lhes é pedido fazer. Em realidade, as evidências disponíveis apontam na direção contrária e eles apenas seguem, numa interpretação livre, as instruções recebidas. Como consequência, procuramos então encontrar formas alternativas de impor ao software o que gostaríamos que ele fizesse. O espaço de manobra disponível para isso, no entanto, é extremamente limitado e se reflete nos algoritmos BC que vamos aqui descrever.

É importante ressaltar que, num ambiente comercial, a conduta imposta pelo Gurobi (e demais softwares comerciais de Otimização Inteira) claramente faz sentido. Nesse caso, via de regra, o usuário não é um especialista em Otimização Matemática. Entretanto, num trabalho de investigação científica, tal conduta limita fortemente

o espaço disponível para avaliar (empiricamente) novas ideias de pesquisa.

Como resultado das diferentes alternativas de implementação investigadas para nossos algoritmos de planos de corte, acabamos por nos defrontar com eventuais ocorrências de *tailing off* (veja PADBERG e RINALDI [24], para detalhes). Em qualquer nó de uma árvore de enumeração BC, o problema se manifesta mediante reduções insignificantes nos *gaps* de dualidade dos problemas lineares que se está resolvendo. Mais especificamente, mesmo depois da execução de várias iterações consecutivas do procedimento de separação (com a identificação de desigualdades válidas violadas e o consequente reforço da formulação do problema), o valor das soluções obtidas se mantêm praticamente inalterado.

Tailing off desperdiça tempo de processamento. Não apenas de forma direta, com a execução de longas e custosas sequências de separação que trazem pouco benefício. Ele também aumenta desnecessariamente a dimensão dos problemas lineares contínuos a resolver, com um consequente impacto no esforço computacional para solucioná-los. Esse efeito colateral, por sua vez, se propaga para os nós descendentes da árvore BC, provocando um *efeito em cascata*, que pode comprometer fortemente o desempenho do algoritmo BC como um todo. Identificada a ocorrência de *tailing off* (em qualquer nó de uma árvore BC), deve-se interromper imediatamente a execução do algoritmo de planos de corte e passar para o procedimento de ramificação. Em termos práticos, isso é feito definindo-se um parâmetro $n_t \in \mathbb{Z}_+$, correspondente ao número máximo permitido de iterações consecutivas do algoritmo de planos de corte sem que ocorra uma melhora percentual de pelo menos $\epsilon_t\%$ entre o valor atual da solução do subproblema e aquele obtido na n_t-ésima iteração anterior. Em nossos experimentos utilizamos os valores $n_t = 20$ e $\epsilon_t = 10^{-4}$.

Deve-se ressaltar que os diferentes softwares comerciais de otimização inteira implementam suas próprias estratégias para lidar com *tailling off*. No entanto, quando obrigado a evitar as limitações de customização por eles impostas, o usuário acaba pagando um considerável custo computacional. Isso ocorre, no nosso caso específico, nas implementações totalmente customizadas que fizemos de algoritmos de planos de corte (no nó zero de alguns de nossos algoritmos BC) e na prevenção de *tailing off* nessas implementações. Tanto em um caso quanto no outro, nossas implementações são muito menos otimizadas que aquelas implementadas pelo Gurobi para seus algoritmos padronizados.

3.6.1 Algoritmos BC e suas eventuais implementações

Os algoritmos do tipo BC que especificamos e a forma como foram eventualmente implementados pelo Gurobi são descritos a seguir:

- **BC**: especificado para implementar (em seus respectivos algoritmos de planos de corte, em qualquer nó das árvores BC) uma separação exaustiva de desigualdades de corte. Separa pontos inteiros por meio de algoritmos de busca (em largura ou profundidade) e pontos não inteiros por meio de um algoritmo de fluxos máximos. No entanto, na forma como o Gurobi o implementou, a separação não é feita de forma exaustiva e, quase sempre, envolve apenas uma ou duas iterações consecutivas do algoritmo de planos de corte, em qualquer nó da árvore BC.

- **LC**: algoritmo do tipo LC, em que (em qualquer nó da árvore BC) o algoritmo de planos de cortes separa apenas soluções inteiras (quando a solução encontrada é fracionária, uma operação de ramificação é efetuada no nó BC em questão). Problemas de separação são sempre resolvidos com o uso de busca (em largura ou profundidade). O Gurobi implementou o **LC** basicamente como especificado.

- **BC-LC**: algoritmo híbrido que se comporta como **BC**, no nó zero da árvore BC, e como **LC**, nos demais.

- **BC-LC-RI**: algoritmo de duas fases. Na primeira fase tenta resolver a relaxação linear da formulação do problema. Isso é feito aplicando a regra de *tailing off* ao algoritmo de planos de corte e separando soluções, sejam as quase inteiras por meio de busca (em largura ou em profundidade), sejam as fracionárias utilizado o algoritmo de fluxo máximo. Na segunda fase implementa um algoritmo **LC** que é inicializado com o problema linear contínuo disponível ao final da primeira fase (contendo todas as desigualdades violadas separadas ao longo da aplicação do algoritmo de planos de corte).

Capítulo 4

Experimentos Computacionais

Vamos agora apresentar e analisar os resultados obtidos em experimentos computacionais que comparam os diferentes algoritmos descritos no Capítulo 3. No entanto, antes de fazê-lo, vamos descrever brevemente os procedimentos que nos levaram às instâncias de teste que utilizamos nesses experimentos.

Na Subseção 2.1.1 é descrito um procedimento de geração de instâncias para o 1-PRVV. Este mesmo procedimento será agora estendido aos demais problemas que investigamos. Como veremos, os acréscimos efetuados são quase sempre marginais.

Definido um digrafo $D = (V, A)$, o 1-PRVV demanda conectividade forte de i para j, para todo par $i, j \in V$, $i \neq j$. Entretanto, para alguns dos problemas adicionais que investigamos, demandas de conectividade são impostas apenas aos pares de vértices definidos em um conjunto próprio de V. Por exemplo, conectividade forte entre todos os pares de vértices distintos de S, onde $S \subset V$. Nesse caso, levando em conta o fato de nossas instâncias do 1-PRVV se originarem de instâncias do PSG, para as quais um subconjunto de vértices terminais $R \subset V$ foi predefinido, simplesmente tomamos $S := R$.

Em casos adicionais, em que um vértice $s \in V$ e um subconjunto $T \subset V \setminus \{s\}$ definem os pares de vértices s-t, $t \in T$, sujeitos a demandas de conectividade, escolhemos o vértice de menor índice em R para representar s e fixamos $T := R \setminus \{s\}$. Para problemas de adição de arcos, optamos por definir um conjunto de arcos A^2 (candidatos à adição), onde $D^2 = (V, A^2)$ é o grafo complemento de D.

Finalmente, para as instâncias do k-PR e do k-PC, o custo de um arco pertencente a A^2 foi fixado no mesmo valor do custo da aresta do PSG que lhe deu origem. Para o k-PA, por sua vez, esse custo é escolhido como um número inteiro aleatório definido no intervalo que vai de zero até o custo máximo de uma aresta de seu grafo PSG correspondente.

Para cada problema investigado, apresentamos resultados que se dividem em duas tabelas. Uma, relativa aos algoritmos **BC**, **LC** e **BC-LC** enquanto a outra é dedicada exclusivamente ao algoritmo **BC-LC-RI**.

Tabelas dedicadas aos algoritmos **BC**, **LC** e **BC-LC** são organizadas em onze colunas. A primeira identifica as instâncias. A segunda indica os valores das soluções ótimas, $v(\mathcal{R}_p)$, $p \in \{1, \ldots, 7\}$, para cada problema considerado. A seguir, são apresentados três blocos consecutivos de resultados, um bloco para cada algoritmo. Cada um desses blocos, por sua vez, é composto por três colunas. A primeira coluna indica o número de nós explorados pela árvore de enumeração implícita. A segunda, o valor do limite inferior obtido (pelo algoritmo) para $\overline{v}(\mathcal{R}_p)$, $p \in \{1, \ldots, 7\}$, no nó zero daquela árvore. Finalmente, a terceira indica o tempo de CPU, em segundos, gasto pelo algoritmo.

Tabelas dedicadas a algoritmos do tipo **BC-LC-RI** são organizadas com suas duas primeiras colunas contendo informações análogas às de suas congêneres para **BC**, **LC** e **BC-LC**. A seguir, aparece um bloco de três colunas. A primeira delas indicando os valores dos limites inferiores obtidos para $\overline{v}(\mathcal{R}_p)$, $p \in \{1, \ldots, 7\}$, na primeira etapa de **BC-LC-RI**. A segunda, mostrando o tempo de CPU, em segundos, requerido para obter a informação anterior. Finalmente, a terceira coluna indica o tempo de CPU total despendido pelo algoritmo para obter uma solução comprovadamente ótima para cada instância de teste.

Por último, sempre que apresentarmos resultados dos experimentos computacionais para os problemas, faremos uso de amostras *representativas* dos resultados obtidos. Essa "representatividade" está ligada às informações obtidas num dado experimento executado sobre uma instância qualquer de um problema, quais sejam o número de nós explorados na árvore de enumeração BC, o tempo computacional despendido para isso e o valor do *gap* obtido na relaxação linear inicial. Ao escolhermos os casos de teste para participarem das amostras, buscamos manter um grau considerável de diversidade entre eles no que se refere aos atributos citados anteriormente.

4.1 Experimentos computacionais para o 1-PRSS

Uma "amostra representativa" dos resultados computacionais obtidos para as instâncias do 1-PRSS é apresentada a seguir. Note que este problema demanda reversões a custo mínimo de arcos de $D = (V, A)$, de forma a obter conectividade forte entre todos os pares orientados de vértices de um conjunto $S \subset V$. As tabelas 4.1 e 4.2 apresentam os resultados obtidos.

Nome	$v(\mathcal{R}_1)$	BC			LC			BC-LC		
		#nós	$\overline{v}(\mathcal{R}_1)$	$t(v(\mathcal{R}_1))$	#nós	$\overline{v}(\mathcal{R}_1)$	$t(v(\mathcal{R}_1))$	#nós	$\overline{v}(\mathcal{R}_1)$	$t(v(\mathcal{R}_1))$
b03	67	0	67	0,01	0	67	0,01	0	67	0,01
b15	185	1	185	0,25	1	185	0,03	1	185	0,26

Nome	$v(\mathcal{R}_1)$	BC			LC			BC-LC		
		#nós	$\overline{v}(\mathcal{R}_1)$	$t(v(\mathcal{R}_1))$	#nós	$\overline{v}(\mathcal{R}_1)$	$t(v(\mathcal{R}_1))$	#nós	$\overline{v}(\mathcal{R}_1)$	$t(v(\mathcal{R}_1))$
c04	397	0	397	0,42	0	397	0,44	0	397	0,42
c11	71	487	67,5	6,01	54	59,5	0,12	5	67,5	0,12
c16	38	416	31,5	3,82	134	31,5	0,11	149	31,5	0,13
bip62p	2161	0	2161	1,5	0	2161	1,48	0	2161	1,48
cc10-2u	210	103	208,5	2453,28	19	208,5	1,86	25	208,5	68,64
cc6-3p	6713	0	6713	0,57	0	6713	0,57	0	6713	0,56
cc9-2u	156	1	156	5,69	1	156	0,76	1	156	5,74
hc7p	3125	0	3125	0,04	0	3125	0,04	0	3125	0,04

Tabela 4.1: Resultados computacionais para os algoritmos **BC**, **LC** e **BC-LC**

Instância	$v(\mathcal{R}_1)$	BC-LC-RI		
		$\overline{v}(\mathcal{R}_1)$	$t(\overline{v}(\mathcal{R}_1))$	$t(v(\mathcal{R}_1))$
b03	67	67	0,005	0,013
b15	185	185	0,048	0,062
c04	397	397	0,526	0,721
c11	71	71	0,226	0,252
c16	38	38	1,398	1,436
bip62p	2161	2161	1,63	3,564
cc10-2u	210	208	1056,382	1058,06
cc6-3p	6713	6713	5,831	6,075
cc9-2u	156	156	19,631	19,754
hc7p	3125	3125	0,024	0,048

Tabela 4.2: Resultados computacionais para o algoritmo **BC-LC-RI**

Os resultados da Tabela 4.1 indicam claramente que a forma como o Gurobi implementou o algoritmo **BC** está longe de ser satisfatória. Ela leva a um número excessivamente alto de nós nas árvores de enumeração (o que poderia ter sido evitado, caso o software permanecesse por mais tempo separando desigualdades de cortes violadas num mesmo nó). Da mesma forma, leva também a tempos excessivos de CPU (justificados em grande parte pelo alto custo de separação de desigualdades de corte através de algoritmos de fluxos máximos). Vale aqui ressaltar que efetuar mais rodadas de separação de desigualdades de corte (do que as duas ou, no máximo três, implementadas pelo Gurobi, em qualquer nó de suas árvores de enumeração implícita) nos aproximaria de um balanceamento ideal entre "cortar" e "enumerar", reduzindo simultaneamente o número de nós enumerados e, com alta probabilidade, também os tempos de CPU.

Para os algoritmos **LC** e **BC-LC**, o número de nós explorados (em cada árvore de enumeração) foi aproximadamente o mesmo. Estes, por sua vez, foram menores que aqueles correspondentes ao algoritmo **BC** (o que não ocorreria se mais iterações do algoritmo de planos de corte fossem permitidas, em cada nó das árvores de enumeração de **BC**). Tal afirmação é validada pelos resultados obtidos pelo algoritmo **BC-LC-RI**, em que permitimos ao algoritmo de planos de corte trabalhar por mais tempo (pelo menos na primeira fase de **BC-LC-RI**).

De qualquer maneira, o fato de nossa formulação do 1-PRSS ter se mostrado forte (pelo menos para as instâncias de teste que utilizamos), permitiu que todos os nossos algoritmos alcançassem sempre *pequenos gaps* de dualidade. Isso, por sua vez, tende a favorecer algoritmos do tipo de **LC**. Essa vantagem torna-se ainda maior quando as alternativas disponíveis, **BC** e **BC-LC** (este último, em seu nó 0 de enumeração) têm custos de separação elevados. Ou seja, se o ponto de partida é próximo ao ótimo (como invariavelmente ocorre para nossas instâncias do 1-PRSS), enumerar mais em detrimento de separar mais torna-se computacionalmente mais atraente. Isso, em essência, explica o desempenho superior apresentado por **LC**, para nossas instâncias de teste.

Deve-se ressaltar que, como esperado, insistir numa separação exaustiva de nós claramente levaria a limites inferiores mais fortes para $\overline{v}(\mathcal{R}_p)$, $p \in \{1, \ldots, 7\}$ (o que é confirmado pelos resultados obtidos na primeira etapa do algoritmo **BC-LC-RI**). Note que, para as instâncias selecionadas, **BC-LC-RI** deixa de resolver à otimalidade, no nó 0 de suas árvores de enumeração, apenas a instância *cc-10-2u* (ainda assim, para um *gap* de dualidade de apenas 0,001%). Em contraponto a isso, os algoritmos **BC** e **BC-LC** (que têm operações análogas no nó raíz de suas respectivas árvores de enumeração) apresentam os seguintes *gaps* de dualidade: $4,92\%$ para *c11*, $17,1\%$ para *c16* e $0,71\%$ para *cc-10-2u*. Concentrando-se agora nos resultados obtidos pelo algoritmo **LC**, note que este apresenta, como seria de se esperar, os piores *gaps* de dualidade alcançados (empates com **BC** e **BC-LC**, para *c16* e *cc-10-2u*, e o pior resultado obtido para *c11*, com um gap de dualidade de $16,19\%$).

É possível concluir que o algoritmo **BC-LC-RI** nos aponta na direção de se tentar separar mais intensamente as desigualdades de corte. Quão mais eficiente for efetuada essa separação, maior será o seu impacto. Nesse sentido, além da separação especializada para pontos quase inteiros (que introduzimos nesta dissertação), seria importante dispor tanto de heurísticas rápidas e efetivas quanto de algoritmos exatos de mais baixa complexidade computacional, voltados para a separação de pontos não inteiros. Essas duas alternativas, entretanto, são atualmente inexistentes e se constituem num importante tópico de investigação futura.

Numa nota final sobre o desempenho de **BC-LC-RI**, vamos nos focar no nú-

mero de nós de enumeração explorados em sua segunda etapa. Partindo-se de uma relaxação linear inicial mais forte, o que ocorre para **BC-LC-RI** (em sua segunda etapa), seria se esperar um impacto positivo no número total de nós explorados pelo algoritmo. Isso, deve-se observar, de fato ocorreu (quando comparado aos demais algoritmos) em nossos experimentos para o 1-PRSS.

Finalmente, para as instâncias em que $g(\mathcal{R}_1)$ resultou diferente de 0 (vide resultados detalhados para todos os algoritmos e instâncias, em `https://github.com/pl225/instancias-dissertacao`), o algoritmo **BC-LC-RI** obteve limitantes duais mais fortes que **BC**, **LC** e **BC-LC** (nas raízes de suas respectivas árvores de enumeração implícita). De qualquer forma, como ressaltado anteriormente, mesmo quando diferentes de 0, os *gaps* de dualidade encontrados para nossas instâncias do 1-PRSS foram, via de regra, extremamente baixos, o que tende a favorecer algoritmos do tipo **LC**.

4.2 Experimentos computacionais para o 1-PRsT

Uma "amostra representativa" dos resultados computacionais obtidos para o 1-PRsT é oferecida pelas tabelas 4.3 e 4.4. O problema, deve-se lembrar, permite a reversão (ao menor custo total possível) dos arcos de $D = (V, A)$, de forma a oferecer conectividade entre $s \in V$ e todo $t \in T$, $T \subseteq V \setminus \{s\}$.

Nome	$v(\mathcal{R}_2)$	BC			LC			BC-LC		
		#nós	$\overline{v}(\mathcal{R}_2)$	$t(v(\mathcal{R}_2))$	#nós	$\overline{v}(\mathcal{R}_2)$	$t(v(\mathcal{R}_2))$	#nós	$\overline{v}(\mathcal{R}_2)$	$t(v(\mathcal{R}_2))$
b03	44	0	44	0	0	44	0,01	0	44	0
b15	111	0	111	0,01	0	111	0,01	0	111	0,01
c04	199	1	198,25	0,37	1	198,25	0,38	1	198,25	0,37
c11	44	5832	31,25	69,05	157	31,25	0,09	65	31,25	0,07
c16	27	1	26,5	0,05	5	26,5	0,04	1	26,5	0,04
bip62p	1641	1	1641	0,55	1	1641	0,28	1	1641	0,56
cc10-2u	106	1	106	1,16	29	106	1,03	1	106	1,19
cc6-3p	3091	0	3091	0,04	0	3091	0,04	0	3091	0,04
cc9-2u	87	854	85,5	34,51	120	85,5	0,42	167	85,5	0,67
hc7p	1869	0	1869	0,01	0	1869	0,01	0	1869	0,01

Tabela 4.3: Resultados computacionais dos algoritmos **BC**, **LC** e **BC-LC**

Instância	$v(\mathcal{R}_2)$	BC-LC-RI		
		$\overline{v}(\mathcal{R}_2)$	$t(\overline{v}(\mathcal{R}_2))$	$t(v(\mathcal{R}_2))$
b03	44	44	0,002	0,004
b15	111	111	0,013	0,02

Instância	$v(\mathcal{R}_2)$	BC-LC-RI		
		$\overline{v}(\mathcal{R}_2)$	$t(\overline{v}(\mathcal{R}_2))$	$t(v(\mathcal{R}_2))$
c04	199	199	0,422	0,513
c11	44	44	6,581	6,932
c16	27	27	0,132	0,14
bip62p	1641	1641	70,585	70,97
cc10-2u	106	100,5	8,99	9,391
cc6-3p	3091	3091	0,072	0,099
cc9-2u	87	87	12,858	13,008
hc7p	1869	1869	0,013	0,022

Tabela 4.4: Resultados computacionais para o algoritmo **BC-LC-RI**

De uma maneira geral, as mesmas conclusões a que chegamos antes, para o 1-PRSS, também se aplicam aqui. Para o 1-PRsT, no entanto, a implementação inadequada feita pelo Gurobi para o algoritmo **BC** se sobressai de forma clara. Em particular, nos resultados obtidos para a instância *c11*. Nesse caso específico, o número de nós explorados por **BC** foi de 5832, o que se contrapõe aos 157 e 65 nós explorados respectivamente por **LC** e **BC-LC**. Se o Gurobi tivesse implementado o algoritmo na forma como o concebemos, **BC** teria certamente enumerado um menor número de nós que **BC-LC**.

Observamos, mais uma vez, a ocorrência quase invariável de *gaps* de dualidade nulos para as instâncias testadas. Uma exceção, dentre aquelas que aparecem na tabela 4.4, é a instância *cc10-2u*, com um *gap* de dualidade de 5,18%. É importante ressaltar que, embora *gaps* de dualidade nulos sejam obtidos para essa mesma instância pela implementação do Gurobi para os algoritmos **BC**, **LC** e **BC-LC**, isso ocorre não porque $g(\mathcal{R}_1)$ é de fato nulo para a mesma. Isso resulta das técnicas de pré-processamento utilizadas pelo *software*, que conseguem fortalecer nossa formulação do problema (fixando valores para algumas de suas variáveis e introduzindo desigualdades válidas adicionais) e acabam por reduzir seu *gap* de dualidade a 0. Entretanto, mesmo com a utilização plena desse módulo de pré-processamento, *gaps* de dualidade de valores $0,37\%$, $28,97\%$, $1,85\%$ e $1,72\%$ são identificados para as instâncias *c04*, *c11*, *c16* e *cc9-2u*, respectivamente. Isso deve-se, provavelmente, ao reduzido número de iterações de separação de desigualdades de corte que o software se permite fazer, em qualquer nó de sua árvore de enumeração implícita.

A título de curiosidade, as demais instâncias do 1-PRsT que apresentam *gaps* de dualidade não nulos podem ser consultadas no mesmo material suplementar já citado anteriormente.

4.3 Experimentos computacionais para o 1-PCSS

Apresentamos nas tabelas 4.5 e 4.6 uma "amostra representativa" dos resultados computacionais obtidos para o 1-PCSS. Este, deve-se lembrar, é um problema que permite (ao menor custo total possível) um complemento dos arcos de $D = (V, A)$ (por arcos pertencentes a um dado conjunto A^2) de forma a obter conectividade forte entre todos os pares de vértices orientados de um conjunto predefinido S, $S \subset V$.

Mais uma vez, as mesmas conclusões das análises de desempenho de algoritmos feitas anteriormente (para o 1-PRSS e o 1-PRsT) também se aplicam aqui. No entanto, os tempos de CPU demandados para a resolver instâncias do 1-PCSS se mostraram um pouco mais elevados que aqueles que se aplicam ao 1-PRSS e ao 1-PRsT. Isto nos leva a sugerir que o 1-PCSS se mostrou experimentalmente mais difícil de ser resolvido que aqueles.

Dentre as instâncias da tabela 4.6, *hc10u* foi a única a apresentar um *gap* de dualidade não nulo, de 1,91%. Da mesma forma, como indicado na Tabela 4.5, mesmo com a utilização dos testes de pre-processamento e outras técnicas utilizadas pelo Gurobi, os algoritmos **BC**, **LC** e **BC-LC** apresentaram *gaps* de dualidade não nulos para as seguintes instâncias: *c02*, *c09* e *cc7-3p*.

Nome	$v(\mathcal{R}_3)$	BC			LC			BC-LC		
		#nós	$\overline{v}(\mathcal{R}_3)$	$t(v(\mathcal{R}_3))$	#nós	$\overline{v}(\mathcal{R}_3)$	$t(v(\mathcal{R}_3))$	#nós	$\overline{v}(\mathcal{R}_3)$	$t(v(\mathcal{R}_3))$
b02	82	1	82	0,03	1	82	0,02	1	82	0,03
b18	86	0	86	0,05	0	86	0,05	0	86	0,05
c02	106	3675	83,5	222,94	332	83,5	0,4	310	83,5	0,45
c09	574	6	573	74,13	14	569,5	1,07	6	573	28,59
c15	550	0	550	0,93	0	550	0,93	0	550	0,9
bip42p	421	0	421	2,38	0	421	2,32	0	421	2,33
cc10-2p	21125	1	21125	20,99	1	21125	1,45	1	21125	20,91
cc3-11p	2250	0	2250	0,8	0	2250	0,84	0	2250	0,83
cc7-3p	27799	417	27634,5	53829,13	7	27634,5	5,8	7	27634,5	409,84
hc10u	262	1	262	305,3	1	262	4,82	1	262	307,6

Tabela 4.5: Resultados computacionais dos algoritmos **BC**, **LC** e **BC-LC**

Instância	$v(\mathcal{R}_3)$	BC-LC-RI		
		$\overline{v}(\mathcal{R}_3)$	$t(\overline{v}(\mathcal{R}_3))$	$t(v(\mathcal{R}_3))$
b02	82	82	0,022	0,028
b18	86	86	0,031	0,071
c02	106	106	14,624	14,804
c09	574	574	838,115	838,478

Instância	$v(\mathcal{R}_3)$	BC-LC-RI		
		$\overline{v}(\mathcal{R}_3)$	$t(\overline{v}(\mathcal{R}_3))$	$t(v(\mathcal{R}_3))$
c15	550	550	0,707	1,368
bip42p	421	421	1,049	2,145
cc10-2p	21125	21125	6204,104	6210,127
cc3-11p	2250	2250	0,37	0,827
cc7-3p	27799	27799	196,289	199,78
hc10u	262	257	1983,155	1987,88

Tabela 4.6: Resultados computacionais para o algoritmo **BC-LC-RI**

4.4 Experimentos computacionais para o 1-PCsT

Apresentamos nas tabelas 4.7 e 4.8 uma "amostra representativa" dos resultados computacionais obtidos para o 1-PCsT. Este, deve-se lembrar, é um problema que permite (ao menor custo total possível) um complemento dos arcos de $D = (V, A)$ (por meio de arcos pertencentes a um conjunto predefinido, A^2) de forma a obter conectividade entre um dado $s \in V$ e todo $t \in T$, $T \subseteq V \setminus \{s\}$.

Em linhas gerais, pode-se perceber (pelos resultados apresentados nas tabelas 4.7 e 4.8) que o desempenho de nossos algoritmos para o 1-PCsT (o segundo problema de complemento de arcos que aqui consideramos) é similar àqueles observados para os problemas anteriores. Dito isso, devemos ressaltar que o 1-PCsT se mostrou, empiricamente, consideravelmente mais fácil de resolver que o 1-PCSS.

Na tabela 4.8, destinada ao algoritmo **BC-LC-RI**, a instância *hc10u* é a única a apresentar um gap de dualidade não nulo, de 7,5%, o que é relativamente alto. Por sua vez, mesmo com o módulo de pré-processamento do Gurobi ativado, os algoritmos **BC**, **LC** e **BC-LC** obtiveram *gaps* de dualidade não nulos para as instâncias *c02* e *c09* (embora, para a instância *hc10u*, tenham todos chegado a um gap de dualidade nulo, pelas facilidades oferecidas pelo *solver*).

Nome	$v(\mathcal{R}_4)$	BC			LC			BC-LC		
		#nós	$\overline{v}(\mathcal{R}_4)$	$t(v(\mathcal{R}_4))$	#nós	$\overline{v}(\mathcal{R}_4)$	$t(v(\mathcal{R}_4))$	#nós	$\overline{v}(\mathcal{R}_4)$	$t(v(\mathcal{R}_4))$
b02	46	1	46	0,01	1	46	0,01	1	46	0,01
b18	38	0	38	0,01	0	38	0,01	0	38	0,01
c02	90	3828	63	31,89	1190	63	0,32	2277	63	0,64
c09	372	1749	362,5	108,11	6	362,5	0,36	20	362,5	0,53
c15	373	0	373	0,34	0	373	0,33	0	373	0,33
bip42p	210	0	210	0,04	0	210	0,03	0	210	0,03
cc10-2p	11007	1	11007	0,77	1	11007	0,58	1	11007	0,74
cc3-11p	313	0	313	0,05	0	313	0,04	0	313	0,05

Nome	$v(\mathcal{R}_4)$	BC			LC			BC-LC		
		#nós	$\overline{v}(\mathcal{R}_4)$	$t(v(\mathcal{R}_4))$	#nós	$\overline{v}(\mathcal{R}_4)$	$t(v(\mathcal{R}_4))$	#nós	$\overline{v}(\mathcal{R}_4)$	$t(v(\mathcal{R}_4))$
cc7-3p	15178	1	15178	1,43	1	15178	0,89	1	15178	1,43
hc10u	166	1	166	2,69	5	166	1,95	1	166	2,66

Tabela 4.7: Resultados computacionais dos algoritmos **BC**, **LC** e **BC-LC**

Instância	$v(\mathcal{R}_4)$	BC-LC-RI		
		$\overline{v}(\mathcal{R}_4)$	$t(\overline{v}(\mathcal{R}_4))$	$t(v(\mathcal{R}_4))$
b02	46	46	0,009	0,012
b18	38	38	0,003	0,004
c02	90	90	6,03	6,156
c09	372	372	21,675	21,851
c15	373	373	0,128	0,418
bip42p	210	210	0,088	0,116
cc10-2p	11007	11007	24,854	25,19
cc3-11p	313	313	0,111	0,119
cc7-3p	15178	15178	30,313	30,569
hc10u	166	153,5	34,614	36,782

Tabela 4.8: Resultados computacionais para o algoritmo **BC-LC-RI**

4.5 Experimentos computacionais para o 1-PASS

Apresentamos nas tabelas 4.9 e 4.10 uma "amostra representativa" dos resultados computacionais obtidos para o 1-PASS. Este, deve-se lembrar, é um problema que permite (ao menor custo total possível) a adição de arcos (pertencentes a um conjunto predefinido, A^2) à $D = (V, A)$, de forma a obter conectividade forte entre todos os pares orientados de vértices de um conjunto predefinido, $S \subset V$.

Embora apresente a mesma tendência identificada nos resultados anteriores, observamos aqui uma maior demanda de tempo de CPU para resolver nossas instâncias representativas do 1-PASS. Ou seja, o problema parece ser empiricamente mais difícil de resolver que os anteriores. Mesmo assim, na Tabela 4.10), nenhum *gap* de dualidade é identificado para o algoritmo **BC-LC-RI**. Ao contrário disso, *gaps* de dualidade estão associados à **BC**, **LC** e **BC-LC** (vide Tabela 4.9).

Nome	$v(\mathcal{R}_5)$	BC			LC			BC-LC		
		#nós	$\overline{v}(\mathcal{R}_5)$	$t(v(\mathcal{R}_5))$	#nós	$\overline{v}(\mathcal{R}_5)$	$t(v(\mathcal{R}_5))$	#nós	$\overline{v}(\mathcal{R}_5)$	$t(v(\mathcal{R}_5))$
b08	170	0	170	0,05	0	170	0,05	0	170	0,05
b14	167	0	167	0,11	0	167	0,1	0	167	0,11

Nome	$v(\mathcal{R}_5)$	BC			LC			BC-LC		
		#nós	$\overline{v}(\mathcal{R}_5)$	$t(v(\mathcal{R}_5))$	#nós	$\overline{v}(\mathcal{R}_5)$	$t(v(\mathcal{R}_5))$	#nós	$\overline{v}(\mathcal{R}_5)$	$t(v(\mathcal{R}_5))$
c05	257	0	257	23,15	0	257	22,97	0	257	23,37
c10	410	0	410	23,19	0	410	23,43	0	410	23,15
c19	247	5	246	246,49	5	246	22,29	5	246	96,4
bip62p	25	1	25	2362,16	1	25	328,18	1	25	2340,68
cc10-2u	141	0	141	124,35	0	141	123,3	0	141	123,07
cc3-12u	12	0	12	113,59	0	12	112,69	0	12	113,83
cc6-3p	75	3	74,5	185,91	3	74,5	57,21	3	74,5	143,03
hc9p	306	0	306	44,67	0	306	45,54	0	306	45,01

Tabela 4.9: Resultados computacionais para os algoritmos **BC**, **LC** e **BC-LC**

Instância	$v(\mathcal{R}_5)$	BC-LC-RI		
		$\overline{v}(\mathcal{R}_5)$	$t(\overline{v}(\mathcal{R}_5))$	$t(v(\mathcal{R}_5))$
b08	170	170	0,035	0,043
b14	167	167	0,051	0,194
c05	257	257	13,076	42,449
c10	410	410	10,426	47,689
c19	247	247	23,057	51,741
bip62p	25	25	1379,005	1386,249
cc10-2u	141	141	110,536	345,495
cc3-12u	12	12	41,496	43,958
cc6-3p	75	75	2464,044	2529,643
hc9p	306	306	23,743	86,824

Tabela 4.10: Resultados computacionais para o algoritmo **BC-LC-RI**

4.6 Experimentos computacionais para o 1-PAsT

Apresentamos nas tabelas 4.11 e 4.12 uma "amostra representativa" dos resultados computacionais obtidos para o 1-PAsT. Este, deve-se lembrar, é um problema que permite (ao menor custo total possível) a adição de arcos (de um conjunto predefinido) a um digrafo $D = (V, A)$, de forma a obter conectividade entre um dado $s \in V$ e todo $t \in T$, $T \subset V \setminus \{s\}$.

Novamente, as mesmas considerações gerais feitas para os problemas anteriores também se aplicam aqui. Um detalhe a ressaltar para o 1-PAsT é que, com uma única exceção, *gaps* de dualidade nulos foram alcançados por todos os nossos algoritmos, para todas as nossas instâncias representativas do 1-PAsT. A exceção se aplica à instância *cc6-3p*, em que um *gap* de dualidade de valor $1,28\%$ resultou para

todos os algoritmos.

Nome	$v(\mathcal{R}_4)$	BC			LC			BC-LC		
		#nós	$\overline{v}(\mathcal{R}_4)$	$t(v(\mathcal{R}_4))$	#nós	$\overline{v}(\mathcal{R}_4)$	$t(v(\mathcal{R}_4))$	#nós	$\overline{v}(\mathcal{R}_4)$	$t(v(\mathcal{R}_4))$
b08	67	0	67	0,01	0	67	0,01	0	67	0,01
b14	65	0	65	0,02	0	65	0,02	0	65	0,02
c05	127	0	127	1,92	0	127	1,79	0	127	1,74
c10	256	0	256	5,78	0	256	5,68	0	256	5,62
c19	123	0	123	3,08	0	123	3,05	0	123	3,06
bip62p	13	0	13	5,96	0	13	5,89	0	13	5,9
cc10-2u	86	0	86	62,62	0	86	62,88	0	86	62,18
cc3-12u	2	0	2	4,75	0	2	4,88	0	2	4,7
cc6-3p	39	3	38,5	24,63	3	38,5	23,34	3	38,5	24,29
hc9p	146	0	146	11,27	0	146	11,34	0	146	11,31

Tabela 4.11: Resultados computacionais para os algoritmos **BC**, **LC** e **BC-LC**

Instância	$v(\mathcal{R}_6)$	BC-LC-RI		
		$\overline{v}(\mathcal{R}_6)$	$t(\overline{v}(\mathcal{R}_6))$	$t(v(\mathcal{R}_6))$
b08	67	67	0,011	0,016
b14	65	65	0,011	0,034
c05	127	127	0,677	0,915
c10	256	256	4,038	6,189
c19	123	123	1,149	9,027
bip62p	13	13	11,596	13,433
cc10-2u	86	86	47,506	72,054
cc3-12u	2	2	3,233	5,27
cc6-3p	39	38,5	52,035	69,051
hc9p	146	146	4,802	20,319

Tabela 4.12: Resultados computacionais para o algoritmo **BC-LC-RI**

4.7 Experimentos computacionais para o k-PAVV

Finalmente, concluindo a apresentação e discussão dos experimentos computacionais que efetuamos, consideramos agora o k-PAVV.

Uma "amostra representativa" dos resultados obtidos para o problema é apresentada nas tabelas 4.13 e 4.14. Este, vale lembrar, permite (a um custo total mínimo) a adição de arcos a um digrafo $D = (V, A)$, de forma a obter k-conectividade forte, $k \in \{1, 2, 3\}$, entre todos os seus pares orientados de vértices.

Mais uma vez, observamos aqui o mesmo padrão dos resultados anteriores. Um ponto a destacar, no entanto, é que nenhuma das instâncias das tabelas 4.13 e 4.14 apresenta *gap* de dualidade não nulo. Por outro lado, apesar disso, o tempo de CPU demandado para resolver as instâncias do k-PAVV se mostrou relativamente alto, aumentando com o aumento do valor de k.

O uso mais intensivo de CPU para a resolução exata do k-PAVV (através de nossa formulação do problema) não deve vir como uma surpresa. Observe que, quando $k > 1$ ocorre, a separação de desigualdades de corte para pontos inteiros envolve duas fases, como indicado na seção 3.4.2. Em particular, esse procedimento exige (em sua segunda fase) a verificação da existência (para toda componente fortemente conexa identificada na fase anterior) de pelo menos k caminhos elementares arco-disjuntos entre todos os pares orientados de vértices da componente. Isso é feito através da utilização direta de algoritmos de fluxos máximos que, como é notório, têm uma complexidade computacional relativamente alta para essa função. Dessa forma, no caso específico das instâncias do grupo PUC, essa restrição prática nos levou a resolver o k-PAVV apenas para $k = 1$.

Concluindo esta seção, vale ressaltar que nossos algoritmos conseguiram resolver todas as instâncias de teste do k-PAVV (para todos os valores de k eventualmente considerados) no nó zero de suas respectivas árvores de enumeração implícita.

Nome	k	$v(\mathcal{R}_7)$	#nós	BC		LC		BC-LC	
				$\overline{v}(\mathcal{R}_7)$	$t(v(\mathcal{R}_7))$	$\overline{v}(\mathcal{R}_7)$	$t(v(\mathcal{R}_7))$	$\overline{v}(\mathcal{R}_7)$	$t(v(\mathcal{R}_7))$
b07	1	318	0	318	0,01	318	0,001	318	0,002
	2	2637	0	2637	1,91	2637	1,85	2637	1,86
	3	7219	0	7219	2,95	7219	2,94	7219	2,89
b14	1	336	0	336	0,02	336	0,02	336	0,02
	2	2875	0	2875	4,74	2875	4,65	2875	4,76
	3	7391	0	7391	7,43	7391	7,35	7391	7,5
c08	1	344	0	344	0,14	344	0,14	344	0,14
	2	2946	0	2946	999,32	2946	996,56	2946	1010,16
	3	7586	0	7586	1797,18	7586	1786,32	7586	1803,8
c12	1	606	0	606	0,15	606	0,15	606	0,14
	2	2890	0	2890	952,74	2890	950,4	2890	957,69
	3	7229	0	7229	1633,51	7229	1607,28	7229	1607,68
c17	1	852	0	852	0,19	852	0,19	852	0,19
	2	3847	0	3847	872,5	3847	867,18	3847	867,56
	3	9013	0	9013	1684,06	9013	1709,43	9013	1716,5
bip52p	1	590	0	590	5,64	590	5,53	590	5,54
cc10-2u	1	599	0	599	38,74	599	38,95	599	38,45
cc3-11p	1	265	0	265	11,75	265	11,77	265	11,74
cc6-3p	1	377	0	377	6,71	377	6,7	377	6,61

Nome	k	$v(\mathcal{R}_7)$	#nós	BC		LC		BC-LC	
				$\overline{v}(\mathcal{R}_7)$	$t(v(\mathcal{R}_7))$	$\overline{v}(\mathcal{R}_7)$	$t(v(\mathcal{R}_7))$	$\overline{v}(\mathcal{R}_7)$	$t(v(\mathcal{R}_7))$
hc11p	1	604	0	604	12,32	604	12,46	604	12,21

Tabela 4.13: Resultados computacionais para os algoritmos **BC**, **LC** e **BC-LC**

Instância	k	$v(\mathcal{R}_7)$	BC-LC-RI		
			$\overline{v}(\mathcal{R}_7)$	$t(\overline{v}(\mathcal{R}_7))$	$t(v(\mathcal{R}_7))$
b07	1	318	318	0,004	0,007
	2	2637	2637	0,894	2,75
	3	7219	7219	1,286	4,218
b14	1	336	336	0,011	0,028
	2	2875	2875	2,202	6,883
	3	7391	7391	3,065	10,515
c08	1	344	344	0,162	0,292
	2	2946	2946	382,996	1402,995
	3	7586	7586	490,713	2314,311
c12	1	606	606	0,176	0,31
	2	2890	2890	416,377	1379,595
	3	7229	7229	510,852	2167,575
c17	1	852	852	0,235	0,385
	2	3847	3847	380,301	1242,321
	3	9013	9013	491,465	2221,831
bip52p	1	590	590	6,631	11,947
cc10-2u	1	599	599	1,808	2,918
cc3-11p	1	265	265	3,065	4,873
cc6-3p	1	377	377	0,906	7,687
hc11p	1	604	604	8,467	13,5

Tabela 4.14: Resultados computacionais para o algoritmo **BC-LC-RI**

Capítulo 5

Conclusão e Trabalhos Futuros

Investigamos nesta dissertação sete problemas NP-Difíceis de Otimização Combinatória. Cada um deles se caracteriza por permitir um único tipo de operação sobre digrafos (reversão, adição ou complemento de arcos) de forma a atender (ao menor custo total possível) a uma dada demanda de conectividade. Os problemas foram em grande parte introduzidos por ARKIN *et al.* [1]. Para cada um deles sugerimos (aparentemente pela primeira vez na literatura) formulações matemáticas, algoritmos de solução exata e conjuntos de instâncias de teste. Vistas sob esses ângulos, muitas das contribuições aqui propostas podem ser então consideradas originais.

Como parte de nossa investigação, introduzimos uma nova formulação de Programação Linear Inteira para o 1-PRVV. Ao contrario dos sete problemas aludidos acima, esse é um problema com complexidade de solução polinomial e formulações para o mesmo podem ser encontradas na literatura. No entanto, a exemplo de nossa própria formulação, nenhuma delas oferece garantias teóricas de levarem sempre a relaxações lineares naturalmente inteiras (e consequentemente ótimas) para o 1-PRVV. Em tese, obter uma tal formulação é possível (dado que o 1-PRVV pode ser resolvido em tempo polinomial). No entanto, até o momento, ainda não foi encontrada.

Apesar da existência de outras formulações para o 1-PRVV, nossa formulação se justifica por razões associadas à investigação mais ampla que aqui conduzimos. A mais importante delas é o fato de ser facilmente estendida aos problemas NP-Difíceis aqui investigados. Além disso, nos experimentos computacionais que efetuamos para o 1-PRVV, obteve sempre relaxações lineares naturalmente inteiras (e consequentemente ótimas) para o problema. Ou seja, apesar de não oferecer garantias teóricas de obter sempre relaxações lineares com esta característica, se mostrou empiricamente capaz de obtê-las (pelo menos para as instâncias de teste que utilizamos).

Em função do exposto acima, nossa formulação para o 1-PRVV se mostrou um ponto de partida muito adequado para formular todos os problemas NP-Difíceis estudados nesta dissertação. Para resolver (de forma exata) essas formulações, im-

plementamos algoritmos do tipo Branch-and-Cut (utilizando para tanto a estrutura disponibilizada pelo *solver* Gurobi de Programação Linear Inteira). Para testar os algoritmos, fomos obrigados a gerar conjuntos específicos de instâncias, diante da inexistência de alternativas óbvias na literatura.

É difícil atestar se as instâncias de teste aqui utilizadas são *difíceis* ou *fáceis* de resolver. Da mesma forma, diante da inexistência de outros conjuntos de teste e de formulações alternativas propostas para nossos problemas, é difícil atestar se nossas formulações são *fortes* ou *fracas*. No entanto, para as instâncias de teste que utilizamos, o desempenho de nossos algoritmos se mostrou extremamente positivo. Na maioria das vezes, obtiveram certificados de otimalidade logo nos nós zero de suas árvores de enumeração Branch-and-Cut. Além disso, os tempos de CPU gastos nesse processo foram, na maioria das vezes, *bastante aceitáveis* (para instâncias de problemas NP-Difíceis com as dimensões consideradas).

Dentre as diferentes variantes de algoritmos do tipo Branch-and-Cut que implementamos, o algoritmo **LC**, que se utiliza do conceito de *Lazy Constraints* (vide Capítulo 3), foi o que obteve melhor desempenho. Por outro lado, o algoritmo **BC**, que em tese deveria efetuar uma separação exaustiva de desigualdades de corte, colocou-se na outra extremidade. Motivos para justificar o fraco desempenho de **BC** (não necessariamente em ordem de importância) são: (a) o fato do Gurobi não implementar **BC** e o **BC-LC** exatamente como os especificamos e (b) a invariável ocorrência de *gaps* de dualidade muito pequenos, para todas as nossas instâncias de teste.

Em uma tentativa de remediar a dificuldade identificada no item (a) acima, implementamos o algoritmo **BC-LC-RI**. Ele envolve duas fases, a primeira delas consistindo num algoritmo de planos de corte que conduz (sob certas condições de parada) uma separação exaustiva de desigualdades de corte (para pontos inteiros ou não). Na segunda fase, as desigualdades de corte identificadas na fase anterior são utilizadas para inicializar um algoritmo de Lazy Constraints para resolver o problema. Especulamos que, para instâncias que envolvam *gaps* de dualidade significativos, a diferença de desempenho entre **LC** e os demais algoritmos deve encurtar significativamente.

Como parte do longo processo de refinamento e ajuste que conduzimos para nossos algoritmos, observamos que a componente capaz de, individualmente, levá-los aos maiores ganhos de desempenho, é a separação de desigualdades de corte. Como ressaltado na literatura, essa separação demanda um uso bastante intensivo de CPU, principalmente se os pontos do espaço Euclidiano a separar são não inteiros e o número de pares de vértices para os quais as desigualdades estão definidas é *elevado*. Por outro lado, como também indicado na literatura, constatamos, empiricamente, as vantagens algorítmicas associadas à separação de pontos inteiros (o que requer

um esforço computacional muito mais baixo do que no caso geral).

Diante das vantagens computacionais oferecidas pela separação de pontos inteiros, tentamos estendê-las à separação de pontos não inteiros. Mais especificamente, introduzimos o conceito de pontos quase inteiros. Definido um parâmetro ϵ suficientemente pequeno, um ponto é chamado de quase inteiro quando todas as suas coordenadas assumem valores menores ou iguais a ϵ ou maiores ou iguais a $(1 - \epsilon)$. Identificado tal ponto, suas coordenadas são arredondadas para 0, no primeiro caso, e para 1, no segundo. Separamos então o ponto inteiro que daí resulta e, caso este viole alguma desigualdade de corte, testamos se a mesma é também violada pelo ponto original. Confirmada a violação, a desigualdade é então utilizada para reforçar nossa formulação do problema. Em nosso trabalho, fixamos ϵ como 10^{-4}.

Ainda com o objetivo de reduzir o esforço computacional despendido na separação de desigualdades de corte (particularmente para pontos não inteiros), testamos diferentes estratégias para a condução desse procedimento. A ideia básica envolvida sendo a de evitar efetuar separações exaustivas (ou seja, entre todos os pares de vértices para os quais as desigualdades de corte estão definidas) em todas as iterações de nossos algoritmos de planos de corte. Para tanto, restringimos (em qualquer uma dessas iterações) o número de pares de vértices para os quais a separação seria efetuada. Se a proposta fosse bem sucedida e encontrássemos desigualdades violadas, a iteração seria então interrompida e tais desigualdades adicionadas à formulação do problema (passando-se, a seguir, para a próxima iteração do algoritmo de planos de corte). Caso contrário, nosso procedimento de separação restrito seria estendido a pares de vértices candidatos adicionais até que, eventualmente, encontrássemos desigualdades de corte violadas ou todos os pares de vértices aplicáveis tivessem sido testados sem, no entanto, encontrá-las (concluindo a execução do algoritmo de planos de corte e efetuando-se então um procedimento de ramificação na árvore Branch-and-Cut). Infelizmente, nos experimentos computacionais que efetuamos, o efeito prático desse procedimento de separação não foi o esperado e ele acabou sendo descartado.

Testamos também estratégias para identificar a ocorrência de *tailing off* e mitigar seus efeitos. *Tailing off* ocorre quando, ao longo da execução de uma algoritmo de planos de corte, um número elevado de iterações sucessivas de separação de planos de cortes é efetuada sem uma contrapartida aceitável em termos da melhora obtida no valor dos limitantes duais. No nosso caso, utilizamos como parâmetro de corte *20* iterações sucessivas sem uma melhora superior a 10^{-4} no valor dos limitantes duais. Identificada a ocorrência de *tailing off*, a execução do algoritmo de planos de corte era então interrompido e procedia-se a uma ramificação no nó (da árvore Branch-and-Cut) em questão. Nos experimentos computacionais que efetuamos, observamos ganhos de desempenho *modestos* diante da aplicação isolada desse procedimento.

Finalmente, concluindo esta dissertação, indicamos como metas para as nossas investigações futuras a obtenção de heurísticas de separação de desigualdades de corte de baixa complexidade computacional e alto desempenho. Em complemento a isso, vamos investigar também procedimentos que nos levem à geração aleatória de instâncias com *gaps* de dualidade mais expressivos para nossas formulações.

Referências Bibliográficas

[1] ARKIN, E. M., HASSIN, R., SHAHAR, S. "Increasing digraph arc-connectivity by arc addition, reversal and complement", *Discrete Applied Mathematics*, v. 122, n. 1, pp. 13–22, 2002. ISSN: 0166-218X. doi: https://doi.org/10.1016/S0166-218X(01)00319-5. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0166218X01003195>.

[2] HUANG, Y., SANTOS, A. C., DUHAMEL, C. "Model and methods to address urban road network problems with disruptions", *International Transactions in Operational Research*, v. 27, n. 6, pp. 2715–2739, 2020. doi: https://doi.org/10.1111/itor.12641. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12641>.

[3] DIESTEL, R. *Graph Theory*, v. 173, *Graduate Texts in Mathematics*. Fourth ed. Heidelberg, New York, Springer, 2010.

[4] ROBBINS, H. E. "A Theorem on Graphs, with an Application to a Problem of Traffic Control", *The American Mathematical Monthly*, v. 46, n. 5, pp. 281–283, 1939. ISSN: 00029890, 19300972. Disponível em: <http://www.jstor.org/stable/2303897>.

[5] ROBERTS, F. *Graph theory and its applications to problems of society*, v. 29. Philadelphia, PA., Society for Industrial and Applied Mathematics (SIAM), 1978.

[6] CHVÁTAL, V., THOMASSEN, C. "Distances in orientations of graphs", *Journal of Combinatorial Theory, Series B*, v. 24, n. 1, pp. 61–75, 1978. ISSN: 0095-8956. doi: https://doi.org/10.1016/0095-8956(78)90078-3. Disponível em: <https://www.sciencedirect.com/science/article/pii/0095895678900783>.

[7] MARTINS, A. X., DUHAMEL, C., SANTOS, A. C. "A column generation approach for the strong network orientation problem", *Electronic Notes in Discrete Mathematics*, v. 62, pp. 75–80, 2017. ISSN:

1571-0653. doi: https://doi.org/10.1016/j.endm.2017.10.014. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1571065317302524>. LAGOS'17 - IX Latin and American Algorithms, Graphs and Optimization.

[8] ESWARAN, K. P., TARJAN, R. E. "Augmentation Problems", *SIAM J. Comput.*, v. 5, n. 4, pp. 653–665, 1974. doi: 10.1137/0205044. Disponível em: <https://doi.org/10.1137/0205044>.

[9] KANT, G. "Augmenting Outerplanar Graphs", *Journal of Algorithms*, v. 21, n. 1, pp. 1–25, 1996. ISSN: 0196-6774. doi: https://doi.org/10.1006/jagm.1996.0034. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0196677496900346>.

[10] KAO, M.-Y. "Data Security Equals Graph Connectivity", *SIAM Journal on Discrete Mathematics*, v. 9, 02 2001. doi: 10.1137/S0895480193243274.

[11] LIU, Y., TRAPPE, W., GARNAEV, A. "Chapter 17 - Applications of Graph Connectivity to Network Security". In: DjuriÄ‡, P. M., Richard, C. (Eds.), *Cooperative and Graph Signal Processing*, Academic Press, pp. 445–467, NY, USA, 2018. ISBN: 978-0-12-813677-5. doi: https://doi.org/10.1016/B978-0-12-813677-5.00017-1. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780128136775000171>.

[12] FRANK, A. "Augmenting graphs to meet edge-connectivity requirements". In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pp. 708–718 vol.2, 1990. doi: 10.1109/FSCS.1990.89593.

[13] LJUBIC, I., RAIDL, G., KRATICA, J. "A Hybrid GA for the Edge-Biconnectivity Augmentation Problem". In: *Parallel Problem Solving from Nature - PPSN VI, 6th International Conference*, v. 1917, pp. 641–650, 09 2000. ISBN: 978-3-540-41056-0. doi: 10.1007/3-540-45356-3_63.

[14] KHULLER, S., RAGHAVACHARI, B., ZHU, A. "A uniform framework for approximating weighted connectivity problems". In: *SODA '99*, 1999.

[15] GABOW, H. "Centroids, Representations, and Submodular Flows", *Journal of Algorithms*, v. 18, n. 3, pp. 586–628, 1995. ISSN: 0196-6774. doi: https://doi.org/10.1006/jagm.1995.1022. Disponível em: <https://www.sciencedirect.com/science/article/pii/S019667748571022X>.

[16] FRANK, A. "An Algorithm for Submodular Functions on Graphs". In: Bachem, A., GrÃ¶tschel, M., Korte, B. (Eds.), *Bonn Workshop on Combi-*

natorial Optimization, v. 66, *North-Holland Mathematics Studies*, North-Holland, pp. 97–120, Netherlands, 1982. doi: https://doi.org/10.1016/S0304-0208(08)72446-0. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0304020808724460>.

[17] WOLSEY, L. A. "Well-Solved Problems". In: *Integer Programming*, cap. 3, pp. 43–62, New Jersey, NJ, USA, John Wiley & Sons, Ltd, 2020. ISBN: 9781119606475. doi: https://doi.org/10.1002/9781119606475.ch3. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119606475.ch3>.

[18] COCO, A. A., DUHAMEL, C., SANTOS, A. "Modeling and solving the multi-period disruptions scheduling problem on urban networks", *Annals of Operations Research*, v. 285, pp. 427–443, 2020.

[19] REBENNACK, S., ARULSELVAN, A., ELEFTERIADOU, L., et al. "Complexity analysis for maximum flow problems with arc reversals", *J. Comb. Optim.*, v. 19, pp. 200–216, 02 2010. doi: 10.1007/s10878-008-9175-8.

[20] AHUJA, R. K., MAGNANTI, T. L., ORLIN, J. B. *Network Flows: Theory, Algorithms, and Applications.* New York, Prentice Hall, 1993.

[21] GASKINS, R., TANCHOCO, J. "Flow path design for automated guided vehicle systems", *International Journal of Production Research*, v. 25, pp. 667–676, 1987.

[22] CPLEX, I. I. "V12. 1: User's Manual for CPLEX", *International Business Machines Corporation*, v. 46, n. 53, pp. 157, 2009.

[23] BONDY, J. A., MURTY, U. S. R. *Graph Theory with Applications.* New York, Elsevier, 1976.

[24] PADBERG, M., RINALDI, G. "A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems", *SIAM Review*, v. 33, n. 1, pp. 60–100, 1991. doi: 10.1137/1033004. Disponível em: <https://doi.org/10.1137/1033004>.

[25] BERTSIMAS, D., TSITSIKLIS, J. *Introduction to linear Optimization.* Belmont, Massachusetts, Athena Scientific, 1997.

[26] MARCHAND, H., MARTIN, A., WEISMANTEL, R., et al. "Cutting planes in integer and mixed integer programming", *Discrete Applied Mathematics*, v. 123, n. 1, pp. 397–446, 2002. ISSN: 0166-218X. doi: https://doi.

org/10.1016/S0166-218X(01)00348-1. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0166218X01003481>.

[27] DINITZ, Y. "Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation", *Soviet Math. Dokl.*, v. 11, pp. 1277–1280, 01 1970.

[28] HAKIMI, S. L. "Steiner's problem in graphs and its implications", *Networks*, v. 1, n. 2, pp. 113–133, 1971. doi: https://doi.org/10.1002/net.3230010203. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230010203>.

[29] BEASLEY, J. "An SST-based algorithm for the Steiner problem in graphs", *Networks*, v. 19, pp. 1–16, 1989.

[30] BEASLEY, J. "An algorithm for the Steiner problem in graphs", *Networks*, v. 14, pp. 147–159, 1984.

[31] ROSSETI, I., DE ARAGÃO, M., RIBEIRO, C., et al. "New benchmark instances for the Steiner problem in graphs". In: *Extended Abstracts of the 4th Metaheuristics International Conference (MIC'2001)*, pp. 557–561, Porto, 2001.

[32] VAN LINT, J. H. "Recent Results on Perfect Codes and Related Topics". In: Hall, M., van Lint, J. H. (Eds.), *Combinatorics*, pp. 163–183, Dordrecht, 1975. Springer Netherlands. ISBN: 978-94-010-1826-5.

[33] TARJAN, R. E. "A Note on Finding the Bridges of a Graph", *Inf. Process. Lett.*, v. 2, pp. 160–161, 1974.

[34] GUROBI OPTIMIZATION, LLC. "Gurobi Optimizer Reference Manual". 2021. Disponível em: <https://www.gurobi.com>.

[35] WOLSEY, L. A. "Branch and Bound". In: *Integer Programming*, cap. 7, pp. 43–62, New Jersey, NJ, USA, John Wiley & Sons, Ltd, 2020. ISBN: 9781119606475. doi: https://doi.org/10.1002/9781119606475.ch3. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119606475.ch3>.

[36] WOLSEY, L. A. "Cutting Plane Algorithms". In: *Integer Programming*, cap. 8, pp. 43–62, New Jersey, NJ, USA, John Wiley & Sons, Ltd, 2020. ISBN: 9781119606475. doi: https://doi.org/10.1002/9781119606475.ch3. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119606475.ch3>.

[37] TÓTH, S. F., MCDILL, M. E., KÖNNYÜ, N., et al. "Testing the Use of Lazy Constraints in Solving Area-Based Adjacency Formulations of Harvest Scheduling Models", *Forest Science*, v. 59, n. 2, pp. 157–176, 04 2013. ISSN: 0015-749X. doi: 10.5849/forsci.11-040. Disponível em: <https://doi.org/10.5849/forsci.11-040>.

[38] LERCH, D., TRAUTMANN, N. "A Lazy-Constraints Approach to Resource-Constrained Project Scheduling". In: *2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pp. 144–148, 2019. doi: 10.1109/IEEM44572.2019.8978524.

[39] AHO, A., AHO, V., HOPCROFT, J., et al. "Directed Graphs". In: *Data Structures and Algorithms*, cap. 6, Massachusetts, MA, USA, Addison-Wesley, 1983. ISBN: 9780201000238. Disponível em: <https://books.google.com.br/books?id=k8pQAAAAMAAJ>.

[40] SZWARCFITER, J. *Teoria Computacional De Grafos*. Rio de Janeiro, ELSEVIER (MEDICINA), 2018. ISBN: 9788535288841. Disponível em: <https://books.google.com.br/books?id=vy1SxQEACAAJ>.