Empacotamento de bicliques em grafos bipartidos

Alexandre da Silva Freire

Tese apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Doutor em Ciências

Programa: Ciência da Computação

Orientador: Prof. Dr. Carlos Eduardo Ferreira

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES e CNPq

São Paulo, outubro de 2012

Empacotamento de bicliques em grafos bipartidos

Esta versão da tese contém as correções e alteraçcões sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 02/10/2012. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof. Dr. Carlos Eduardo Ferreira (orientador) - IME-USP
- Profa. Dra. Yoshiko Wakabayashi - IME-USP
- Prof. Dr. Nelson Maculan - UFRJ
- Prof. Dr. Eduardo Xavier - UNICAMP
- Prof. Dr. Abílio Lucena - UFRJ

Agradecimentos

Agradeço principalmente ao meu orientador Carlinhos (Prof. Dr. Carlos Eduardo Ferreira), que me acompanhou ao longo de aproximadamente seis anos (2 anos de mestrado e 4 de doutorado). Depois de conviver tanto tempo com uma pessoa, torna-se muito difícil não herdar algumas de suas qualidades. Algumas das qualidades que eu enxergo no Carlinhos e que eu gostaria de herdar são: simplicidade, paciência, senso de humor e respeito às posições alheias, mesmo quando elas vão bastante contra as suas próprias. O Carlinhos teve muita paciência comigo no início de nosso trabalho, quando eu praticamente não sabia do que se tratava fazer uma pós-graduação. Muitas vezes ele respeitou minhas posições, mesmo sabendo que estavam equivocadas, deixando a cargo do tempo fazer o trabalho árduo de modificar a minha forma de pensar. Todas as vezes que entrei na sala do Carlinhos fui recebido com um sorriso. Nunca fui cobrado por resultados "melhores" e o pouco que fiz, dadas as minhas limitações, foi sempre recebido com elogios. Muito obrigado por tudo, Carlinhos!

Possivelmente, em algumas ocasiões serei bem visto por ter estudado no IME-USP. No entanto, sinto que pouco importa onde estudei, pois o que realmente me importa é lembrar com quem estudei. Sou imensamente grato às pessoas que ao longo de minha passagem pelo IME-USP me transmitiram parte de seus conhecimentos. Além de gratidão, sinto por essas pessoas muita admiração, respeito e afeto. Sentirei muitas saudades dos meus eternos professores: Yoshiko W., J. Coelho de Pina., Paulo Feofiloff (embora não tenha sido meu professor, aprendi muito com seus manuscritos), Cristina G. Fernandes, Carlinhos e José Augusto. Ao entrar em uma sala de aula em que uma dessas pessoas era o professor, me sentia como o doente que chega ao consultório médico e, ao perceber que está diante de um bom médico, já se sente praticamente curado.

Por sorte, fiz várias amizades durante o período em que passei pelo IME-USP e também durante o estágio de um ano que fiz em Lyon, França. Sou muito grato a essas pessoas queridas, pois certamente a minha vida teria sido insuportável sem a presença delas durante esse tempo.

Por fim, gostaria de deixar registrado um agradecimento especial à minha Mãe, que sempre me apoiou incondicionalmente.

Resumo

FREIRE, A. S. **Empacotamento de bicliques em grafos bipartidos**. 2012, 74f. Tese (Doutorado) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2012.

Nesta tese, estudamos o problema de EMPACOTAMENTO DE BICLIQUES. Um *biclique* é um grafo bipartido completo. No problema de EMPACOTAMENTO DE BICLIQUES são dados um inteiro k e um grafo bipartido G e deseja-se encontrar um conjunto de k bicliques, subgrafos de G, dois a dois disjuntos nos vértices, tal que a quantidade total de arestas dos bicliques escolhidos seja máxima. No caso em que $k = 1$, temos o problema de BICLIQUE MÁXIMO. Esses dois problemas possuem aplicações na área de Bioinformática. Mantemos neste trabalho um enfoque prático, no sentido de que nosso interesse é resolver instâncias desses dois problemas com tamanho razoavelmente grande. Para isso, utilizamos técnicas de Programação Linear Inteira. Para avaliar os métodos propostos aqui, mostramos resultados de experimentos computacionais feitos com instâncias vindas de aplicações e também com instâncias geradas aleatoriamente.

Palavras-chave: geração de colunas, biclique máximo, empacotamento de bicliques, programação inteira.

Abstract

FREIRE, A. S. **Biclique packing in bipartite graphs**. 2012, 74f. Tese (Doutorado) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2012.

In this thesis, we study the BICLIQUE PACKING problem. A *biclique* is a complete bipartite graph. In the BICLIQUE PACKING problem we are given an integer k and a bipartite graph G and we want to find a set of k vertex disjoint bicliques of G, such that the total number of biclique's edges is maximum. When $k = 1$, we have the MAXIMUM BICLIQUE problem. These two problems have applications in Bioinformatics. In this work we keep a practical focus, in the sense that we are interested in solving large size instances of these problems. To tackle these problems, we use Integer Linear Programming techniques. In order to evaluate the methods proposed here, we show results of computational experiments carried out with practical application's instances and also with randomly generated ones.

Keywords: column generation, maximum biclique, biclique packing, integer programming.

Sumário

Capítulo 1

Introdução

Nesta tese, tratamos o problema de EMPACOTAMENTO DE BICLIQUES. Um *biclique* é um grafo bipartido completo, ou seja, um grafo no qual todos os vértices de uma classe da bipartição são adjacentes a todos os vértices da outra classe da bipartição. No problema de EMPACOTAMENTO DE BICLIQUES são dados um inteiro k e um grafo bipartido G, e deseja-se encontrar um conjunto de k bicliques, subgrafos de G, dois a dois disjuntos nos vértices, tal que a quantidade total de arestas dos bicliques escolhidos seja máxima. Consideramos também o caso em que a restrição de disjunção aplica-se apenas a uma das classes da bipartição. No caso em que $k = 1$, temos o problema de BICLIQUE MÁXIMO, para o qual existem diversos resultados na literatura, conforme detalharemos neste capítulo. Não temos conhecimento de nenhum resultado existente na literatura, anterior ao nosso trabalho, para o caso geral do problema de EMPACOTAMENTO DE BICLIQUES. Tomamos conhecimento deste problema através de nossos colaboradores chilenos Eduardo Moreno e Vicente Acuña, que nos apresentaram uma aplicação em bioinformática que eles tentavam modelar como um problema de otimização. Formalizamos o problema de EMPACOTAMENTO DE BICLIQUES juntamente com esses colaboradores.

Neste capítulo, apresentamos uma breve revisão sobre alguns conceitos básicos necessários para a compreensão desta tese, introduzimos a notação que será utilizada ao longo de todo o texto, enunciamos formalmente os problemas investigados nesta tese, descrevemos as aplicações que nos motivaram a estudar o problema de EMPACOTAMENTO DE BICLIQUES e apresentamos resultados encontrados na literatura sobre o problema de BICLIQUE MÁXIMO. Mencionamos também nossas contribuições obtidas durante o doutorado, descrevemos os objetivos de nosso trabalho e, por fim, descrevemos como está organizado o restante do texto.

1.1 Conceitos Básicos

Para a compreensão do conteúdo exposto nesta tese, é necessário que o leitor tenha um conhecimento básico de Programação Linear e Programação Linear Inteira. Nesta seção, apresentamos uma pequena resenha sobre esses assuntos, baseada nos livros de Chvátal (1983), Schrijver (1986) e Wolsey (1998). Apresentamos também algumas definições básicas da área de Algoritmos de Aproximação, baseadas no livro de Carvalho *et al.* (2001).

1.1.1 Programação Linear

Considere o seguinte problema.

Problema 1.1.1. *Dados vetores $c \in \mathbb{R}^m$, $b \in \mathbb{R}^n$ e uma matriz $A \in \mathbb{R}^{n \times m}$, encontrar um vetor $x \in \mathbb{R}^m$ que maximize $c^\top x$, tal que $Ax \leq b$ e $x \geq 0$.*

Dizemos que a tripla (c, b, A) é um *programa linear*, no qual $c^\top x$ é a *função objetivo* e o sistema de inequações $Ax \leq b$ e $x \geq 0$ são as *restrições*. Cada vetor x que satisfaz as restrições do programa linear é uma *solução viável*. Uma solução viável x^* é *ótima* se $c^\top x^* \geq c^\top x$, para toda solução viável x. Dizemos que $c^\top x$ é o *valor* da solução x. Dizemos que um programa linear é *inviável*, se ele não possui soluções viáveis, e *ilimitado*, se ele possui uma solução viável, mas não possui uma solução ótima (ou seja, para qualquer solução viável x, existe uma solução viável x', tal que $c^\top x' > c^\top x$). Por simplicidade, trataremos apenas de programas lineares viáveis e limitados. Um programa linear pode ser de minimização ou maximização, suas restrições são descritas através de equações e/ou inequações lineares, e suas variáveis podem assumir qualquer valor real. No entanto, adotamos a *forma padrão* para descrever programas lineares, na qual existe uma restrição de não-negatividade sobre cada variável e as demais restrições são todas da forma $A_i x \leq b_i$, no caso de maximização, e $A_i x \geq b_i$ no caso de minimização. Dado um programa linear que não está na forma padrão, podemos facilmente escrever um programa linear equivalente na forma padrão, no qual cada igualdade $A_i x = b_i$ é substituida pelas desigualdades $A_i x \leq b_i$ e $-A_i x \leq -b_i$, e cada variável x_j livre de sinal é substituída por duas variáveis x_j^+ e x_j^-, de forma que $x_j = x_j^+ - x_j^-$ e $x_j^+, x_j^- \geq 0$. Considere os seguintes programas lineares.

$$\begin{array}{rlrcl} & \text{maximize} & c^\top x & & \\ \text{(PL)} & \text{sujeito a} & Ax & \leq & b \\ & & x & \geq & 0 \end{array} \qquad \begin{array}{rlrcl} & \text{minimize} & b^\top y & & \\ \text{(DL)} & \text{sujeito a} & A^\top y & \geq & c \\ & & y & \geq & 0 \end{array}$$

Dizemos que o programa linear (PL) é *primal* e o programa linear (DL) é o *dual* de (PL). Equivalentemente, podemos considerar que (DL) é primal e (PL) é o seu dual. Ou seja, todo programa linear possui um dual, sendo que se o primal for de maximização, o dual será de minimização, e vice-versa. Dizemos que (x, y) é um *par primal-dual* se x é uma solução viável de (PL) e y é uma solução viável de (DL). Podemos reescrever (PL) e (DL) da seguinte maneira.

$$\begin{array}{rlrcl} & \text{maximize} & c^\top x & & \\ \text{(PL}') & \text{sujeito a} & Ax + w & = & b \\ & & x, w & \geq & 0 \end{array} \qquad \begin{array}{rlrcl} & \text{minimize} & b^\top y & & \\ \text{(DL}') & \text{sujeito a} & A^\top y - z & = & c \\ & & y, z & \geq & 0 \end{array}$$

As variáveis w e z são chamadas de *variáveis de folga*. Abaixo, enunciamos os principais teoremas da área de Programação Linear.

Teorema 1.1.1 (Dualidade Fraca)**.** *Dado um par primal-dual (x, y), temos que $c^\top x \leq b^\top y$.*

Teorema 1.1.2 (Dualidade Forte)**.** *Dado um par primal-dual (x^*, y^*), temos que se x^* é uma solução ótima de* (PL) *e y^* é uma solução ótima de* (DL)*, então $c^\top x^* = b^\top y^*$.*

Teorema 1.1.3 (Folgas Complementares)**.** *Dado um par primal-dual (x^*, y^*) e as respectivas variáveis de folga (z^*, w^*), temos que x^* e y^* são soluções ótimas de* (PL) *e* (DL)*, respectivamente,*

se e somente se $x_j^* \cdot z_j^* = 0$, *para* $j = 1, 2, \ldots, n$, *e* $w_i^* \cdot y_i^* = 0$, *para* $i = 1, 2, \ldots, m$.

Dada uma solução ótima x^* de (PL), podemos facilmente encontrar uma solução ótima y^* de (DL), utilizando o teorema de folgas complementares e, por essa razão, dizemos que tal solução y^* é *dual complementar* de x^*. Para cada variável de folga w_i^* com valor positivo, temos que o valor da variável dual y_i^* deve ser igual a zero. Para que tal condição seja satisfeita, adicionamos a restrição $y_i = 0$ em (DL), para todo i, tal que $w_i^* > 0$, obtendo o programa linear (DL*). Para obter os valores das demais variáveis duais y^*, basta encontrar uma solução viável de (DL*).

Em 1947, George B. Dantzig publicou o primeiro algoritmo, denominado *simplex*, para resolver programas lineares. Na época, não havia um estudo detalhado sobre a complexidade do simplex. Em 1972, Victor Klee e George J. Minty apresentaram um exemplo no qual o simplex consumia tempo exponencial no tamanho da entrada. Trinta e dois anos após a publicação do simplex, em 1979, Leonid Khachiyan mostrou que era possível resolver qualquer programa linear em tempo polinomial, através do método dos *elipsoides*. No entanto, essa descoberta foi apenas de interesse teórico, pois na prática o método dos elipsoides era pouco eficiente. Em 1984, Narendra Karmarkar introduziu o método de *pontos interiores*, que também tem consumo de tempo polinomial no tamanho da entrada, mas além disso, obtém um bom desempenho na prática, superando o simplex em algumas situações. Apesar de o consumo de tempo no pior caso do algoritmo simplex ser exponencial no tamanho da entrada, até os dias de hoje ele é o algoritmo mais utilizado para resolver programas lineares. Isso deve-se ao fato de que os casos em que seu consumo de tempo é exponencial aparecem muito raramente na prática.

1.1.2 Programação Linear Inteira

Na área de *Programação Linear Inteira*, consideramos programas lineares nos quais as variáveis devem assumir apenas valores inteiros. Abaixo, apresentamos um Programa Linear Inteiro (PLI).

$$\begin{array}{lrcl} & \text{maximize} & c^\top x & \\ \text{(PI)} & \text{sujeito a} & Ax & \leq & b \\ & & x & \in & \mathbb{N}^n \end{array}$$

Podemos considerar também problemas de *Programação Linear Inteira Mista*, nos quais a restrição de integralidade é aplicada apenas a uma parte das variáveis. Dizemos que um PLI é uma formulação para um problema se qualquer solução ótima do PLI corresponde a uma solução ótima do problema original. Muitos problemas de Otimização Combinatória podem ser formulados como um PLI. No geral, o problema de resolver um PLI é NP-difícil. Dadas soluções x^* e x^{**} ótimas de (PL) e (PI), respectivamente, observe que $c^\top x^{**} \leq c^\top x^*$, pois toda solução viável de (PI) é também viável de (PL) (a recíproca, em geral, não é verdadeira). Ou seja, o valor de uma solução ótima de (PL) é um limitante superior no valor de uma solução ótima de (PI) (limitante inferior, no caso de minimização). Dizemos que o programa linear (PL) é a *relaxação linear* de (PI). Existem diversas técnicas para a resolução de PLIs, sendo que a escolha de qual técnica utilizar deve considerar principalmente a estrutura do PLI em questão.

1.1.3 Algoritmos de Aproximação

Um problema de otimização (minimização ou maximização) é composto por um conjunto de instâncias, um conjunto $Sol(I)$ de soluções viáveis para cada instância I, e uma função que atribui um número não-negativo $val(S)$ a cada solução viável S. Assuma que estamos tratando de um problema de minimização. Uma solução S é *ótima* se $val(S) \leq val(S')$ (no caso de maximização, $val(S) \geq val(S')$), para toda solução viável S'. O valor de qualquer das soluções ótimas para uma instância I será denotado por $opt(I)$. Seja A um algoritmo que, para toda instância I do problema, devolve uma solução viável $A(I)$ de $Sol(I)$. Observe que $val(A(I)) \geq opt(I)$ (no caso de maximização, $val(A(I)) \leq opt(I)$), para toda instância I. Dizemos que A é uma α-aproximação se $val(A(I)) \leq \alpha \cdot opt(I)$ (no caso de maximização, $val(A(I)) \geq \alpha \cdot opt(I)$), para toda instância I. O valor de α é o *fator de aproximação* do algoritmo A. Observe que $\alpha \geq 1$ (no caso de maximização, $\alpha \leq 1$).

O interesse por algoritmos de aproximação surge em um contexto no qual o problema a ser resolvido é NP-difícil e deseja-se encontrar um algoritmo com consumo de tempo polinomial no tamanho da entrada que garanta um bom fator de aproximação. Um problema de otimização é dito *inaproximável* se ele não admite um algoritmo de tempo polinomial com fator de aproximação $O(n^\epsilon)$ (no caso de maximização $\frac{1}{O(n^\epsilon)}$), onde n é o tamanho da entrada, para algum $\epsilon > 0$, a menos que P = NP.

1.2 Notação e Definições

Seja $G = (V, E)$ um grafo bipartido. As duas classes da bipartição de V são denotadas por L e R (ou seja, $V = L \cup R$ e $L \cap R = \emptyset$). Por simplicidade, dada uma aresta uv de E, assumimos por convenção que $u \in L$ e $v \in R$. Em outras palavras, o rótulo do lado esquerdo corresponde ao vértice que pertence à classe L e o rótulo do lado direito corresponde ao vértice que pertence à classe R. Dado um vértice $u \in V$, denotamos por $N(u)$ o conjunto dos vértices adjacentes a u. Dado um subconjunto de vértices $V' \subset V$, denotamos por $N^\cap(V') = \cap_{u \in V'} N(u)$ o conjunto dos vértices que são adjacentes a todos os vértices de V', e denotamos por $G[V']$ o subgrafo de G induzido por V'. Dado um subgrafo B de G, denotamos seu conjunto de arestas por E_B e seu conjunto de vértices por V_B. As duas classes da bipartição de V_B são denotadas por $L_B = V_B \cap L$ e $R_B = V_B \cap R$.

Dizemos que um grafo bipartido B é um *biclique* se para todo par de vértices u e v, tal que $u \in L_B$ e $v \in R_B$, temos que $uv \in E_B$. Dados subconjuntos $A \subseteq L$ e $C \subseteq R$, tais que $uv \in E$, para todo $u \in A$ e $v \in C$, denotamos por (A, C) o biclique B, onde $L_B = A$ e $R_B = C$, sendo que o conjunto de arestas $E_B = \{uv \in E \mid u \in A \text{ e } v \in C\}$ fica implícito. Por simplicidade, utilizaremos a frase "um biclique B em G", com o significado de "um biclique B subgrafo de G". Dizemos que um biclique B em G é *maximal*, se não existe biclique B' em G, tal que $E_B \subset E_{B'}$, ou *máximo*, se não existe biclique B' em G, tal que $|E_{B'}| > |E_B|$.

Um *empacotamento* de bicliques em G é um conjunto P de bicliques em G, dois a dois disjuntos nos vértices que estão contidos em S, onde $S = V$ ou $S \in \{L, R\}$. Se $S = V$, cada vértice pode estar contido em no máximo um biclique de P, enquanto que se $S \in \{L, R\}$, esta restrição aplica-se apenas aos vértices de uma das classes da bipartição. Nos dois casos, cada aresta de E pode estar contida em no máximo um biclique de P. Investigamos nesta tese o problema enunciado abaixo.

Problema 1.2.1. *Dado um grafo bipartido G e um inteiro positivo k, o problema de* EMPACOTAMENTO DE BICLIQUES *consiste em encontrar um empacotamento P de bicliques em G, tal que $|P| \leq k$ e $\sum_{B \in P} |E_B|$ seja máximo.*

No caso especial do problema de EMPACOTAMENTO DE BICLIQUES em que $k = 1$, temos o problema de BICLIQUE MÁXIMO, que pode ser enunciado da seguinte maneira.

Problema 1.2.2. *Dado um grafo bipartido G, o problema de* BICLIQUE MÁXIMO *consiste em encontrar um biclique máximo em G.*

Um caso mais geral do problema de BICLIQUE MÁXIMO é o problema de BICLIQUE DE PESO MÁXIMO, enunciado a seguir.

Problema 1.2.3. *Dado um grafo bipartido G e uma função $w : E \to \mathbb{R}$, tal que $w(uv) \geq 0$, para toda aresta uv de E, o problema de* BICLIQUE DE PESO MÁXIMO *consiste em encontrar um biclique B em G, tal que $\sum_{uv \in E_B} w(uv)$ seja máximo.*

Apesar de o problema de BICLIQUE DE PESO MÁXIMO estar fora do escopo principal deste trabalho, julgamos interessante fazer algumas considerações sobre este problema ao longo do texto, pois muitas das abordagens que apresentamos para resolver o problema de BICLIQUE MÁXIMO são facilmente adaptáveis para resolver o problema de BICLIQUE DE PESO MÁXIMO.

Dado um grafo arbitrário H, denotamos seu conjunto de vértices por $V(H)$ e seu conjunto de arestas por $A(H)$. A seguir, enunciamos alguns problemas clássicos de Otimização Combinatória que, conforme veremos nas próximas seções, possuem alguma relação com o problema de BICLIQUE MÁXIMO.

Problema 1.2.4. *Dado um grafo H, o problema de* CONJUNTO INDEPENDENTE MÁXIMO *consiste em encontrar um conjunto Q de vértices de $V(H)$ de cardinalidade máxima, tal que $\{u, v\} \notin A(H)$, para todo par u, v em Q.*

Problema 1.2.5. *Dado um grafo H, o problema de* CLIQUE MÁXIMO *consiste em encontrar um conjunto Q de vértices de $V(H)$ de cardinalidade máxima, tal que $\{u, v\} \in A(H)$, para todo par de vértices distintos u, v em Q.*

Problema 1.2.6. *Dado um grafo H e uma função $w : V(H) \to \mathbb{R}$, o problema de* CLIQUE DE PESO MÁXIMO *consiste em encontrar um conjunto Q de vértices de $V(H)$, tal que $\{u, v\} \in A(H)$, para todo par de vértices distintos u, v em Q, e $\sum_{v \in Q} w(v)$ é máximo.*

Problema 1.2.7. *Dado um grafo H, o problema de* COBERTURA MÍNIMA DE ARESTAS POR VÉRTICES *consiste em encontrar um conjunto Q de vértices de $V(H)$ de cardinalidade mínima, tal que para toda aresta $\{u, v\}$ em $A(H)$, temos que $u \in Q$ ou $v \in Q$.*

1.3 Aplicações Práticas

Uma *rede metabólica* consiste em um conjunto de compostos químicos e um conjunto de reações químicas que consomem e produzem subconjuntos dos compostos químicos da rede em uma certa proporção. Um *modo* de uma rede metabólica é um conjunto de reações em uma dada proporção que operam em *estado estacionário*, no qual todos os compostos químicos são produzidos e consumidos

na mesma proporção. Um modo é dito *elementar* quando não existe um subconjunto próprio de suas reações que seja também um modo. No artigo de Schuster e Hilgetag (1994), é dada uma explanação mais detalhada desses conceitos, de um ponto de vista de biologia. Não raro, o número de modos elementares presentes em uma rede metabólica é enorme e, portanto, torna-se impraticável analisá-los diretamente, sem agrupá-los de alguma forma. Por exemplo, de acordo com Terzer e Stelling (2008), o metabolismo central da bactéria *Escherichia Coli* corresponde a uma rede metabólica com aproximadamente cem reações químicas e vinte e seis milhões de modos elementares. Neste contexto, uma forma de facilitar a interpretação dos dados é separar k conjuntos de reações químicas de forma que cada conjunto contenha reações que participam juntas de muitos modos elementares, conforme descrito no trabalho de Notebaart *et al.* (2008). Isto corresponde a resolver o problema de EMPACOTAMENTO DE BICLIQUES no grafo em que L representa o conjunto de reações químicas e R representa o conjunto de modos elementares. Em tal grafo, existe uma aresta $uv \in E$ se e só se a reação u está presente no modo v, e os bicliques empacotados são dois a dois disjuntos nos seus subconjuntos de reações químicas (ou seja, $S = L$). Note que nesta aplicação o grafo G é bastante desbalanceado (no exemplo citado, $|L| \simeq 100$ e $|R| \simeq 26.000.000$).

Outra aplicação que consideramos é a de EMPACOTAMENTO DE PRODUTOS, na qual são dados um conjunto de clientes e um conjunto de produtos, bem como a demanda de cada cliente por cada produto, e o objetivo é definir no máximo k pacotes de produtos e estipular as quantidades de cada produto em cada pacote, de forma a maximizar a venda total de produtos. Assume-se que cada cliente compra a quantidade máxima de pacotes tal que não ultrapasse sua demanda por nenhum produto. O problema de empacotamento de produtos é considerado nos trabalhos de McCardle *et al.* (2007); Stremersch e Tellis (2002). Nesta aplicação, os vértices de L representam os clientes, os vértices de R representam os produtos e cada aresta uv em E indica que o cliente u possui demanda positiva pelo produto v, sendo que o valor da demanda é dado através de uma função de pesos $w : E \to \mathbb{N}$ associada às arestas de E. Um biclique neste grafo corresponde a um conjunto de clientes que desejam comprar um conjunto de produtos. O objetivo é encontrar k pacotes de produtos que maximizam a venda total de produtos, sujeito ao limite da demanda de cada cliente por cada produto. No caso em que $S = R$, cada cliente pode comprar no máximo um pacote de produtos, e quando $S = L$, cada produto pode aparecer em no máximo um pacote. Nesta aplicação também ocorre que o grafo G é bastante desbalanceado (o número de clientes é muito maior do que o número de produtos). No Capítulo 4, consideramos o caso especial do problema de criação de pacotes em que $k = 1$. Considere uma variante simplificada desta aplicação em que as demandas são binárias. Neste caso, temos uma instância do problem de EMPACOTAMENTO DE BICLIQUES.

No artigo de Swaminathan e Tayur (1998), é descrita uma aplicação na qual L representa um conjunto de produtos a serem montados (computadores de diversos modelos, por exemplo), R representa um conjunto de componentes (por exemplo: chip de memória, unidade de disco rígido, unidade de CD, etc), e existe uma aresta uv em E se e só se o produto u contém o componente v. Nesse contexto, um biclique B em G representa um *vanilla box*, que consiste em um conjunto de componentes R_B que estão contidos em todos os produtos de L_B. Em muitos casos, os recursos consumidos na montagem de um produto (o tempo, por exemplo) são diretamente proporcionais à quantidade de componentes contidos em tal produto, sendo que o consumo de tais recursos é reduzido quando se utiliza um *vanilla box* previamente fabricado na montagem de um produto.

Portanto, quanto maior for a quantidade de componentes contidos em um *vanilla box*, menor o consumo de recursos na montagem dos produtos que o utilizam. Um *vanilla box* só pode ser utilizado na montagem de um produto que contém todos os seus componentes. Portanto, quanto maior for o número de componentes de um *vanilla box*, menor tende a ser a quantidade de produtos que podem utilizá-lo em sua montagem. Um empacotamento de bicliques, considerando $S = R$, representa um conjunto de k *vanilla boxes* que resultam na maior economia de recursos, considerando a produção total dos produtos. No artigo de Swaminathan e Tayur (1998), são apresentadas apenas heurísticas para resolver o problema e são considerados outros fatores, tais como a demanda esperada de cada produto, custo para manter um *vanilla box* em estoque, etc. Essa aplicação surgiu como alternativa para diminuir o tempo de espera pela montagem de computadores na fábrica da IBM, atendendo a pedidos customizados, nos quais os clientes podiam escolher com grande flexibilidade as configurações dos computadores que estavam adquirindo.

No artigo de Ding *et al.* (2006), é apresentada uma aplicação do problema de BICLIQUE MÁXIMO na análise de *redes de interação de genes*. Nesta aplicação, os vértices de L representam um conjunto de genes, os vértices de R representam um conjunto de condições observadas em experimentos e uma aresta uv está presente em E se e só se o gene u é expresso na condição v. Deseja-se encontrar um biclique máximo em G, que representa um conjunto de genes e condições de maior interação. Madeira e Oliveira (2004) apresentam um resumo no qual listam diversas aplicações do problema de BICLIQUE MÁXIMO em análise de dados biológicos e também em outras áreas.

1.4 Resultados Encontrados na Literatura

Seja $G = (V, E)$ um grafo bipartido. No livro de Garey e Johnson (1979), são discutidas as complexidades de diferentes problemas relacionados a encontrar um biclique em G. O problema de encontrar um biclique *balanceado* B em G, tal que $|L_B| = |R_B| = k$, para um dado inteiro k, é NP-completo. No caso em que deseja-se encontrar um biclique em G, tal que $|L_B| + |R_B| \geq k$, existe algoritmo com consumo de tempo polinomial no tamanho da entrada. Essa afirmação baseia-se no fato de que encontrar um biclique com k vértices em G é equivalente a encontrar um conjunto independente máximo no grafo bipartido $G' = (V', E')$, onde $V' = V$ e $E' = \{uv \mid u \in L, v \in R \text{ e } uv \notin E\}$. Como G' é bipartido, conforme o teorema de König, neste caso o problema de CONJUNTO INDEPENDENTE MÁXIMO é equivalente ao problema de EMPARELHAMENTO MÁXIMO, que pode ser resolvido em tempo polinomial. Mesmo no caso em que é dado um grafo arbitrário H (ou seja, H não necessariamente é bipartido), o problema de encontrar um biclique com número máximo de vértices pode ser resolvido em tempo polinomial. Essa afirmação baseia-se no fato de que existe uma correspondência entre os bicliques de H e os bicliques do grafo bipartido G, onde cada classe L e R dos vértices de G consiste em uma cópia de $V(H)$ e para cada aresta $\{u, v\}$ em $A(H)$, temos as arestas uv e vu em E. No entanto, o problema de encontrar um biclique induzido com número máximo de vértices em um grafo arbitrário é NP-difícil, conforme mostrado no trabalho de Yannakakis (1978) (observe que no caso em que G é bipartido, todo biclique em G é induzido).

Não pudemos encontrar na literatura qual foi a primeira vez em que foi proposto o problema de BICLIQUE MÁXIMO, que consiste em encontrar um biclique com número máximo de arestas em um grafo bipartido G. Nos artigos de Dawande *et al.* (1997), Hochbaum (1998) e Haemers (2001), comenta-se sobre a complexidade do problema, sugerindo que provavelmente tratava-se de um pro-

blema NP-difícil, porém sem fornecer nenhuma prova de que tal afirmação fosse verdadeira. Em particular, no artigo de Dawande *et al.* (1997) é apresentado um algoritmo com razão de aproximação esperada igual a 2, para grafos aleatórios suficientemente densos. Nesse mesmo artigo, é mostrado que o problema de BICLIQUE DE PESO MÁXIMO é NP-difícil. Alexe *et al.* (2004) e Nussbaum *et al.* (2010) introduziram, independentemente, algoritmos com consumo de tempo polinomial no tamanho da entrada para o caso especial em que G é *convexo*, sendo que G é dito convexo se existe uma permutação $(v_1, v_2, \ldots, v_m)$ dos vértices de R, tal que $N(u) = (v_i, v_{i+1}, \ldots, v_{i+|N(u)|-1})$ é um segmento de $(v_1, v_2, \ldots, v_m)$, para todo u em L. No artigo de Ding *et al.* (2006), é apresentado um método de Otimização Contínua para o problema de BICLIQUE MÁXIMO. Tal algoritmo é baseado em um teorema de *Motzkin-Straus* que relaciona o problema de CLIQUE MÁXIMO ao problema de otimizar uma certa função quadrática. Os autores fazem uma adaptação desse teorema e mostram que o algoritmo apresentado converge em tempo polinomial, caso exista apenas um biclique máximo em G. Na prática, esse algoritmo obtém um desempenho razoável apenas para instâncias de pequeno tamanho. O problema de BICLIQUE MÁXIMO possui uma forte relação com o problema de CONJUNTO INDEPENDENTE MÁXIMO ou, equivalentemente, com o problema de CLIQUE MÁXIMO, para os quais encontra-se uma vasta literatura. A seguir, mostramos em mais detalhes a relação entre esses problemas.

Dizemos que duas arestas uv e pq de E são *incompatíveis* se $u \neq p$, $v \neq q$ e $uq \notin E$ ou $pv \notin E$. Seja $\mathcal{I} = \{\{uv, pq\} \in E \times E \mid uv \text{ e } pq \text{ são incompatíveis}\}$ o conjunto de todos os pares de arestas incompatíveis de E. Seja H o grafo em que para cada aresta uv de E existe um vértice de rótulo $l(uv)$ em $V(H)$, e $\{l(uv), l(pq)\} \in A(H)$ se e só se $\{uv, pq\} \in \mathcal{I}$. Observe que encontrar um biclique máximo em G equivale a encontrar um conjunto independente máximo em H ou, equivalentemente, encontrar um clique máximo no grafo $\overline{H}$, onde $\overline{H}$ é o complemento de H (ou seja, $V(\overline{H}) = V(H)$ e $e \in A(\overline{H})$ se e só se $e \notin A(H)$). Na Seção 2.1, voltaremos a falar sobre a relação entre esses problemas. O problema de decidir se um grafo contém um clique com k vértices consta na famosa lista de 21 problemas NP-completos de Karp (1972). No artigo de Berman e Schnitger (1992), é provado que o problema de CLIQUE MÁXIMO é inaproximável. Considere a versão de decisão dos problemas de BICLIQUE MÁXIMO e CLIQUE MÁXIMO, conforme enunciamos abaixo.

Problema 1.4.1. *Dados um grafo H e um inteiro q, decidir se H contém um clique com q vértices.*

Problema 1.4.2. *Dados um grafo bipartido G e um inteiro k, decidir se G contém um biclique com pelo menos k arestas.*

A complexidade do problema de BICLIQUE MÁXIMO ficou em aberto até o ano de 2003. A seguir, apresentamos a prova contida no artigo de Peeters (2003), para mostrar que o problema de BICLIQUE MÁXIMO é NP-difícil, através da redução do Problema 1.4.1 ao Problema 1.4.2. Essa prova foi inspirada na redução de Garey e Johnson (1979) do Problema 1.4.1 ao problema de encontrar um biclique B balanceado com $|L_B| = |R_B| = k$ em um grafo bipartido.

Lema 1.4.1. *O problema de* BICLIQUE MÁXIMO *é* NP-difícil.

Demonstração. Seja (H, q) uma instância do Problema 1.4.1. Sem perda de generalidade, assumimos que $|V(H)|$ é par e $q = \frac{|V(H)|}{2}$. Construímos uma instância (G, k) do Problema 1.4.2, tal que $k = q^3 - \frac{3}{2}q^2$, $L = V(H)$, $R = A(H) \cup W$, onde W contém $\frac{1}{2}q^2 - q$ novos elementos, e $E = \{uv \mid u \in V(H) \text{ e } v \in W\} \cup \{ua \mid u \in V(H), a \in A(H) \text{ e } u \notin a\}$. Observe que essa construção é feita em tempo polinomial no tamanho de H.

Suponha que H contém um clique C com q vértices. Seja $L_B = V(H) \setminus C$ e $R_B = W \cup \{\{u, v\} \in A(H) \mid u, v \in C\}$. Observe que $B = (L_B, R_B)$ é um biclique em G, tal que $|E_B| = |L_B| \cdot |R_B| = (|V(H)| - |C|) \cdot (|W| + \frac{|C|^2 - |C|}{2}) = (2q - q) \cdot (\frac{1}{2}q^2 - q + \frac{q^2 - q}{2}) = q^3 - \frac{3}{2}q^2 = k$. Portanto, se H contém um clique com q vértices, então G contém um biclique com k arestas.

Agora, suponha que H não contém um clique com q vértices. Seja $B = (L_B, R_B)$ um biclique máximo em G. É suficiente mostrar que $|E_B| < k$. Pela escolha de B e pela construção de G, temos que $W \subseteq R_B$. Sejam $l = |L_B|$ e $r = |R_B| - |W|$. Cada vértice a em $R_B \setminus W$ corresponde a uma aresta $a = \{u, v\}$ de $A(H)$, tal que $ua, va \notin E$ e, consequentemente, $u, v \notin L_B$. Como existem $2q - l$ vértices contidos em $L \setminus L_B$, temos que $r \leq \frac{1}{2}(2q - l)(2q - l - 1)$, sendo que vale a igualdade se e só se $C = V(H) \setminus L_B$ é um clique em H, onde $A(C) = R_B \setminus W$. Dividimos o restante da prova nos dois casos abaixo.

- Suponha que $l > q$. Neste caso, temos que $|L \setminus L_B| = q - c$, onde $c = l - q$ e $0 \leq c \leq q$. Logo, $r \leq \frac{1}{2}(q - c)(q - c - 1)$ e, consequentemente, temos que

$$\begin{aligned} |L_B| \cdot |R_B| &= l \cdot (|W| + r) \\ &\leq (q + c) \cdot \left[\tfrac{1}{2}q^2 - q + \tfrac{1}{2}(q - c)(q - c - 1)\right] \\ &\leq q \cdot \left[\tfrac{1}{2}q^2 - q + \tfrac{1}{2}(q - c)(q - c - 1)\right] + c \cdot \left[\tfrac{1}{2}q^2 - q + \tfrac{1}{2}(q - c)(q - c - 1)\right] \\ &\leq (q^3 - \tfrac{3}{2}q^2) + \tfrac{1}{2}c^2 q - cq^2 + \tfrac{1}{2}cq + c \cdot \left[\tfrac{1}{2}q^2 - q + \tfrac{1}{2}(q - c)(q - c - 1)\right] \\ &\leq (q^3 - \tfrac{3}{2}q^2) + \tfrac{1}{2}c\left[c^2 - (q - 1)c - 2q\right] \\ &\leq k + \tfrac{1}{2}c\left[c^2 - (q - 1)c - 2q\right]. \end{aligned}$$

  Como $0 \leq c \leq q$, temos que $\frac{1}{2}c[c^2 - (q-1)c - 2q]$ é negativo e, consequentemente, $|L_B| \cdot |R_B| < k$.

- Suponha que $l \leq q$. Neste caso, temos que $|L \setminus L_B| = q + c$, onde $c = q - l$ e $0 \leq c \leq q$. Como H não contém nenhum clique com q vértices, temos que o subgrafo de H induzido por $L \setminus L_B$ possui menos do que $\frac{1}{2}(q+c)(q+c-1) - c$ arestas e, consequentemente, $r < \frac{1}{2}(q+c)(q+c-1) - c$. Portanto, conclui-se que

$$\begin{aligned} |L_B| \cdot |R_B| &= l \cdot (|W| + r) \\ &< (q - c) \cdot \left[\tfrac{1}{2}q^2 - q + \tfrac{1}{2}(q + c)(q + c - 1) - c\right] \\ &< (q^3 - \tfrac{3}{2}q^2) + \tfrac{1}{2}c^2(-c + 3 - q) \\ &< k + \tfrac{1}{2}c^2(-c + 3 - q). \end{aligned}$$

  Como $0 \leq c \leq q$ e podemos assumir que $q \geq 4$, temos que $\frac{1}{2}c^2(-c+3-q) \leq 0$ e, consequentemente, $|L_B| \cdot |R_B| < k$.

□

Além do algoritmo de Dawande *et al.* (1997) com razão de aproximação esperada igual a 2, para grafos aleatórios suficientemente densos, não encontramos na literatura nenhum algoritmo de aproximação para o problema de BICLIQUE MÁXIMO, e tampouco uma prova de que o problema é inaproximável. Nos artigos de Lanka e Goerdt (2004), Feige e Kogan (2004) e Ambühl *et al.* (2011), os autores apresentam a conjectura de que o problema de BICLIQUE MÁXIMO é inaproximável e mostram que, sob certas hipóteses, essa conjectura é verdadeira. Uma dessas hipóteses é de que não existe algoritmo com consumo de tempo subexponencial para o problema de 3-SATISFABILIDADE. No

artigo de Feige e Kogan (2004), é mostrado que o problema de BICLIQUE BALANCEADO MÁXIMO é inaproximável, e no artigo de Tan (2008), é mostrado que o problema de BICLIQUE DE PESO MÁXIMO é inaproximável.

Apesar de haver uma vasta literatura a respeito do problema de BICLIQUE MÁXIMO e algumas de suas variantes, não encontramos nenhum resultado sobre o problema de EMPACOTAMENTO DE BICLIQUES anterior ao nosso trabalho. Como o problema de EMPACOTAMENTO DE BICLIQUES tem como caso especial o problema de BICLIQUE MÁXIMO, seguem imediatamente os resultados de complexidade mencionados acima. Ou seja, o problema de EMPACOTAMENTO DE BICLIQUES é NP-difícil e, sob certas hipóteses, é inaproximável.

1.5 Resultados Obtidos Durante o Doutorado

No início do doutorado, quando ainda estávamos em fase de definição de qual seria o tema desta tese, nos dedicamos a concluir alguns trabalhos que havíamos começado durante nosso mestrado. Além de reorganizar o que havíamos feito durante o mestrado, resultando nos artigos Ferreira *et al.* (2011b) e Ferreira *et al.* (2011a) (este último artigo é uma extensão do primeiro e foi submetido em novembro de 2011, mas ainda não foi emitido o parecer dos revisores), conseguimos novos resultados nesse tema, resultando no artigo Freire *et al.* (2010). No artigo Freire *et al.* (2010), tivemos a colaboração de Roberto Cesar Marcondes Junior, do IME-USP.

Durante uma visita de alguns pesquisadores ao IME-USP, tivemos contato com Eduardo Moreno, da *Universidad Adolfo Ibanez*, e Vicente Acuña, da *Universitè de Lyon - INRIA*, ambos chilenos, que nos apresentaram uma aplicação em Bioinformática que eles tentavam modelar como um problema de otimização. Formalizamos o problema de EMPACOTAMENTO DE BICLIQUES juntamente com esses pesquisadores, que futuramente foram nossos coautores nos artigos Acuña *et al.* (2010) e Acuña *et al.* (2011). Através de E. Moreno, tivemos contato com um terceiro pesquisador chileno chamado Juan Pablo Vielma, da *University of Pittsburgh*. E. Moreno e J. P. Vielma foram nossos coautores no artigo Freire *et al.* (2011), no qual tratamos um problema de programação bilinear que é uma generalização do problema de BICLIQUE MÁXIMO.

No período de 12/2010 a 12/2011, o aluno A. S. Freire fez um estágio, conhecido como *doutorado sanduíche*, na *Universitè de Lyon - INRIA*, sob supervisão da diretora de pesquisa Marie-France Sagot. Durante esse período, trabalhamos em um problema com aplicação em análise de redes metabólicas, resultando no artigo Freire *et al.* (2012). Neste trabalho, tivemos a colaboração dos pesquisadores Pierluigi Crescenzi, da *Università degli Studi di Firenze*, Vincent Lacroix e Paulo Vieira Milreu, da *Universitè de Lyon - INRIA*, além de E. Moreno, V. Acuña e M.-F. Sagot. A ideia de fazer o doutorado sanduíche na *Universitè de Lyon - INRIA* nos pareceu interessante pelas seguintes razões. Na área de Bioinformática encontramos muitos problemas para os quais podemos aplicar técnicas de Otimização Combinatória, que é a nossa área de interesse. A pesquisadora M.-F. Sagot se formou no IME-USP e, por isso, possui um forte contato com alguns pesquisadores do IME-USP, já tendo trabalhando anteriormente com o orientador desse trabalho. Além disso, ela faz parte de um grupo de pesquisa multidisciplinar composto por importantes pesquisadores de diversos países, o que nos permitiu ampliar nossa rede de contatos e adquirir experiência em colaborar com pesquisadores de outras áreas.

Nas próximas seções, descreveremos em mais detalhes quais foram nossas contribuições nos

artigos mencionados acima.

1.5.1 Mapeamento de Grafos

Sejam $I = (V_I, E_I)$ e $M = (V_M, E_M)$ dois grafos conexos. Dizemos que uma função $\alpha : V_I \to V_M$ é um (I, M)-mapeamento. Para todo vértice v de V_M, denotamos por $\alpha^{-1}(v) = \{i \in V_I \mid \alpha(i) = v\}$ o conjunto dos vértices de V_I que estão associados a v em α. Dizemos que um (I, M)-mapeamento α é *viável* se para todo vértice v de V_M, temos que o subgrafo de I induzido por $\alpha^{-1}(v)$ é conexo, e para toda aresta uv de E_I, temos que $\{\alpha(u), \alpha(v)\} \in E_M$ ou $\alpha(u) = \alpha(v)$. A seguir, enunciamos o problema de MAPEAMENTO DE GRAFOS.

Problema 1.5.1. *Dados dois grafos conexos $I = (V_I, E_I)$ e $M = (V_M, E_M)$, e funções $c : V_I \times V_M \to \mathbb{R}$, $d : E_I \times E_M \to \mathbb{R}$ e $f : E_I \times V_M \to \mathbb{R}$, o problema de* MAPEAMENTO DE GRAFOS *consiste em encontrar um (I, M)-mapeamento α viável que minimize $w(\alpha)$, onde*

$$w(\alpha) = \sum_{u \in V_I} c(u, \alpha(u)) + \sum_{uv \in E_I : \{\alpha(u), \alpha(v)\} \in E_M} d(uv, \{\alpha(u), \alpha(v)\}) + \sum_{uv \in E_I : \alpha(u) = \alpha(v)} f(uv, \alpha(u)).$$

O problema de MAPEAMENTO DE GRAFOS é aplicado na resolução de problemas de reconhecimento de objetos em imagens. O grafo M representa um modelo no qual cada vértice corresponde a uma parte do objeto e as arestas correpondem a relações entre essas partes. O grafo I é construído de forma análoga, após algumas etapas de pré-processamento na imagem. Características de aparência das partes (por exemplo: cor e luminosidade) são armazenadas nos vértices e as relações entre partes diferentes (por exemplo: distância e orientação) são armazenadas nas arestas. Comparando essas caracteísticas, construímos as funções c, d e f para descrever a dissimilaridade entre o modelo e a imagem. O reconhecimento do objeto em questão consiste em identificar a qual parte do modelo devemos atribuir cada parte da imagem, sendo que tal tarefa é efetuada através da resolução do problema de MAPEAMENTO DE GRAFOS.

Nos artigos Ferreira *et al.* (2011b) e Ferreira *et al.* (2011a), tratamos o caso em que os grafos I e M são árvores. Mostramos que mesmo nesse caso o problema é NP-difícil, através de uma redução do problema de 3D-MATCHING, e introduzimos um algoritmo de Programação Dinâmica no qual a etapa de combinação das soluções ótimas de subproblemas é feita através da resolução de um Programa Linear Inteiro (PLI). Através de experimentos computacionais, mostramos que na prática a grande maioria das relaxações lineares de tais PLIs tinham soluções ótimas inteiras. Mostramos que a matriz de restrições de tal PLI é *totalmente unimodular* no caso em que a árvore I contém apenas um vértice com grau superior a 2, resultando em um algoritmo com consumo de tempo polinomial nos casos em que a quantidade de vértices de I com grau superior a 2 é $O(\lg n)$, onde n é o número total de vértices em I e M. No artigo Freire *et al.* (2010), introduzimos um algoritmo de *geração de colunas* para o caso geral do problema e apresentamos experimentos computacionais com instâncias geradas de forma a simular as características das instâncias reais da aplicação de reconhecimento de objetos.

1.5.2 Empacotamento de Bicliques

Nos artigos Acuña *et al.* (2010) e Acuña *et al.* (2011), introduzimos um algoritmo de *branch-and-price* para o problema de EMPACOTAMENTO DE BICLIQUES. O artigo Acuña *et al.* (2011) é

uma extensão de Acuña *et al.* (2010). Conforme mencionamos anteriormente, o problema de BICLIQUE MÁXIMO é um caso especial do problema de EMPACOTAMENTO DE BICLIQUES. Nossa contribuição com relação ao problema de BICLIQUE MÁXIMO foi introduzir duas formulações diferentes em Acuña *et al.* (2011) e em Freire *et al.* (2011), e um algoritmo de *branch-and-bound* que será apresentado na Seção 2.4. Em particular, a formulação introduzida por Freire *et al.* (2011) tem como objetivo resolver um problema mais geral, do qual o problema de BICLIQUE MÁXIMO é um caso especial. Nossas contribuições contidas nos artigos Acuña *et al.* (2010), Acuña *et al.* (2011) e Freire *et al.* (2011) estão relacionadas ao tema desta tese e, portanto, serão detalhadas nos demais capítulos.

1.5.3 Cobertura de Matrizes Por Raios Extremais

Dada uma matriz $A \in \mathbb{R}^{m \times n}$, seja $\mathcal{F} = \{v \in \mathbb{R}^n \mid Av = \mathbf{0},\ v \geq \mathbf{0}\}$. Dizemos que $\mathcal{F}$ é um *cone*, pois qualquer vetor obtido através de uma combinação cônica de vetores de $\mathcal{F}$ está em $\mathcal{F}$. Ou seja, dados vetores u e v em $\mathcal{F}$, temos que o vetor $x = \alpha u + \beta v$ está contido em $\mathcal{F}$, para quaisquer escalares α e β não-negativos. Dizemos que um vetor não-nulo contido em $\mathcal{F}$ é um *raio*. O *suporte* de um raio v, denotado por $sup(v)$, é o conjunto dos índices dos elementos não-nulos de v. Um raio v é *extremal* se seu suporte é minimal, no sentido de que não existe outro raio v' tal que $sup(v') \subset sup(v)$. Dizemos que uma coluna i de A é *coberta* por um raio extremal v se $v_i > 0$. Dado um conjunto $Q \subseteq \{1, 2, \ldots, n\}$ de colunas de A, dizemos que um subconjunto C de raios extremais é uma *cobertura* de Q se cada coluna de Q é coberta por pelo menos um raio extremal de C. Dado um raio v, seja $r(v) = \frac{\max_i v_i}{\min_{j \in sup(v)} v_j}$. Consideramos duas versões do problema de encontrar uma cobertura de Q, sendo que em cada uma delas uma certa função objetivo deve ser minimizada.

Problema 1.5.2. *Dada uma matriz $A \in \mathbb{R}^{m \times n}$ e um subconjunto de colunas $Q \subseteq \{1, 2, \ldots, n\}$ de A, o problema de* COBERTURA MÍNIMA AGREGADA *consiste em encontrar uma cobertura C de Q que minimize $\psi(C) = r(\sum_{v \in C} v)$.*

Problema 1.5.3. *Dada uma matriz $A \in \mathbb{R}^{m \times n}$ e um subconjunto de colunas $Q \subseteq \{1, 2, \ldots, n\}$ de A, o problema de* COBERTURA MÍNIMA DESAGREGADA *consiste em encontrar uma cobertura C de Q que minimize $\phi(C) = \max_{v \in C} r(v)$.*

Os Problemas 1.5.2 e 1.5.3 surgem naturalmente no contexto de análise de redes metabólicas, em particular no estudo dos *modos elementares* de uma rede metabólica. Cada coluna da matriz A representa uma reação química presente na rede metabólica e cada linha de A corresponde a um composto químico, sendo que $A_{ij} > 0$ se e só se a reação j produz A_{ij} unidades do composto i, e $A_{ij} < 0$ se e só se a reação j consome $-A_{ij}$ unidades do composto i. Os raios de $\mathcal{F}$ correspondem a *modos* da rede, que representam vias metabólicas que operam em *estado estacionário*, no qual todas as unidades dos compostos produzidos são consumidas. Os raios extremais de $\mathcal{F}$ correspondem aos *modos elementares* da rede, que são os objetos de interesse para a realização da análise biológica da mesma. Em geral, o número de modos elementares de uma rede metabólica é muito grande, sendo impraticável realizar a enumeração de todos eles. Conforme observado por biólogos, os modos elementares que ocorrem com maior frequência no metabolismo celular de alguns organismos são os mais *balanceados*, no sentido de que a discrepância, medida através da função $r : \mathbb{R}^n \to \mathbb{R}$ descrita acima, entre as reações que ocorrem com maior e menor frequência em uma via metabólica é pequena.

No artigo Freire *et al.* (2012), mostramos que os Problemas 1.5.2 e 1.5.3 são NP-difíceis, mesmo no caso em que $|Q| = 1$, através da redução do problema de 3D-MATCHING. Introduzimos formulações de Programação Linear Inteira para os Problemas 1.5.2 e 1.5.3. Em nossa formulação para o Problema 1.5.2, definimos $n - |Q|$ variáveis inteiras e, consequentemente, no caso em que Q contém todas as colunas de A, temos que o Problema 1.5.2 pode ser resolvido em tempo polinomial, através da resolução de um programa linear. Apresentamos experimentos computacionais feitos com instâncias vindas da aplicação. No caso do Problema 1.5.2, pudemos resolver instâncias com mais de 14 mil colunas e 2 mil linhas em poucos segundos. Nossa formulação para o Problema 1.5.3 contém uma quantidade exponencial de restrições, para as quais o respectivo *problema de separação* é NP-difícil. Introduzimos um algoritmo de *branch-and-cut* para o Problema 1.5.3 no qual o problema de separação é resolvido através de um Programa Linear Inteiro. Com nosso algoritmo de *branch-and-cut*, pudemos resolver instâncias com aproximadamente 1000 colunas e 500 linhas em alguns minutos. Pretendemos continuar nossa pesquisa nesse tema.

1.6 Objetivo Deste Trabalho

Nosso objetivo nessa tese é apresentar métodos de Otimização Combinatória para resolver os problemas de BICLIQUE MÁXIMO e EMPACOTAMENTO DE BICLIQUES. Como esses problemas são NP-difíceis, mantemos o foco no desempenho prático de nossos métodos, avaliando-os através de experimentos computacionais. A formulação de Programação Linear Inteira que apresentaremos para o problema de EMPACOTAMENTO DE BICLIQUES é resolvida utilizando a técnica de *branch-and-price*. Não há necessidade de que o leitor conheça previamente essa técnica, pois também temos como objetivo introduzir o leitor no assunto através desse exemplo. Na literatura são encontradas diversas descrições genéricas de como aplicar a técnica de *branch-and-price*. Optamos por não apresentar uma dessas descrições genéricas aqui por julgar tal descrição desnecessária para a compreensão de nosso trabalho. Recomendamos, no entando, que o leitor interessado em se aprofundar no assunto consulte os artigos de Barnhart *et al.* (1998) e Wilhelm (2001).

1.7 Organização do Texto

O restante do texto está organizado da seguinte forma. No Capítulo 2, apresentamos três métodos diferentes para o problema de BICLIQUE MÁXIMO e mostramos como esse problema está relacionado com o problema de CLIQUE MÁXIMO. No Capítulo 3, apresentamos duas formulações para o problema de EMPACOTAMENMTO DE BICLIQUES. Argumentamos que a primeira formulação não é satisfatória do ponto de vista prático, devido ao problema de simetria que ela possui. Em seguida, mostramos que a técnica de *branch-and-price* pode ser aplicada neste caso para eliminar a simetria da formulação anterior. No Capítulo 4, apresentamos um método para resolver um problema de Programação Bilinear Inteira que possui como caso particular o problema de BICLIQUE MÁXIMO. Por fim, no Capítulo 5, concluímos nosso trabalho e mencionamos possíveis trabalhos futuros.

Capítulo 2

O Problema de Biclique Máximo

Neste capítulo, apresentamos nossos resultados para o problema de BICLIQUE MÁXIMO. Na Seção 2.1, mostramos como transformar uma instância do problema de BICLIQUE MÁXIMO em uma instância do problema de CLIQUE MÁXIMO. Nas Seções 2.2 e 2.3, apresentamos dois métodos de Programação Linear Inteira. Na Seção 2.4, apresentamos um algoritmo de *branch-and-bound* e, na Seção 2.5, apresentamos uma heurística. Na Seção 2.7, apresentamos resultados de experimentos computacionais comparando todas as abordagens apresentadas e, na Seção 2.8, encerramos este capítulo fazendo algumas considerações finais sobre o conteúdo apresentado.

2.1 Relação Com o Problema de Clique Máximo

Seja $G = (V, E)$ um grafo bipartido. Dizemos que duas arestas uv e pq de E são *incompatíveis* se $u \neq p$, $v \neq q$ e $uq \notin E$ ou $pv \notin E$. Observe que duas arestas uv e pq de E são incompatíveis se e só se não existe biclique em G que contém uv e pq. Seja $\mathcal{I} = \{\{uv, pq\} \in E \times E \mid uv \text{ e } pq \text{ são incompatíveis}\}$ o conjunto de todos os pares de arestas incompatíveis de E. Definimos um vetor de variáveis de decisão $x \in \{0,1\}^{|E|}$ que representará o *vetor característico* de um subconjunto de arestas E' de E. Ou seja, $x_{uv} = 1$ se e só se $uv \in E'$.

A seguir, apresentamos uma formulação para o problema de BICLIQUE MÁXIMO que é uma adaptação natural da formulação introduzida por Dawande *et al.* (1997).

$$
\begin{array}{lrcll}
 & \text{maximize} & \displaystyle\sum_{uv \in E} x_{uv} & & & \\
(\text{PI}_{\text{BM}}) & \text{sujeito a} & x_{uv} + x_{pq} & \leq & 1, \quad \text{para todo } \{uv, pq\} \in \mathcal{I} & (1.1) \\
 & & x_{uv} & \in & \{0,1\}^{|E|} & (1.2)
\end{array}
$$

A partir de uma solução viável x de (PI_{BM}), construímos um subgrafo B de G, tal que $uv \in E_B$ se e só se $x_{uv} = 1$. Observe que o vetor característico das arestas de qualquer biclique B em G corresponde a uma solução viável de (PI_{BM}) com valor $|E_B|$ na função objetivo. Reciprocamente, qualquer solução x viável de (PI_{BM}) corresponde ao vetor característico de um subconjunto de arestas $E' \subseteq E_B$ que está contido em algum biclique B em G. Portanto, (PI_{BM}) é de fato uma formulação para o problema de BICLIQUE MÁXIMO.

Originalmente, a formulação introduzida por Dawande *et al.* (1997) para o problema de BICLIQUE MÁXIMO utiliza a mesma ideia de (PI_{BM}), porém em vez de maximizar a quantidade de arestas que estão contidas no biclique, minimiza a quantidade de arestas que não estão incluídas no

biclique, conforme apresentamos a seguir.

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{uv\in E} \overline{x}_{uv} & \\
\text{sujeito a} & \overline{x}_{uv} + \overline{x}_{pq} \geq 1, & \text{para todo } \{uv, pq\} \in \mathcal{I} \quad (2.1) \\
& \overline{x}_{uv} \in \{0,1\}^{|E|} & (2.2)
\end{array}
\qquad (\text{PI2}_{\text{BM}})
$$

Para cada solução viável x de (PI_{BM}), existe uma solução viável $\overline{x}$ de (PI2_{BM}), tal que $\overline{x}_{uv} = 1 - x_{uv}$, para toda aresta uv de E. Dizemos que x e $\overline{x}$ são soluções *equivalentes*. Sejam x e $\overline{x}$ soluções equivalentes de (PI_{BM}) e (PI2_{BM}), respectivamente. Observe que $\sum_{uv\in E} \overline{x}_{uv} = |E| - \sum_{uv\in E} x_{uv}$ e, consequentemente, x é uma solução ótima de (PI_{BM}) se e somente se $\overline{x}$ é uma solução ótima de (PI2_{BM}).

Podemos considerar que as formulações (PI_{BM}) e (PI2_{BM}) são equivalentes, no sentido de que uma solução ótima de uma pode ser obtida através de uma solução ótima da outra. Porém, do ponto de vista de aproximação, não podemos dizer o mesmo. Denotamos por (PL_{BM}) e (PL2_{BM}) as relaxações lineares de (PI_{BM}) e (PI2_{BM}), respectivamente, onde a restrição de integralidade é substituída por $0 \leq x_{uv}, \overline{x}_{uv} \leq 1$, para todo $uv \in E$. Seja $\overline{x}^*$ uma solução ótima de (PL2_{BM}) e seja $\overline{x}^{**} \in \{0,1\}^{|E|}$ um vetor, tal que $\overline{x}^{**} = 1$ se $\overline{x}^* \geq \frac{1}{2}$, caso contrário $\overline{x}^{**} = 0$. Considerando (PL2_{BM}), em toda restrição (2.1), pelo menos uma das variáveis tem valor maior ou igual a $\frac{1}{2}$. Portanto, esse procedimento de arredondamento garante que todas restrições (2.1) são satisfeitas por $\overline{x}^{**}$ e, consequentemente, o vetor $\overline{x}^{**}$ é uma solução viável de (PI2_{BM}). Além disso, esse procedimento de arredondamento resulta em uma 2-aproximação para o problema de encontrar uma solução ótima de (PI2_{BM}), já que $\sum_{uv\in E} \overline{x}^{**}_{uv} \leq 2 \cdot \sum_{uv\in E} \overline{x}^{*}_{uv}$. Por outro lado, não existe um procedimento análogo que garanta uma boa aproximação para o problema de encontrar uma solução ótima de (PI_{BM}). Essa observação segue do fato de que $x_{uv} = \frac{1}{2}$, para todo $uv \in E$, é uma solução viável de (PL_{BM}) e, consequentemente, para qualquer grafo bipartido G, o valor de uma solução ótima de (PL_{BM}) é pelo menos $\frac{|E|}{2}$. Considere um grafo bipartido $G = (V, E)$ composto por apenas um circuito par, tal que $|E| \geq 6$. Nesse caso, temos que uma solução ótima de (PI_{BM}) tem valor 1 e uma solução de (PL_{BM}) tem valor pelo menos $\frac{|E|}{2}$. Logo, a razão de aproximação desse procedimento de arredondamento é no máximo $\frac{2}{|E|}$.

Seja H o grafo em que, para cada aresta uv de E, existe um vértice de rótulo $l(uv)$ em $V(H)$, e $\{l(uv), l(pq)\} \in A(H)$ se e só se $\{uv, pq\} \in \mathcal{I}$. Chamaremos H de o *grafo de incompatibilidade* de G. Observe que, encontrar uma solução ótima de (PI_{BM}) é equivalente a encontrar um conjunto independente máximo de vértices, e encontrar uma solução ótima de (PI2_{BM}) é equivalente a encontrar uma cobertura mínima de arestas por vértices. De fato, Berman e Schnitger (1992) mostram que o problema de CONJUNTO INDEPENDENTE MÁXIMO é inaproximável, enquanto que um procedimento análogo ao arredondamento descrito acima resulta em uma 2-aproximação para o problema de COBERTURA MÍNIMA DE ARESTAS POR VÉRTICES. Observe que, resolver o problema de CONJUNTO INDEPENDENTE MÁXIMO no grafo H é equivalente a resolver o problema de CLIQUE MÁXIMO no grafo $\overline{H}$, onde $\overline{H}$ é o complemento de H (ou seja, $V(\overline{H}) = V(H)$ e $e \in A(\overline{H})$ se e só se $e \notin A(H)$).

Conforme mencionaremos na Seção 2.7, as formulações (PI_{BM}) e (PI2_{BM}) não são de interesse prático, pois o tempo de processamento necessário para resolver suas respectivas relaxações lineares

é bastante superior ao tempo de processamento necessário para resolver o problema de BICLIQUE MÁXIMO utilizando qualquer uma das outras abordagens que apresentaremos nesse capítulo. Mesmo assim, o grafo de incompatibilidade de G é bastante útil, pois existem algoritmos de *branch-and-bound* que resolvem instâncias grandes do problema de CLIQUE MÁXIMO em poucos segundos. Alguns exemplos são encontrados em Östergård (2002), Tomita e Seki (2003) e Konc e Janezic (2007). Na Seção 2.7, mostramos experimentos computacionais que apontam as vantagens e as limitações da abordagem de resolver o problema de BICLIQUE MÁXIMO através da resolução do problema de CLIQUE MÁXIMO no grafo $\overline{H}$.

Para resolver o problema de BICLIQUE DE PESO MÁXIMO utilizando esta mesma abordagem, basta utilizar algoritmos de *branch-and-bound* para o problema de encontrar um clique de peso máximo, como por exemplo o algoritmo introduzido por Warren e Hicks (2007). Nesse caso, o peso de um vértice de rótulo $l(uv)$ de $V(\overline{H})$ é dado pelo peso $w(uv)$ da aresta $uv \in E$.

2.2 Uma Formulação Com Menos Variáveis e Restrições

As formulações apresentadas nesta seção foram concebidas a fim de resolver o problema de BICLIQUE MÁXIMO. No entanto, apresentamos versões adaptadas para resolver o problema de BICLIQUE DE PESO MÁXIMO. Para obter as versões originais dessas formulações, basta considerar que a função de pesos $w : E \to \mathbb{R}$ é definida de forma que $w(uv) = 1$, para toda aresta $uv \in E$.

O que torna ineficiente o uso das formulações (PI_{BM}) e (PI2_{BM}) é a grande quantidade de variáveis e restrições. A seguir, apresentamos formulações que contornam esse problema. Os resultados apresentados nesta seção consistem em uma explanação mais detalhada do que foi publicado por Acuña *et al.* (2011). Sem perda de generalidade, assumimos que $|L| \leq |R|$. Definimos um vetor de variáveis de decisão $z \in \{0,1\}^{|V|}$ para representar o vetor característico dos vértices de um biclique máximo que desejamos encontrar. Definimos também um vetor de variáveis $y \in R_+$, tal que para todo $u \in L$, a variável y_u representa o peso total das arestas incidentes em u que fazem parte do biclique induzido por z. Seja $W_u = \left(\sum_{v \in N(u)} w(uv)\right)$ o peso total das arestas que incidem em u, para todo vértice $u \in V$. Abaixo, apresentamos uma formulação para o problema de BICLIQUE DE PESO MÁXIMO.

$$
\begin{array}{lrcll}
 & \text{maximize} & \displaystyle\sum_{u \in L} y_u & & & \\
(\text{PI3}_{\text{BM}}) & \text{sujeito a} & z_u + z_v & \leq & 1, \quad \text{para cada } u \in L \text{ e } v \in R \setminus N(u) & (3.1) \\
 & & y_u & \leq & W_u \cdot z_u, \quad \text{para cada } u \in L & (3.2) \\
 & & y_u & \leq & \displaystyle\sum_{v \in N(u)} w(uv) \cdot z_v, \quad \text{para cada } u \in L & (3.3) \\
 & & y_u & \geq & 0, \quad \text{para cada } u \in L & (3.4) \\
 & & z & \in & \{0,1\}^{|V|} & (3.5)
\end{array}
$$

Lema 2.2.1. (PI3_{BM}) *é uma formulação para o problema de* BICLIQUE DE PESO MÁXIMO.

Demonstração. Observe que, para cada biclique B em G, podemos construir uma solução (z, y) viável de (PI3_{BM}), tal que $z_u = 1$ se e só se $u \in V_B$, e se $u \in L_B$ então $y_u = \sum_{v \in R_B} w(uv)$, caso contrário $y_u = 0$. O valor de tal solução na função objetivo é $W(B)$. Resta mostrar que qualquer

solução ótima de (PI3_{BM}) corresponde a um biclique em G.

Seja (z^*, y^*) uma solução ótima de (PI3_{BM}) e seja B^* o subgrafo de G induzido por $V_{B^*} \subseteq V$, onde $u \in V_{B^*}$ se e só se $z^*_u = 1$. Pelas restrições (3.1), temos que não existe par de vértices $u \in L_{B^*}$ e $v \in R_{B^*}$, tal que $uv \notin E_{B^*}$. Logo, B^* é um biclique. Pelas restrições (3.2), temos que $y^*_u > 0$ implica em $z^*_u = 1$. Logo, $y^*_u = 0$, para todo $u \in L \setminus L_{R^*}$. Pelas restrições (3.3) e considerando a função objetivo, temos que $y^*_u = \sum_{v \in R_{B^*}} w(uv)$, para cada $u \in L_{B^*}$. Consequentemente, temos que $\sum_{u \in L} y^*_v = W(B^*)$. Portanto, ($\text{PI3}_{\text{BM}}$) é uma formulação para o problema de BICLIQUE DE PESO MÁXIMO. □

Observe que, como há restrição de integralidade nas variáveis z, então em uma solução ótima de (PI3_{BM}) temos que y é inteiro também, mesmo sem exigirmos explicitamente esta condição através de uma restrição de integralidade.

Nas formulações (PI_{BM}) e (PI2_{BM}) são definidas $|E|$ variáveis e $|\mathcal{I}| = O(|E|^2)$ restrições, enquanto que na formulação (PI3_{BM}) são definidas $|V| + |L| = O(|V|)$ variáveis e $(|L| \cdot |R| - |E| + 2 \cdot |L|) = O(|L| \cdot |R|)$ restrições. É possível fazer pequenas modificações em (PI3_{BM}) de forma a reduzir o número de variáveis inteiras de $|V|$ para $|L|$. Para isso, considere a seguinte observação.

Observação 2.2.1. *Dado um subconjunto de vértices $L' \subseteq L$, sejam $L_B = L'$ e $R_B = \{v \in R \mid L' \subseteq N(v)\}$. Temos que $B = (L_B, R_B)$ é um biclique, tal que $W(B) \geq W(B')$, para todo biclique B' tal que $L_{B'} = L'$.*

Como estamos interessados em encontrar um biclique B^* de peso máximo em G, da Observação 2.2.1 segue que, uma vez decidido qual é o subconjunto de vértices em L_{B^*}, os vértices de R_{B^*} são determinados de forma simples e unívoca. A seguir, apresentamos uma formulação para o problema de BICLIQUE DE PESO MÁXIMO que utiliza esta ideia.

$$
\begin{array}{lrcll}
 & \text{maximize} & \displaystyle\sum_{v \in R} y_v & & \\
(\text{PI4}_{\text{BM}}) & \text{sujeito a} \quad y_v & \leq & \displaystyle\sum_{u \in N(v)} w(uv) \cdot z_u, & \text{para cada } v \in R \qquad (4.1) \\
 & y_v & \leq & W_v \cdot (1 - z_u), & \text{para cada } u \in L \text{ e } v \in R \setminus N(u) \qquad (4.2) \\
 & y_v & \geq & 0, & \text{para cada } v \in R \qquad (4.3) \\
 & z & \in & \{0,1\}^{|L|} & \qquad (4.4)
\end{array}
$$

Na formulação (PI4_{BM}), as variáveis z e y são interpretadas da mesma forma que na formulação (PI3_{BM}). Porém, na formulação (PI4_{BM}) as variáveis y são definidas para os vértices em R em vez de L, e as variáveis z são definidas para os vértices em L em vez de V. Ou seja, em (PI4_{BM}) o número total de variáveis é menor e, além disso, o número total de variáveis inteiras é reduzida pelo menos pela metade.

Dada uma solução ótima (z^*, y^*) de (PI4_{BM}), sejam $L_{B^*} = \{u \in L \mid z^*_u = 1\}$ e $R_{B^*} = \{v \in R \mid y^*_v > 0\}$. Pelos mesmos argumentos usados na prova do Lema 2.2.1, temos que B^* é um biclique de peso máximo em G. Seja $\hat{W}_u = \sum_{v \in R \setminus N(u)} W_v$ o peso total das arestas que incidem nos vértices de R que não são adjacentes a u, para todo vértice $u \in L$. Agregando as restrições (4.2), podemos reduzir o número de restrições de (PI4_{BM}) de $O(|L| \cdot |R|)$ para $O(|V|)$, conforme mostramos na formulação abaixo.

$$
\begin{aligned}
& & \text{maximize} \quad & \sum_{v \in R} y_v & & & & \\
(\text{PI5}_{\text{BM}}) \qquad & & \text{sujeito a} \quad & y_v & \leq & \sum_{u \in N(v)} w(uv) \cdot z_u, & \text{para cada } v \in R & \quad (5.1) \\
& & & \sum_{v \in R \setminus N(u)} y_v & \leq & \hat{W}_u \cdot (1 - z_u), & \text{para cada } u \in L & \quad (5.2) \\
& & & y_v & \geq & 0, & \text{para cada } v \in R & \quad (5.3) \\
& & & z & \in & \{0,1\}^{|L|} & & \quad (5.4)
\end{aligned}
$$

É importante notar que quando G é bastante desbalanceado, a quantidade de variáveis inteiras das formulações (PI4_{BM}) e (PI5_{BM}) é muito menor do que a quantidade de variáveis inteiras das demais formulações apresentadas nesta seção. Considerando o número de restrições multiplicado pelo número de variáveis, (PI5_{BM}) é a menor formulação dentre todas as apresentadas nesta seção. No total, a formulação (PI5_{BM}) possui $|L|$ variáveis inteiras, $|R|$ variáveis fracionárias e $|V|$ restrições.

Sejam (PL4_{BM}) e (PL5_{BM}) as relaxação lineares de (PI4_{BM}) e (PI5_{BM}), respectivamente. Apesar de (PL5_{BM}) possuir menos restrições do que (PL4_{BM}), a formulação (PL4_{BM}) é mais forte do que (PL5_{BM}), no sentido de que o espaço de soluções viáveis de (PL5_{BM}) contém propriamente o espaço de soluções viáveis de (PL4_{BM}). Em outras palavras, toda solução viável de (PL4_{BM}) é também uma solução viável de (PL5_{BM}), porém a recíproca não é verdadeira. Consequentemente, o limitante superior dado por uma solução ótima de (PL4_{BM}) pode ser menor (ou seja, melhor) do que o limitante obtido com (PL5_{BM}).

Estamos interessados em resolver instâncias do problema de BICLIQUE DE PESO MÁXIMO utilizando resolvedores de Programção Linear Inteira, como por exemplo o *CPLEX* e o *GUROBI*. O tempo de processamento necessário para encontrar uma solução ótima de um Programa Linear Inteiro depende de outros fatores, além da quantidade de restrições e variáveis presentes na formulação. Alguns desses fatores são, por exemplo, a densidade da matriz de restrições e a qualidade dos limitantes obtidos através das relaxações lineares. Além disso, como cada resolvedor tem suas implementações particulares, a eficiência de cada formulação pode oscilar bastante entre diferentes resolvedores. Na Seção 2.7, apresentamos experimentos computacionais comparando as formulações (PI3_{BM}), (PI4_{BM}) e (PI5_{BM}).

2.3 Um Método de Relaxação da Função Objetivo

Nesta seção apresentamos um método para resolver o problema de BICLIQUE MÁXIMO que é uma simplificação de um método introduzido por Freire *et al.* (2011) para resolver um problema de Programação Bilinear Inteira. Ao contrário do que apresentamos na seção anterior, não pudemos encontrar uma forma de adaptar este método para resolver o problema de BICLIQUE DE PESO MÁXIMO.

Primeiramente, considere o seguinte problema:

Problema 2.3.1. *Dados um grafo bipartido G e um inteiro positivo γ, encontrar um biclique B em G, tal que $|L_B| \geq \gamma$, $|R_B| \geq \gamma$ e $|V_B|$ seja máximo.*

Assuma que temos um procedimento, digamos RESOLVE-PROB-2.3.1, que recebe uma instância

(G, γ) do Problema 2.3.1 e devolve uma solução ótima B ou NULL, caso não exista solução viável para tal instância. Apresentamos a seguir um algoritmo que, utilizando a sub-rotina RESOLVE-PROB-2.3.1, resolve o problema de BICLIQUE MÁXIMO.

Algoritmo 1: MRFO(G)

Entrada: um grafo bipartido G
Saída: um biclique B^* em G tal que $|E_{B^*}|$ é máximo

```
1  γ ← 1, B* ← (∅, ∅)
2  enquanto VERDADEIRO faça
3  |  B ← RESOLVE-PROB-2.3.1(G, γ)
4  |  se B = NULL então
5  |  |  devolva B*
6  |  fim
7  |  se |E_B| > |E_B*| então
8  |  |  B* ← B
9  |  fim
10 |  γ ← min{|L_B|, |R_B|} + 1
11 fim
```

Antes de mostrar que o algoritmo MRFO está correto, provamos o seguinte lema.

Lema 2.3.1. *Sejam a_1, a_2, b_1, b_2 números positivos. Se $a_1 + a_2 \geq b_1 + b_2$ e $a_1 \cdot a_2 < b_1 \cdot b_2$, então* $\min\{a_1, a_2\} < \min\{b_1, b_2\}$.

Demonstração. Sem perda de generalidade, suponha que $a_1 = \min\{a_1, a_2\}$ e $b_1 = \min\{b_1, b_2\}$. Considere a função $f(k, x) = kx - x^2$, onde k e x são dois números reais. Observe que, para qualquer número positivo k fixo, a função $f(k, x)$ é estritamente crescente, para x no intervalo $[0, \frac{k}{2}]$. Como $a_1 + a_2 \geq b_1 + b_2$ e $[0, b_1] \subseteq [0, \frac{b_1+b_2}{2}]$ então $f(a_1 + a_2, x) \geq f(b_1 + b_2, x)$, para todo x no intervalo $[0, b_1]$. Como $f(a_1 + a_2, x)$ é estritamente crescente, para x no intervalo $[0, a_1]$, e $f(a_1 + a_2, a_1) = a_1 a_2 < b_1 b_2 = f(b_1 + b_2, b_1)$, concluímos que $a_1 < b_1$. □

Lema 2.3.2. *O algoritmo* MRFO *está correto.*

Demonstração. Considere a propriedade invariante (i_1) que enunciamos a seguir: $|E_{B^*}| \geq |E_B|$, *para todo biclique B em G, tal que* $\min\{|L_B|, |R_B|\} < \gamma$. Mostraremos que (i_1) é válida no inicio de cada execução da linha 2. Na primeira iteração, temos que $|E_{B^*}| = 0$ e $\gamma = 1$, logo (i_1) é satisfeita. Dada uma iteração qualquer em que $\gamma > 1$ e (i_1) é satisfeita, mostraremos que na iteração seguinte (i_1) se manterá satisfeita. Considere o instante imediatamente antes da execução da linha 10. Seja B o biclique obtido na linha 3. É suficiente mostrar que $|E_B| \geq |E_{B'}|$, para todo biclique B' em G tal que $\gamma \leq \min\{|L_{B'}|, |R_{B'}|\} < \min\{|L_B|, |R_B|\} + 1$. Suponha, por absurdo, que existe um biclique B' em G tal que $|E_{B'}| > |E_B|$ e $\gamma \leq \min\{|L_{B'}|, |R_{B'}|\} < \min\{|L_B|, |R_B|\} + 1$. Usando o Lema 2.3.1 com $a_1 = |L_B|$, $a_2 = |R_B|$, $b_1 = |L_{B'}|$ e $b_2 = |R_{B'}|$, temos que $\min\{|L_B|, |R_B|\} < \min\{|L_{B'}|, |R_{B'}|\}$, o que é uma contradição.

Como a cada iteração a variável γ é incrementada em pelo menos uma unidade, temos que o algoritmo para em no máximo $|L_B|$ iterações. Pela condição de parada e pela propriedade (i_1), segue que o algoritmo está correto. □

Do funcionamento do algoritmo MRFO podemos concluir o seguinte fato sobre o Problema 2.3.1:

Lema 2.3.3. *O Problema 2.3.1 é* NP-difícil.

Demonstração. Conforme argumentamos na prova do Lema 2.3.2, o algoritmo MRFO para em no máximo $|L_B|$ iterações. Logo, se o Problema 2.3.1 pudesse ser resolvido em tempo polinomial, teríamos um algoritmo polinomial para o problem de BICLIQUE MÁXIMO, que é NP-difícil. Portanto, o Problema 2.3.1 é NP-difícil. □

A seguir, apresentamos uma formulação para o Problema 2.3.1.

$$
\begin{array}{lrcll}
 & \text{maximize} & \sum_{u\in V} z_u & & & \\
(\mathrm{PI}_{2.3.1}) & \text{sujeito a} & z_u + z_v & \leq & 1, \quad \text{para cada } u \in L \text{ e } v \in R \setminus N(u) & (6.1)\\
 & & \sum_{u\in L} z_u & \geq & \gamma & (6.2)\\
 & & \sum_{v\in R} z_v & \geq & \gamma & (6.3)\\
 & & z & \in & \{0,1\}^{|V|} & (6.4)
\end{array}
$$

O vetor de variáveis binárias z corresponde ao vetor característico dos vértices do biclique que desejamos encontrar. É fácil ver que a formulação ($\mathrm{PI}_{2.3.1}$) pode ser usada para resolver o Problema 2.3.1, conforme descrevemos a seguir.

Observação 2.3.1. *Dada uma solução ótima z^* de* ($\mathrm{PI}_{2.3.1}$)*, seja B^* o subgrafo de G induzido por $V_{B^*} = \{v \in V \mid z_v = 1\}$. Temos que B^* é um biclique em G, tal que $|L_{B^*}| \geq \gamma$, $|R_{B^*}| \geq \gamma$ e $|V_{B^*}|$ é máximo.*

Observe que podem ocorrer diversas iterações do laço que começa na linha 2 do Algoritmo 1 em que a variável B^* não é atualizada. Uma ideia que surge naturalmente desta observação é adaptar a formulação ($\mathrm{PI}_{2.3.1}$) para que a cada passo a variável B^* seja atualizada, evitando as iterações "infrutíferas". Formalmente, estamos interessados em resolver o seguinte problema.

Problema 2.3.2. *Dados um grafo bipartido G e um biclique B^*, tal que $|E_{B^*}| \geq |E_B|$, para todo biclique B em G, tal que $\min\{|L_B|, |R_B|\} < \gamma$, encontrar um biclique B^{**} em G, tal que $|E_{B^{**}}| > |E_{B^*}|$ e $|V_{B^{**}}|$ seja máximo.*

Observe que, de acordo com o Lema 2.3.1, a condição de que $\min\{|L_{B^{**}}|, |R_{B^{**}}|\} \geq \gamma$ está implícita no enunciado do Problema 2.3.2. No lema a seguir, mostramos como garantir que $|E_{B^{**}}| > |E_{B^*}|$ seja satisfeito, impondo um limitante inferior em $|V_{B^{**}}|$ em função de $\min\{|L_{B^{**}}|, |R_{B^{**}}|\}$.

Lema 2.3.4. *Seja B^* um biclique em G, tal que $|E_{B^*}| \geq |E_B|$, para todo biclique B em G, tal que $\min\{|L_B|, |R_B|\} < \gamma$. Dado um biclique B^{**} em G, tal que $\min\{|L_{B^{**}}|, |R_{B^{**}}|\} = \gamma + i$, temos que $|E_{B^{**}}| > |E_{B^*}|$ se e só se $|V_{B^{**}}| \geq \beta(\gamma, i)$, onde i é um inteiro não negativo e $\beta(\gamma, i) = \left\lfloor \frac{|E_{B^*}|}{\gamma + i} \right\rfloor + \gamma + i + 1$.*

Demonstração. Sem perda de generalidade, suponha que $|L_{B^*}| \leq |R_{B^*}|$. Supondo que $|E_{B^{**}}| > |E_{B^*}|$, temos que

$$(\gamma + i) \cdot |R_{B^{**}}| > |E_{B^*}|$$

$$|R_{B^{**}}| > \frac{|E_{B^*}|}{\gamma + i}$$

$$|L_{B^{**}}| + |R_{B^{**}}| > \frac{|E_{B^*}|}{\gamma + i} + \gamma + i$$

$$|L_{B^{**}}| + |R_{B^{**}}| \geq \left\lfloor \frac{|E_{B^*}|}{\gamma + i} \right\rfloor + \gamma + i + 1$$

$$|V_{B^{**}}| \geq \beta(\gamma, i).$$

Reciprocamente, supondo que $|V_{B^{**}}| \geq \beta(\gamma, i)$, temos que

$$|L_{B^{**}}| + |R_{B^{**}}| \geq \left\lfloor \frac{|E_{B^*}|}{\gamma + i} \right\rfloor + \gamma + i + 1$$

$$|R_{B^{**}}| \geq \left\lfloor \frac{|E_{B^*}|}{\gamma + i} \right\rfloor + 1$$

$$|R_{B^{**}}| > \frac{|E_{B^*}|}{\gamma + i}$$

$$(\gamma + i) \cdot |R_{B^{**}}| > |E_{B^*}|$$

$$|E_{B^{**}}| > |E_{B^*}|.$$

□

Introduzimos um vetor de variáveis binárias $s \in \{0,1\}^{|L_B|-\gamma+1}$ com a interpretação de que $s_i = 1$ implica em $\min\{|L_{B^{**}}|, |R_{B^{**}}|\} \geq \gamma + i$. A seguir, apresentamos uma formulação para resolver o Problema 2.3.2.

$$
\begin{aligned}
& \text{maximize} && \sum_{u \in V} z_u && && \\
(\text{PI}_{2.3.2}) \quad & \text{sujeito a} && z_u + z_v \leq 1, && \forall u \in L \text{ e } v \in R \setminus N(u) && (7.1)\\
& && \sum_{i=0}^{|L|-\gamma} s_i = 1 && && (7.2)\\
& && \sum_{u \in L} z_u \geq \sum_{i=0}^{|L|-\gamma} (\gamma + i) \cdot s_i && && (7.3)\\
& && \sum_{v \in R} z_v \geq \sum_{i=0}^{|L|-\gamma} (\gamma + i) \cdot s_i && && (7.4)\\
& && \sum_{u \in L} z_u + \sum_{v \in R} z_v \geq \sum_{i=0}^{|L|-\gamma} \beta(\gamma, i) \cdot s_i && && (7.5)\\
& && s \in \{0,1\}^{|L|-\gamma+1} && && (7.6)\\
& && z \in \{0,1\}^{|V|} && && (7.7)
\end{aligned}
$$

Lema 2.3.5. ($\text{PI}_{2.3.2}$) *é uma formulação para o Problema 2.3.2.*

Demonstração. Podemos obter uma solução para uma instância do Problema 2.3.2 a partir de uma

solução ótima (z^*, s^*) de $(\text{PI}_{2.3.2})$ da seguinte maneira. Seja B^{**} o subgrafo de G induzido por $V_{B^{**}} = \{v \in V \mid z_v^* = 1\}$. A restrição (7.1) garante que B^{**} é um biclique. A restrição (7.2) garante que $s_i^* = 1$ para exatamente um índice $i \in \{0, 1, \ldots, |L_B| - \gamma\}$. As restrições (7.3) e (7.4) garantem que se $s_i^* = 1$ então $\min\{|L_{B^{**}}|, |R_{B^{**}}|\} \geq \gamma + i$. As restrições (7.5) garantem que a condição do Lema 2.3.4 é satisfeita. Ou seja, se $\min\{|L_{B^{**}}|, |R_{B^{**}}|\} = \gamma + i$ então $|V_{B^{**}}| \geq \beta(\gamma, i)$, o que implica em $|E_{B^{**}}| > |E_{B^*}|$, onde B^* é um biclique em G, tal que $|E_{B^*}| \geq |E_B|$, para todo biclique B em G, tal que $\min\{|L_B|, |L_B|\} < \gamma$. Portanto, $(\text{PI}_{2.3.2})$ é uma formulação para o Problema 2.3.2. □

Na Seção 2.7, apresentamos experimentos computacionais comparando o desempenho do método de relaxação da função objetivo utilizando as formulações $(\text{PI}_{2.3.1})$ e $(\text{PI}_{2.3.2})$.

2.4 Um Algoritmo de *Branch-and-Bound*

Nesta seção, apresentamos um algoritmo de *branch-and-bound* para o problema de BICLIQUE MÁXIMO. No final da seção, mostramos como adaptar nosso algoritmo para resolver o problema de BICLIQUE DE PESO MÁXIMO.

Considere uma ordem arbitrária $(u_1, u_2, \ldots, u_{|L|})$ dos vértices de L. Dados um subconjunto $L' \subseteq L$ e um inteiro $j \leq |L|$, seja $\mathcal{B}(L', j) = \{B \text{ é um biclique em } G \mid \{u_1, u_2, \ldots, u_j\} \cap L_B = L'\}$. Para cada subconjunto L' de L, consideramos o biclique B', onde $L_{B'} = L'$ e $R_{B'} = N^{\cap}(L_{B'})$. Note que $R_{B''} \subset R_{B'}$, para qualquer biclique B'' em G, tal que $B'' \neq B'$ e $L_{B''} = L_{B'}$. Portanto, dentre todos os bicliques em G que possuem $L_{B'}$ como sendo uma de suas classes da bipartição, B' é o que possui mais arestas, conforme Observação 2.2.1. Nosso algoritmo de *branch-and-bound* explora o espaço de busca que corresponde a todos os possíveis subconjuntos de L. A seguir, mostramos uma forma de obter um limitante superior para a quantidade de arestas que pode conter um biclique contido em um determinado subconjunto $\mathcal{B}(L', j)$.

Lema 2.4.1. *Dados um subconjunto $L' \subseteq L$ e um inteiro j, tal que $|L'| \leq j \leq |L|$, seja $R' = N^{\cap}(L')$ e, sem perda de generalidade, assuma que $|N(u_l) \cap R'| \leq |N(u_{l+1}) \cap R'|$, para $l = j+1, \ldots, |L|-1$. O valor de*

$$LimSup = \max\left\{|L'| \cdot |R'|, \max_{j+1 \leq l \leq |L|} f(l)\right\}, \text{ onde } f(l) = |N(u_l) \cap R'| \cdot (|L'| + |L| - l + 1),$$

é um limitante superior para a quantidade de arestas de qualquer biclique contido em $\mathcal{B}(L', j)$.

Demonstração. Seja B^* um biclique de $\mathcal{B}(L', j)$ com quantidade máxima de arestas. Suponha que $L_{B^*} = L'$. Pela escolha de B^*, temos que $R_{B^*} = N^{\cap}(L')$. Observe que $LimSup \geq |L'| \cdot |R'| = |E_{B^*}|$, como queríamos mostrar. Suponha agora que $L_{B^*} \neq L'$. Seja l' um índice tal que $j < l' \leq |L|$ e $f(l')$ é máximo, e seja u_l o vértice de menor índice em $\{u_{j+1}, \ldots, u_{|L|}\}$ tal que $u_l \in L_{B^*}$. Como $R_{B^*} \subseteq (N(u_l) \cap R')$, temos que $|R_{B^*}| \leq |N(u_l) \cap R'|$. Pela escolha de u_l, temos que $u_i \notin L_{B^*}$, para todo i tal que $j < i < l$, o que implica em $|L_{B*}| \leq |L'| + |L| - l + 1$. Portanto, $LimSup = f(l') \geq f(l) \geq |E_{B^*}|$, como queríamos mostrar. □

Nosso algoritmo de *branch-and-bound* faz a enumeração implícita de todos os subconjuntos de L, no sentido de que com o uso do Lema 2.4.1 alguns subconjuntos não precisam ser enumerados explicitamente durante o processo de enumeração. Dividimos o procedimento em três subrotinas. O Algoritmo 2 apenas inicializa as variáveis globais utilizadas durante o procedimento e faz a primeira

chamada ao Algoritmo 3, que por sua vez faz a busca por um biclique com maior quantidade de arestas em um determinado subconjunto $\mathcal{B}(L', j)$. O Algoritmo 4 utiliza o Lema 2.4.1 para calcular um limitante superior para a quantidade de arestas de qualquer biclique contido em $\mathcal{B}(L', j)$.

Algoritmo 2: EncontraBicliqueMaximo(G)

Entrada: um grafo bipartido G.

Saída: um biclique em G com número máximo de arestas.

```
1  L' ← ∅
2  R' ← R
3  B* ← (L', R')
4  /* L', R' e B* são variáveis globais */
5  /* (L', R') corresponde ao biclique que está sendo considerado */
6  /* B* corresponde ao biclique com mais arestas dentre todos os considerados */
7  B&B(0)
8  devolva B*
```

Algoritmo 3: B&B(i)

Entrada: um índice i em $\{0, 1, \ldots, |L| - 1\}$.

Saída: atualiza a variável B^* como sendo um biclique com a maior quantidade de arestas dentre todos os bicliques contidos em $\{B^*\} \cup \mathcal{B}(L', i + 1)$.

```
1  /* Inclui u_{i+1} em L' e busca por um biclique máximo em B(L', i+1) */
2  L' ← L' ∪ {u_{i+1}}
3  R' ← R' ∩ N(u_{i+1})
4  Seja B' = (L', R')
5  se |E_{B'}| > |E_{B*}| então
6  |  B* ← B'
7  fim
8  se i < |L|  e  calc-lim-sup(i + 1) > |E_{B*}| então
9  |  B&B(i + 1)
10 fim
11 /* Remove u_{i+1} de L' e busca por um biclique máximo em B(L', i+1) */
12 L' ← L' \ {u_{i+1}}
13 R' ← N^∩(L')
14 se i < |L|  e  calc-lim-sup(i + 1) > |E_{B*}| então
15 |  B&B(i + 1)
16 fim
```

Algoritmo 4: CALC-LIM-SUP(j)

Entrada: um inteiro positivo j

Saída: O limitante superior para a quantidade de arestas de qualquer biclique contido em $\mathcal{B}(L', j)$ calculado conforme o Lema 2.4.1

```
1  para l ← j + 1 até |L| faça
2  |  h[l − j] ← |N(u_l) ∩ R'|
3  fim
4  n ← |L| − j
5  Ordene h[1..n] em ordem não-decrescente
6  LimSup ← 0
7  para l ← 1 até n faça
8  |  ls ← h[l] · (n − l + 1 + |L'|)
9  |  se ls > LimSup então
10 |  |  LimSup ← ls
11 |  fim
12 fim
13 devolva LimSup
```

Para adaptar o procedimento apresentado nesta seção para resolver o problema de BICLIQUE DE PESO MÁXIMO, basta utilizar um limitante superior adequado no lugar do proposto no Lema 2.4.1, conforme apresentamos abaixo.

Observação 2.4.1. *Dados um subconjunto $L' \subseteq L$ e um inteiro não-negativo $j \leq |L|$, seja $B' = (L', N^{\cap}(L'))$. O valor de*

$$LimSup_w = \max\left\{W(B'), \sum_{u \in \{u_{j+1},\ldots,u_{|L|}\}} \sum_{v \in N(u) \cap R'} w(uv)\right\}$$

é um limitante superior para $W(B)$, para qualquer biclique B contido em $\mathcal{B}(L', j)$.

2.5 Uma Heurística

A seguir, apresentamos uma heurística para os problemas de BICLIQUE MÁXIMO e BICLIQUE DE PESO MÁXIMO. A ideia do Algoritmo 5 é iniciar a construção de um biclique B' com um único vértice u de L e seus vizinhos $N(u)$ (ou seja, inicialmente $L_{B'} = u$ e $R_{B'} = N(u)$). A cada passo, inserimos um novo vértice v em $L_{B'}$ e removemos os vértices $R_{B'} \setminus N(v)$. A escolha do próximo vértice que será incluído em cada passo é *gulosa*, no sentido de que escolhemos um vértice que maximiza o ganho na função objetivo. Executamos este mesmo procedimento considerando cada vértice de L como sendo o vértice inicial a ser inserido em $L_{B'}$. Descrevemos o algoritmo considerando-o como uma heurística para o problema de BICLIQUE DE PESO MÁXIMO. No caso em que deseja-se uma heurística para o problema de BICLIQUE MÁXIMO, basta assumir que $w(uv) = 1$, para todo $uv \in E$. Abaixo, mostramos este precedimento em detalhes.

Algoritmo 5: HEURISTICA(G)

Entrada: um grafo bipartido G
Saída: um biclique B em G

1 $B^* \leftarrow (\emptyset, \emptyset)$
2 **para cada** $u \in L$ **faça**
3 $\quad L' \leftarrow \{u\}$
4 $\quad R' \leftarrow N(u)$
5 $\quad B' \leftarrow (L', R')$
6 $\quad \hat{L} \leftarrow L \setminus \{u\}$
7 $\quad$ **se** $W(B') > W(B^*)$ **então**
8 $\quad\quad B^* \leftarrow (L', R')$
9 $\quad$ **fim**
10 $\quad$ **enquanto** $\hat{L} \neq \emptyset$ **faça**
11 $\quad\quad v \leftarrow \arg\max_{v \in \hat{L}} \left(\sum_{l \in N(v) \cap R'} w(vl) - \sum_{k \in R' \setminus N(v)} w(vk) \right)$
12 $\quad\quad \hat{L} \leftarrow \hat{L} \setminus \{v\}$
13 $\quad\quad L' \leftarrow L' \cup \{v\}$
14 $\quad\quad R' \leftarrow R' \cap N(v)$
15 $\quad\quad B' \leftarrow (L', R')$
16 $\quad\quad$ **se** $W(B') > W(B^*)$ **então**
17 $\quad\quad\quad B^* \leftarrow B'$
18 $\quad\quad$ **fim**
19 $\quad$ **fim**
20 **fim**
21 **devolva** B^*

Observe que no problema de BICLIQUE MÁXIMO, a linha 11 do Algoritmo 5 é equivalente a $v \leftarrow \arg\max_{v \in \hat{L}} |N(v) \cap R'|$. Na Seção 2.7, mostramos resultados de experimentos computacionais feitos com o Algoritmo 5.

2.6 Considerações Sobre a Implementação

Nesta seção, fazemos algumas considerações sobre a implementação do algoritmo de *branch-and-bound* proposto na Seção 2.4 e também sobre o método de relaxação da função objetivo proposto na Seção 2.3.

Com relação ao método de relaxação da função objetivo, podemos inicializar a variável γ na linha 1 do algoritmo MRFO com a quantidade de arestas de um biclique qualquer em G. O ideal é que se disponha de uma boa heurística para que o valor inicial de γ seja próximo do valor de uma solução ótima, fazendo com que o número de iterações seja pequeno. Em nossa implementação, utilizamos a heurística descrita na Seção 2.5 para atribuir um valor inicial a γ. De forma análoga, utilizamos essa heurística também no algoritmo de *branch-and-bound* proposto na Seção 2.4. Em nossa implementação, na linha 3 do algoritmo EncontraBicliqueMaximo, em vez de inicializar a variável B^* com $\emptyset$, inicializamos B^* com o biclique encontrado pela heurística.

No algoritmo B&B, uma vez fixada uma ordem para os vértices de L, a ordem relativa em que

os vértices de L são incluídos em L' é sempre a mesma. Chamamos esta estratágia de *escolha estática*. Uma outra estratégia possível é a *escolha dinâmica*, na qual dada uma determinada chamada recursiva, escolhemos dinamicamente qual será o próximo vértice a ser incluído em L'. Por motivo de facilidade na exposição, optamos por descrever nosso algoritmo utilizando a estratégia de escolha estática. No entanto, em nossa implementação utilizamos uma estratégia de escolha dinâmica na qual escolhe-se um vértice u que minimize $|N(u) \cap R'|$ para ser o próximo vértice a ser incluído em L'. Testes práticos mostraram que tal estratégia é significantemente superior a diferentes estratégias estáticas e dinâmicas testadas. Intuitivamente, em tal estratégia o conjunto R' tende a diminuir rapidamente de tamanho, fazendo com que o nível de cada chamada recursiva tenda a ser pequeno.

2.7 Experimentos Computacionais

Nesta seção, mostramos os resultados de experimentos computacionais realizados com os métodos porpostos neste capítulo para resolver os problemas de BICLIQUE MÁXIMO e BICLIQUE DE PESO MÁXIMO. A seguir, listamos quais foram os métodos propostos neste capítulo, bem como a notação que será utilizada nesta seção quando nos referirmos a tais métodos:

- Na Seção 2.1, apresentamos a formulação (PI_{BM}), na qual definimos uma variável binária para cada aresta de G e a função objetivo consiste na maximização do número de arestas escolhidas. Denotamos esta abordagem por PI-CLIQUE.

- Na Seção 2.1, argumentamos que existe uma equivalência entre os problemas de CLIQUE MÁXIMO e de BICLIQUE MÁXIMO. Uma abordagem que consideramos consiste em, dada uma instância do problema de BICLIQUE MÁXIMO, construir o grafo H conforme descrito na Seção 2.1 e utilizar algum dos algoritmos de *branch-and-bound* propostos na literatura para encontrar um clique máximo no grafo $\overline{H}$. Consideramos os algoritmos de Östergård (2002) e de Konc e Janezic (2007) para resolver o problema de CLIQUE MÁXIMO. O código-fonte do algoritmo de Östergård (2002) está disponível no sítio http://users.tkk.fi/pat/cliquer.html. O algoritmo de Konc e Janezic (2007) é baseado no algoritmo de Tomita e Seki (2003) e seu código-fonte encontra-se disponível no sítio http://www.sicmm.org/~konc/maxclique/. Denotamos esses dois algoritmos por OSTERGARD e KONC-JANEZIC, respectivamente.

- Na Seção 2.2, apresentamos a formulação (PI3_{BM}) na qual definimos uma variável binária para cada um dos vértices de $L \cup R$. Denotamos esta abordagem por PI(LR). Ainda na Seção 2.2, apresentamos uma variação desta formulação, na qual definimos variáveis binárias apenas para os vértices de L (formulação (PI4_{BM})) e, por último, mostramos uma versão agregada de (PI4_{BM}) (formulação (PI5_{BM})). Denotamos estas duas últimas abordagens por PI(L) e PI-AGR(L), respectivamente.

- Na Seção 2.3, apresentamos um método que denominamos de *Método de Relaxação da Função Objetivo*, no qual em vez de procurar por um biclique com número máximo de arestas, procuramos por um biclique com número máximo de vértices, sujeito a restrições de tamanho mínimo nas classes da bipartição. Denotamos esta abordagem por MRFO.

- Na Seção 2.4, apresentamos uma algoritmo de *branch-and-bound* que denotamos aqui por B&B.

- Na Seção 2.5, apresentamos uma heurística que denotamos aqui por HEUR.

Todos os programas mencionados acima foram implementados em linguagem C++. Realizamos os experimentos em um computador com 65 GB de memória RAM e um processador com velocidade de 1.6 GHz. Na Seção 2.7.1, apresentamos experimentos computacionais realizados com grafos gerados aleatoriamente, enquanto que, na Seção 2.7.2, mostramos experimentos computacionais realizados com instâncias vindas de algumas das aplicações descritas na Seção 1.3.

2.7.1 Instâncias Geradas Aleatoriamente

Para gerar um grafo bipartido aleatoriamente, utilizamos o procedimento descrito a seguir. Fixamos um inteiro positivo m e dois números α e p, tais que $0 < \alpha \leq 1$ e $0 < p < 1$, então calculamos os tamanhos das classes da bipartição da seguinte forma:

$$|R| = \sqrt{\frac{m}{\alpha p}} \text{ e } |L| = \alpha |R|.$$

Por fim, utilizando um gerador de números pseudoaleatórios, sorteamos cada possível aresta entre os vértices de L e R com probabilidade p. Desta forma, temos que a quantidade esperada de arestas de G é dada por

$$\mathrm{E}[|E|] = |L||R|p = \alpha |R|^2 p = \alpha p \left(\sqrt{\frac{m}{\alpha p}}\right)^2 = m$$

e a densidade esperada de G é dada por $\mathrm{E}[\frac{|E|}{|L|\cdot|R|}] = p$. O parâmetro α representa o "fator de balanceamento" de G. No caso de grafos com pesos nas arestas, sorteamos aleatoriamente o peso de cada aresta entre 1 e $|E|$. Para cada linha das tabelas apresentadas nesta seção, foram geradas 100 instâncias diferentes e calculada a média aritmética dos tempos de processamento em segundos. Excepcionalmente, devido ao grande consumo de tempo de processamento no caso em que a formulação PI-CLIQUE foi utlizada, escolhemos aleatoriamente apenas 10 instâncias para calcular o tempo médio de processamento das execuções dessa formulação.

Experimentos com grafos sem pesos nas arestas

Na Tabela 2.7.1, comparamos os desempenhos dos algoritmos OSTERGARD e KONC-JANEZIC com instâncias geradas aleatoriamente.

Conforme mostrado na Tabela 2.7.1, o algoritmo KONC-JANEZIC obteve um desempenho melhor do que o algoritmo OSTERGARD, principalmente para grafos mais densos. Uma vantagem do algoritmo OSTERGARD é que, ao contrário do algoritmo KONC-JANEZIC, ele pode ser utilizado para resolver o problema de CLIQUE DE PESO MÁXIMO. Portanto, na comparacação com os demais métodos, utilizaremos o algoritmo KONC-JANEZIC no caso do problema de BICLQUE MÁXIMO, e o no caso do problema de BICLIQUE DE PESO MÁXIMO utilizaremos o algoritmo OSTERGARD.

Tabela 2.7.1. Algoritmos de clique máximo – Instâncias aleatórias, onde $m = 500$.

	OSTERGARD		KONC-JANEZIC	
Densidade	$\alpha = 1$	$\alpha = \frac{1}{2}$	$\alpha = 1$	$\alpha = \frac{1}{2}$
0.1	0.0108	0.0100	0.0000	0.0002
0.2	0.0121	0.0099	0.0000	0.0001
0.3	0.0120	0.0100	0.0001	0.0005
0.4	0.0123	0.0115	0.0009	0.0026
0.5	0.0201	0.0249	0.0061	0.0077
0.6	0.1642	0.1418	0.0110	0.0114
0.7	8.0966	3.6573	0.0246	0.0176
0.8	1364.5798	949.4893	0.1626	0.0471
0.9	2201.5232	1422.1040	1.3719	0.1364

Na Seção 2.3, apresentamos duas variantes do *Método de Relaxação da Função Objetivo*, sendo que ambas funcionam basicamente da mesma forma, exceto na escolha entre as formulações ($PI_{2.3.1}$) ou ($PI_{2.3.2}$) em cada iteração do método. Na Tabela 2.7.2, comparamos essas duas abordagens, medindo o tempo de processamento e o número de iterações.

Tabela 2.7.2. Método de Relaxação da Função Objetivo – Instâncias aleatórias, onde $m = 500$.

	MRFO-($PI_{2.3.1}$)				MRFO-($PI_{2.3.2}$)			
	$\alpha = 1$		$\alpha = \frac{1}{2}$		$\alpha = 1$		$\alpha = \frac{1}{2}$	
Densidade	Tempo	Num. It.	Tempo	Num. It.	Tempo	Num. It.	Tempo	Num. It.
0.1	35.6840	3.09	32.0373	3.10	17.2401	1.02	18.7229	1.00
0.2	19.1703	3.95	17.6685	4.00	7.0604	1.01	6.9529	1.00
0.3	9.8275	4.63	7.9160	4.72	3.9161	1.07	3.4744	1.00
0.4	5.6660	5.12	4.0576	5.24	2.5506	1.06	2.3785	1.01
0.5	2.5086	5.90	1.6738	6.08	1.4492	1.16	1.1834	1.01
0.6	0.9084	7.00	0.7405	7.12	0.6176	1.25	0.2802	1.09
0.7	0.4027	8.07	0.3631	8.23	0.1128	1.19	0.0923	1.08
0.8	0.1669	9.10	0.2242	10.10	0.0526	1.25	0.0427	1.10
0.9	0.0232	3.80	0.0341	9.62	0.0145	1.23	0.0083	1.10

Conforme mostrado na Tabela 2.7.2, o MRFO obteve um melhor desempenho utilizando a formulação ($PI_{2.3.2}$). Observe que se considerarmos o tempo total dividido pelo número de iterações, o desempenho de MRFO-($PI_{2.3.1}$) é melhor do que o de MRFO-($PI_{2.3.2}$), mas como o número de iterações de MRFO-($PI_{2.3.2}$) é muito pequeno, esta abordagem é mais eficiente. Nas próximas tabelas, denotaremos MRFO-($PI_{2.3.2}$) apenas por MRFO.

Na Tabela 2.7.3, mostramos os experimentos feitos com grafos balanceados. Como as abordagens de *branch-and-bound* se destacaram das demais, para facilitar a interpretação dos dados, apresentamos os gráficos referentes aos dados da Tabela 2.7.3 separadamente nas Figuras 2.7.4 e 2.7.5, que mostram os resultados das abordagens de Programação Linear Inteira e *branch-and-bound*, respectivamente. Não incluímos nos gráficos mostrados nas Figuras 2.7.4 e 2.7.5 os resultados obtidos por PI-CLIQUE, pois seu tempo de processamento foi muito superior aos demais métodos.

Tabela 2.7.3. Experimentos com grafos balanceados, onde $m = 500$ e $\alpha = 1$.

Dens.	PI-CLIQUE	PI(LR)	PI(L)	PI-AGR(L)	MRFO	B&B	KONC-JANEZIC
0.1	274.7000	30.3278	20.2035	3.7846	17.0697	0.0057	0.0000
0.2	484.7315	12.7935	7.7595	3.2156	6.7404	0.0001	0.0000
0.3	905.2620	7.6075	5.0621	2.5651	3.7176	0.0000	0.0001
0.4	4707.2598	5.8191	4.4255	3.2321	2.3766	0.0016	0.0009
0.5	7522.1548	4.6334	3.4438	2.8737	1.3441	0.0008	0.0061
0.6	8804.6299	3.8466	2.2816	2.2206	0.6088	0.0019	0.0110
0.7	10480.9775	2.8367	1.6718	1.9174	0.0974	0.0041	0.0246
0.8	3602.5488	1.6096	1.0881	1.4069	0.0323	0.0070	0.1626
0.9	48.1856	0.2235	0.2785	0.2594	0.0016	0.0046	1.3719

Figura 2.7.4. Abordagens de Programação Linear Inteira (grafos balanceados).

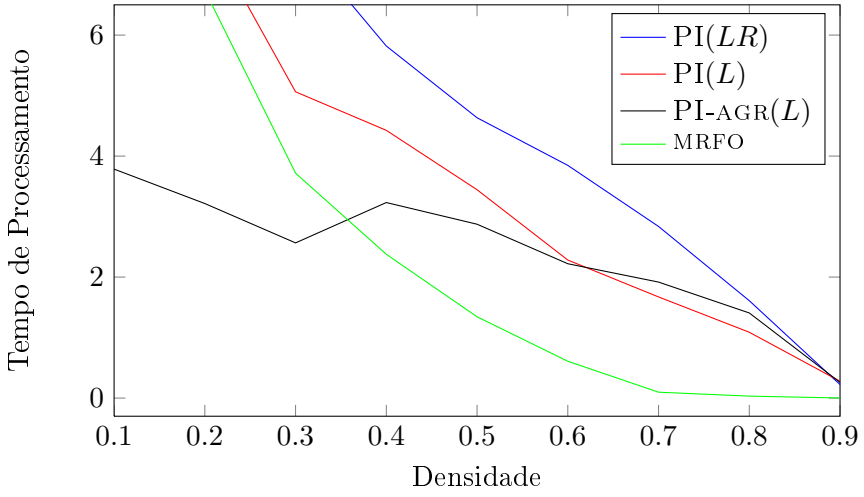
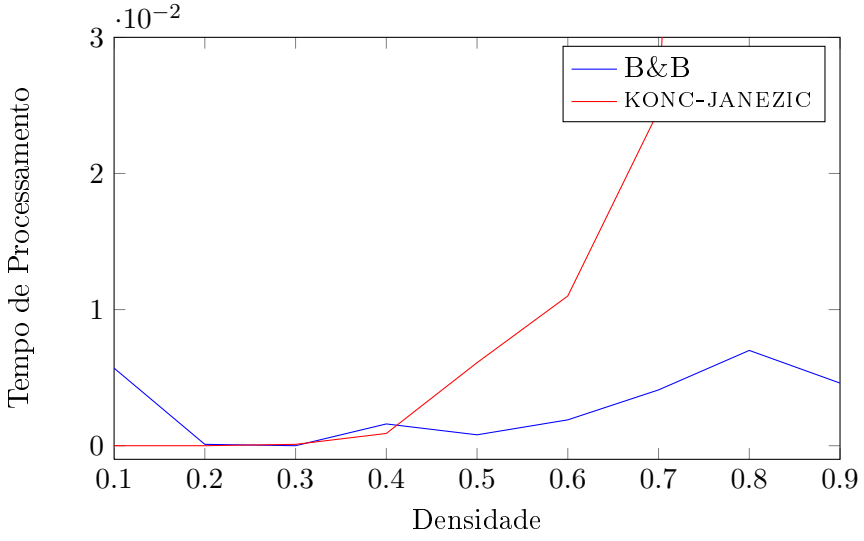


Figura 2.7.5. Abordagens de *branch-and-bound* (grafos balanceados).



Conforme mostrado na Figura 2.7.4, dentre as abordagens de Programação Linear Inteira, os métodos que obtiveram melhor desempenho foram PI-AGR(L), quando a densidade está entre 0.1 e 0.3, e MRFO, quando a densidade está entre 0.4 e 0.9. Em todos os métodos de Programação Linear Inteira, o tempo de processamento diminuiu à medida em que a densidade aumentou, sendo que o

método MRFO se mostrou o mais sensível à variação de densidade, chegando a superar todas as outras abordagens, incluindo as de *branch-and-bound*, quando a densidade é 0.9. Aparentemente, este comportamento deve-se ao fato de que o número de restrições na formulação ($PI_{2.3.1}$) é $O(|L||R| - |E|)$ e, portanto, é bastante pequena quando a densidade está próxima de 1.

Conforme mostrado na Figura 2.7.5, o algoritmo KONC-JANEZIC teve um desempenho ligeiramente melhor do que o B&B quando a densidade está entre 0.1 e 0.4. Porém, quando a densidade está entre 0.5 e 0.9, o algoritmo B&B teve um desempenho muito melhor do que o KONC-JANEZIC, sendo que quanto maior a densidade do grafo, maior foi a discrepância entre o consumo de tempo dos dois algoritmos. Acreditamos que este comportamento se deve ao fato de que quanto maior a densidade de G, menor tende a ser a densidade do grafo H no qual procuramos por um clique e, provavelmente, o algoritmo KONC-JANEZIC encontra mais dificultade em grafos esparsos. Ao contrário das abordagens de Programação Linear Inteira, o tempo de processamento do algoritmo KONC-JANEZIC aumentou à medida em que a densidade aumentou. O algoritmo B&B obteve seus picos de maior tempo de processamento nas densidades 0.1 e 0.8, oscilando entre as transições de densidade de forma não padronizada, embora tenha tido, de forma geral, maior dificuldade com as instâncias de maior densidade.

Na Tabela 2.7.6, mostramos os experimentos computacionais realizados com grafos desbalanceados, nos quais o tamanho de R é o dobro do tamanho de L.

Tabela 2.7.6. Experimentos com grafos desbalanceados, onde $m = 500$ e $\alpha = \frac{1}{2}$.

Dens.	PI-CLIQUE	PI(LR)	PI(L)	PI-AGR(L)	MRFO	B&B	KONC-JANEZIC
0.1	348.7616	22.5516	18.4671	2.7715	18.1552	0.0000	0.0002
0.2	484.6295	9.0671	5.8487	1.9704	6.8767	0.0000	0.0001
0.3	782.5916	4.5657	3.0695	1.5088	3.4095	0.0000	0.0005
0.4	4081.8677	3.4767	2.4597	1.6845	2.3842	0.0000	0.0026
0.5	8241.7773	2.5286	1.7455	1.3812	1.0361	0.0000	0.0077
0.6	11566.3447	1.9863	1.2521	1.1703	0.3011	0.0001	0.0114
0.7	6057.6489	1.2839	0.7972	0.9337	0.0803	0.0001	0.0176
0.8	707.8251	0.6836	0.5159	0.7315	0.0348	0.0011	0.0471
0.9	4.2341	0.0572	0.0665	0.0788	0.0022	0.0005	0.1364

Comparando as Tabelas 2.7.3 e 2.7.6, de forma geral todos os métodos tiveram melhor desempenho para grafos desbalanceados. No entanto, alguns métodos foram mais sensíveis à variação no fator de balanceamento do que outros. Nas Figuras 2.7.7, 2.7.8 e 2.7.9, comparamos os resultados mostrados nas Tabelas 2.7.3 e 2.7.6. Conforme mostramos na Figura 2.7.8, o método MRFO variou muito pouco com a mudança no fator de balanceamento. Já o algoritmo B&B obteve um desempenho muito melhor para grafos desbalanceados, superando o algoritmo KONC-JANEZIC em todas as densidades.

Figura 2.7.7. Grafos balanceados vs. desbalanceados (PI(LR) e PI(L)).

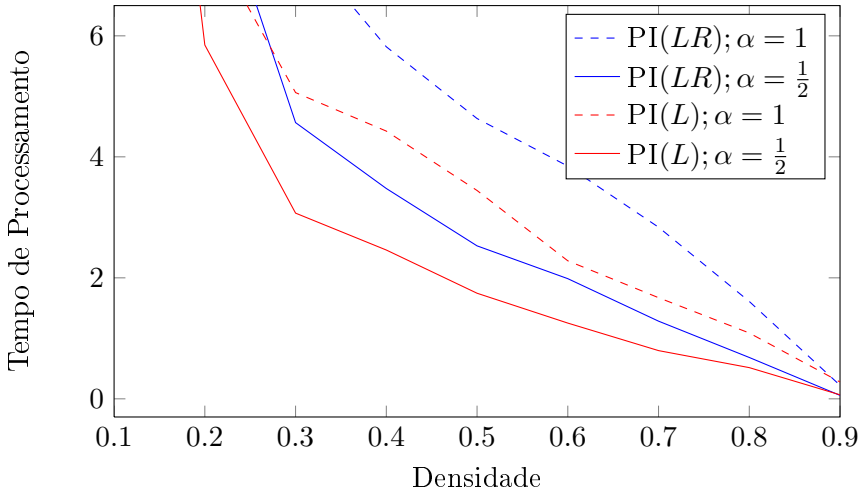


Figura 2.7.8. Grafos balanceados vs. desbalanceados (PI-AGR(L) e MRFO).

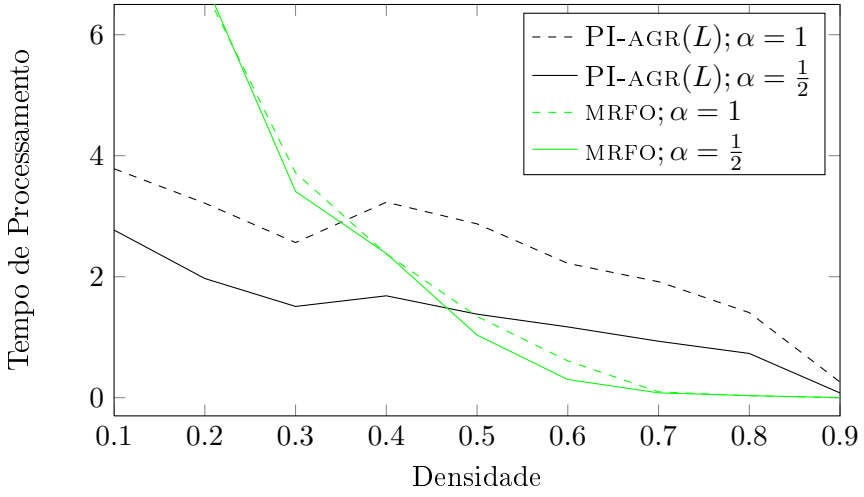
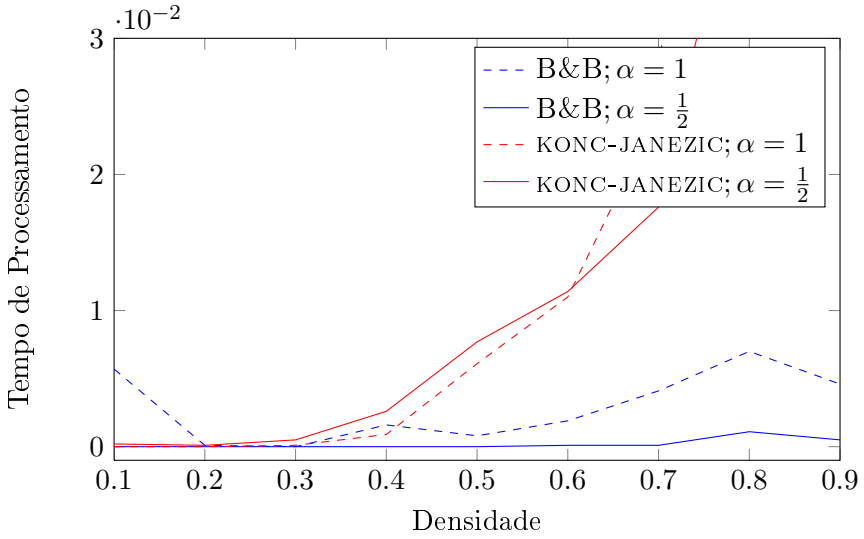


Figura 2.7.9. Grafos balanceados vs. desbalanceados (B&B e KONC-JANEZIC).



Para m suficientemente grande e α suficientemente pequeno, os métodos de Programação Linear Inteira podem ter um desempenho melhor do que o algoritmo KONC-JANEZIC. Na Tabela 2.7.10,

mostramos uma escolha desses parâmetros ($m = 5000$ e $\alpha = \frac{1}{100}$) em que tal comportamento é observado quando o grafo de entrada possui densidade maior ou igual a 0.4.

Tabela 2.7.10. Experimentos com grafos desbalanceados, onde $m = 5000$ e $\alpha = \frac{1}{100}$.

Densidade	PI-AGR(L)	MRFO	B&B	KONC-JANEZIC
0.1	51.6378	397.4775	0.1407	1.1239
0.2	19.5601	67.5292	0.0917	2.6254
0.3	12.4703	21.4789	0.0757	5.6074
0.4	8.9978	13.4101	0.0882	12.7643
0.5	13.3709	7.7397	0.1169	25.4739
0.6	12.2656	3.8061	0.1231	45.6351
0.7	9.7654	1.3934	0.1222	76.6909
0.8	3.4001	0.5005	0.0829	185.0621
0.9	0.1321	0.0630	0.0360	869.4933

Na Tabela 2.7.11, mostramos os resultados obtidos por HEUR no mesmo conjunto de instâncias utilizado nas Tabelas 2.7.3 e 2.7.6. Dada uma instância do problema de BICLIQUE MÁXIMO, seja *opt* o valor de uma solução ótima e *hr* o valor de uma solução encontrada por HEUR. O *gap* entre essas duas soluções é dado por $gap = 100 \cdot \frac{(opt-hr)}{hr}$. O valor do *gap* é usado para medir o quanto a solução encontrada por HEUR está próxima de uma solução ótima, sendo que quanto mais próximo de zero está o valor de *gap*, mais próximos estão *hr* e *opt*. Na coluna "% Sol. Ot.", mostramos o percentual de instâncias para as quais HEUR encontrou uma solução ótima. Nas colunas seguintes, mostramos a média aritmética entre os *gaps* das soluções não-ótimas, seguido do desvio padrão entre os *gaps* das soluções não-ótimas e o *gap* máximo, respectivamente. Como o tempo de processamento foi inferior a 10^{-4} segundos para todas as instâncias, decidimos não mostrar os tempos de processamento na Tabela 2.7.11.

Tabela 2.7.11. Experiementos realizados com o algoritmo HEUR.

	% Sol. Ot.		Gap Médio		Desvio Padrão		Gap Máximo	
Densidade	$\alpha = 1$	$\alpha = \frac{1}{2}$	$\alpha = 1$	$\alpha = \frac{1}{2}$	$\alpha = 1$	$\alpha = \frac{1}{2}$	$\alpha = 1$	$\alpha = \frac{1}{2}$
0.1	98%	100%	8.01%	0.00%	0.32%	0.00%	8.33%	0.00%
0.2	98%	100%	5.56%	0.00%	0.00%	0.00%	5.56%	0.00%
0.3	92%	100%	8.32%	0.00%	4.43%	0.00%	18.18%	0.00%
0.4	94%	100%	5.82%	0.00%	3.10%	0.00%	11.11%	0.00%
0.5	87%	99%	5.45%	4.17%	2.67%	0.00%	12.50%	4.17%
0.6	75%	98%	5.71%	3.54%	2.75%	1.46%	12.00%	5.00%
0.7	77%	93%	5.90%	4.11%	2.77%	1.74%	12.50%	6.06%
0.8	78%	91%	3.70%	3.43%	2.17%	2.64%	7.69%	8.33%
0.9	80%	90%	4.92%	2.52%	2.78%	1.77%	11.11%	6.94%

Conforme mostrado na Tabela 2.7.11, na maioria dos casos o algoritmo HEUR encontrou uma solução ótima. Além disso, nos casos em que a solução encontrada não é ótima, o *gap* obtido foi pequeno. Considerando o percentual de soluções ótimas encontradas, os resultados obtidos por HEUR foram melhores para os grafos mais esparsos. Comparando os resultados obtidos para grafos

balanceados e desbalanceados, observa-se que o algoritmo HEUR teve resultados melhores para grafos desbalanceados.

Experimentos com grafos com pesos nas arestas

A seguir, apresentamos os resultados dos testes realizados com grafos com pesos nas arestas gerados aleatoriamente. Na Tabela 2.7.12, mostramos os experimentos feitos com grafos balanceados. Nas Figuras 2.7.13 e 2.7.14, apresentamos gráficos com os dados contidos na Tabela 2.7.12.

O comportamento observado nos gráficos das Figuras 2.7.13 e 2.7.14 é essencialmente o mesmo observado no caso em que o grafo de entrada não possui pesos nas arestas (veja as Figuras 2.7.4 e 2.7.5). Assim como no caso em que o grafo de entrada não possui pesos nas arestas, o algoritmo OSTERGARD consumiu muito mais tempo de processamento quando os grafos de entrada possuem densidade maior ou igual a 0.8. Comparando os dados das Tabelas 2.6.1 e 2.7.12, podemos notar que o desempenho do algoritmo OSTERGARD foi muito melhor no caso em que o grafo de entrada possui pesos nas arestas. Nosso algoritmo B&B teve o melhor desempenho dentre todas as abordagens apresentadas, para grafos com densidade maior ou igual a 0.2.

Tabela 2.7.12. Grafos balanceados com pesos nas arestas, onde $m = 500$ e $\alpha = 1$.

Densidade	PI-CLIQUE	PI(LR)	PI(L)	PI-AGR(L)	B&B	OSTERGARD
0.1	314.4500	20.8326	20.3467	4.0649	0.0139	0.0107
0.2	511.3070	10.7722	8.0137	3.1384	0.0089	0.0126
0.3	758.7010	6.0634	5.0078	2.4747	0.0095	0.0136
0.4	2430.6050	5.3211	4.3416	2.5744	0.0117	0.0201
0.5	5282.7764	4.3568	3.0264	2.6884	0.0127	0.0313
0.6	5127.6968	3.2579	1.9694	2.2658	0.0167	0.0753
0.7	5652.1196	2.3492	1.2724	1.9146	0.0233	0.3598
0.8	1757.7100	1.4956	0.8902	1.4360	0.0379	7.7781
0.9	69.6800	0.1443	0.1843	0.1850	0.0296	243.2138

Figura 2.7.13. Abordagens de Programação Linear Inteira (grafos balanceados).

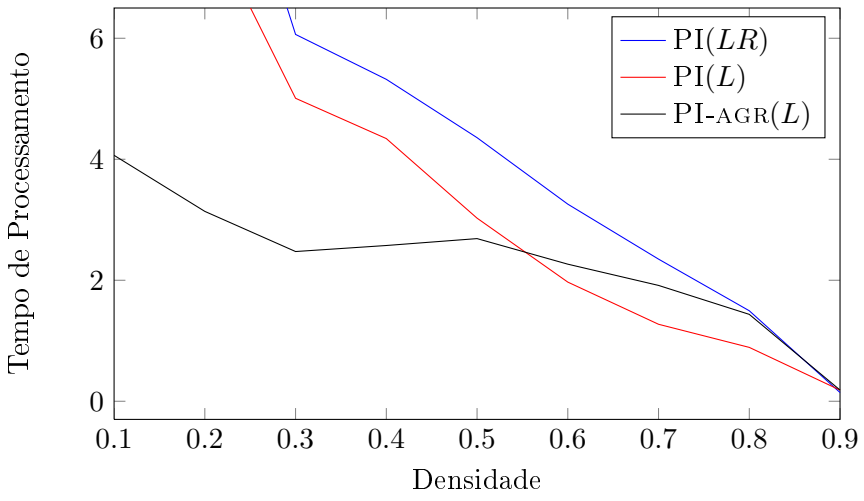
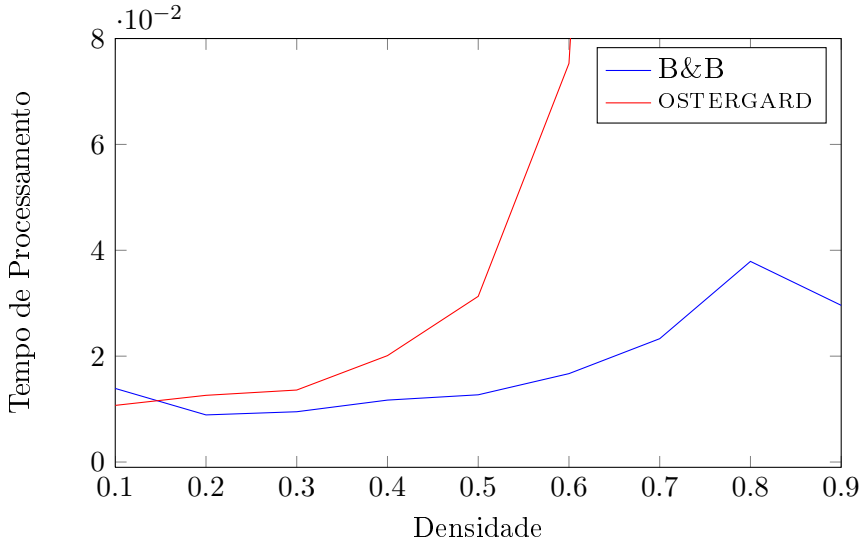


Figura 2.7.14. Abordagens de *branch-and-bound* (grafos balanceados).



Na Tabela 2.7.15, mostramos os experimentos computacionais realizados com grafos desbalanceados, nos quais o tamanho de R é o dobro do tamanho de L.

Tabela 2.7.15. Grafos desbalanceados com pesos nas arestas, onde $m = 500$ e $\alpha = \frac{1}{2}$.

Densidade	PI-CLIQUE	PI(LR)	PI(L)	PI-AGR(L)	B&B	OSTERGARD
0.1	391.6300	10.1664	14.3743	2.5941	0.0050	0.0114
0.2	526.5651	6.2077	5.4393	1.7461	0.0001	0.0119
0.3	630.8140	4.0148	3.0754	1.4112	0.0003	0.0141
0.4	2063.4849	3.3001	2.3231	1.1514	0.0010	0.0210
0.5	3541.7134	2.4408	1.6248	1.3144	0.0014	0.0309
0.6	4234.9800	1.7943	1.1482	1.2736	0.0041	0.0703
0.7	3891.2981	1.2592	0.7207	1.0378	0.0046	0.2573
0.8	401.7880	0.5994	0.4574	0.6679	0.0073	3.4928
0.9	4.3160	0.0638	0.0629	0.0694	0.0029	111.1583

Comparando as Tabelas 2.7.12 e 2.7.15, de forma geral todos os métodos obtiveram melhor desempenho para grafos desbalanceados, assim como no caso em que o grafo de entrada não possui pesos nas arestas. Nas Figuras 2.7.16, 2.7.17 e 2.7.18, comparamos os resultados mostrados nas Tabelas 2.7.12 e 2.7.15.

Figura 2.7.16. Grafos balanceados vs. desbalanceados – Aborgadens de PLI.

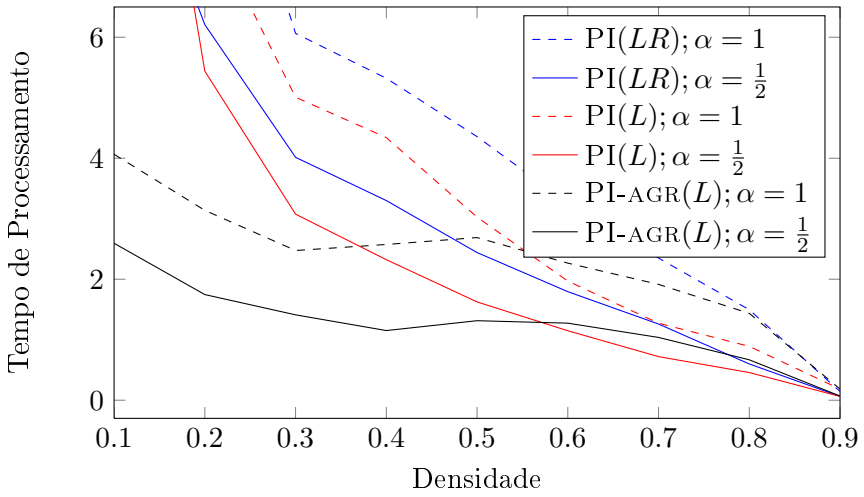
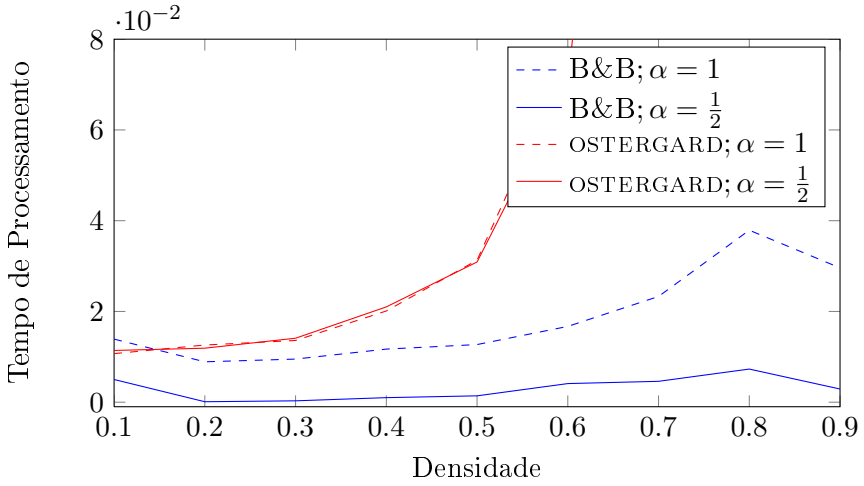


Figura 2.7.17. Grafos balanceados vs. desbalanceados – Algoritmos de *branch-and-bound*.



Na Tabela 2.7.18, mostramos os resultados obtidos por HEUR no mesmo conjunto de instâncias utilizado nas Tabelas 2.7.12 e 2.7.15. Como o tempo de processamento foi inferior a 10^{-4} segundos para todas as instâncias, decidimos não mostrar os tempos de processamento na Tabela 2.7.18.

Tabela 2.7.18. Grafos com pesos nas arestas – Algoritmo HEUR.

	% Sol. Ot.		Gap Médio		Desvio Padrão		Gap Máximo	
Densidade	$\alpha = 1$	$\alpha = \frac{1}{2}$	$\alpha = 1$	$\alpha = \frac{1}{2}$	$\alpha = 1$	$\alpha = \frac{1}{2}$	$\alpha = 1$	$\alpha = \frac{1}{2}$
0.1	95%	100%	3.93%	0.00%	1.87%	0.00%	7.06%	0.00%
0.2	96%	100%	2.72%	0.00%	1.29%	0.00%	4.40%	0.00%
0.3	65%	92%	5.37%	3.92%	4.46%	2.41%	16.52%	8.19%
0.4	52%	96%	6.07%	6.25%	4.31%	3.84%	16.98%	12.65%
0.5	52%	81%	6.84%	3.00%	5.08%	2.34%	19.74%	8.82%
0.6	24%	54%	7.73%	3.92%	6.22%	3.71%	26.28%	19.19%
0.7	15%	38%	8.32%	4.87%	6.37%	4.15%	26.70%	19.21%
0.8	8%	23%	10.28%	6.94%	7.00%	4.75%	31.33%	19.92%
0.9	2%	6%	13.14%	8.26%	7.58%	5.55%	34.06%	28.16%

Conforme mostrado na Tabela 2.7.18, considerando os grafos com densidade menor ou igual a 0.6, na maioria dos casos o algoritmo HEUR encontrou uma solução ótima. Quanto maior a densidade do grafo de entrada, menores foram os percentuais de soluções ótimas encontradas. Além disso, nos casos em que a solução encontrada não é ótima, quanto maior a densidade do grafo de entrada, maiores foram os *gaps* obtidos. De forma geral, os *gaps* obtidos foram pequenos. Comparando os resultados obtidos para grafos balanceados e desbalanceados, observa-se que o algoritmo HEUR teve resultados melhores para grafos desbalanceados, assim como no caso em que o grafo de entrada não possui pesos nas arestas.

2.7.2 Instâncias Vindas de Aplicações

Na Tabela 2.7.19, mostramos o resultado de experimentos computacionais realizados com instâncias vindas das aplicações de *empacotamento de produtos* (EP) e *redes metabólicas* (RM). As instâncias utilizadas para gerar a Tabela 2.7.19 são as mesmas utilizadas no trabalho de Acuña *et al.* (2011).

Tabela 2.7.19. Experimentos computacionais realizados com instâncias vindas de aplicações.

Inst.	$\|L\|$	$\|R\|$	α	$\|E\|$	Densidade	PI-AGG(L)	MRFO	B&B	KONC-JANEZIC
EP1	32	760	$\frac{1}{24}$	9449	0.4	1m13s	14s	0.03s	13m01s
EP2	77	760	$\frac{1}{10}$	15519	0.3	14m36s	46m30s	0.37s	55m19s
RM1	16	4637	$\frac{1}{290}$	41010	0.6	9m06s	2m	0.001s	12h36m31s
RM2	40	4637	$\frac{1}{116}$	71140	0.4	8m10s	14m55s	0.04s	LME

Apesar de os grafos de entrada possuirem um grande número de arestas, o tamanho da classe L em todos eles é bem pequeno ($|L| \leq 77$). Como consequência deste fato, o algoritmo KONC-JANEZIC foi o que teve pior desempenho quando comparado aos demais, enquanto que B&B teve um desempenho bastante satisfatório. Vale salientar que o algoritmo HEUR encontrou uma solução ótima para cada uma das instâncias mostradas na Tabela 2.7.19 em menos de 0.05 segundos. Nas instâncias EP1 e RM1, o método MRFO foi o que teve o segundo melhor desempenho. Acreditamos que tal comportamento não se verificou nas instâncias EP2 e RM2 pelo fato de elas serem menos densas que EP1 e RM1, respectivamente.

Uma das grandes dificuldades do algoritmo KONC-JANEZIC em lidar com instâncias em que G possui uma grande quantidade de arestas é o uso de memória, dado que este algoritmo explora o espaço das arestas e não o espaço dos vértices em L, como os demais. Da forma como o algoritmo KONC-JANEZIC está implementado, é necessário espaço $O(|E|^2)$ para armazenar a matriz de adjacência do grafo H. Supondo que cada entrada da matriz ocupe 1 byte na memória, seriam necessários 5 gigabytes de memória para armazenar a matriz de adjacência correspondente à instância RM2. Evidentemente, esta é uma estimativa que pode estar bastante abaixo do consumo real de memória do algoritmo KONC-JANEZIC, dado que existem outras estruturas utilizadas no programa. Na instância RM2 o algoritmo KONC-JANEZIC teve o limite de memória excedido (LME). Uma outra limitação do algoritmo KONC-JANEZIC é o fato de ele não se beneficiar tanto quanto os demais do fato de L ser pequeno, fazendo com que seu desempenho seja competitivo apenas no caso em que G não é muito desbalanceado.

Infelizmente, não dispomos de uma quantidade suficiente de instâncias para identificar se existe algum padrão na estrutura das instâncias vindas dessas aplicações. Analisando as poucas instân-

cias que dispomos, não pudemos identificar nenhuma estrutura que seja relevante para explicar o comportamento observado na Tabela 2.7.19. Consideramos nossos resultados satisfatórios, pois pudemos resolver todas as instâncias em menos de 1 segundo.

Conforme apresentado na Tabela 2.7.19, as instâncias das aplicações que consideramos possuem densidade menor ou igual a 0.6 e os grafos são bastante desbalanceados. Na Tabela 2.7.20, apresentamos experimentos computacionais realizados com instâncias geradas aleatoriamente, simulando as mesmas características das instâncias vindas das aplicações e utilizando o algoritmo B&B para resolver tais instâncias. Nosso objetivo foi o de verificar o tempo de execução do algoritmo em função do tamanho das instâncias, sendo que estipulamos um tempo máximo de processamento de 2 horas (utilizamos a sigla TLE para designar Tempo Limite Excedido).

Tabela 2.7.20. Experimentos realizados com o algoritmo B&B ($\alpha = \frac{1}{24}$).

	$\|E\|$				
Densidade	10000	20000	30000	40000	50000
0.1	7.88	22.18	39.77	61.49	84.39
0.2	5.44	22.53	66.64	132.63	221.86
0.3	9.96	44.56	104.66	199.53	358.13
0.4	14.71	95.83	313.68	773.79	1504.75
0.5	29.47	230.97	1026.03	2767.80	TLE
0.6	69.17	840.87	3626.25	TLE	TLE

Conforme mostrado na Tabela 2.7.20, o algoritmo B&B é bastante robusto, quando aplicado a grafos desbalanceados e não muito densos.

2.8 Considerações

Em nosso primeiro contato com o problema de BICLIQUE MÁXIMO, consideramos a formulação PI-CLIQUE. Após tentativas de obter uma formulação mais compacta, desenvolvemos as formulações PI(LR), PI(L) e PI-AGR(L). A formulação PI-AGR(L) foi suficiente para melhorar bastante os resultados anteriores que possuíamos e também para superar o algoritmo KONC-JANEZIC no caso em que G é bastante desbalanceado. Ao notar que os algoritmos mais rápidos para o problema de CLIQUE MÁXIMO são os que utilizam a técnica de *branch-and-bound*, decidimos desenvolver um algoritmo de *branch-and-bound* específico para o problema de BICLIQUE MÁXIMO. Foi então que chegamos à solução que consideramos a mais eficiente para o problema. O único caso em que o algoritmo KONC-JANEZIC ainda é o mais eficiente é quando G é balanceado e esparso, sendo que neste caso a diferença de tempo de processamento entre os algoritmos KONC-JANEZIC e B&B não é muito significativa.

Paralelamente a isso, desenvolvemos o método descrito no Capítulo 4 para resolver um problema de Programação Bilinear Inteira, cujo problema de BICLIQUE MÁXIMO é um caso especial. O método MRFO descrito na Seção 2.3 é uma simplificação bastante natural do método descrito no Capítulo 4. Não tínhamos a pretenção de que o MRFO fosse competitivo quando comparado aos demais métodos apresentados nesta seção. Ficamos surpresos ao perceber que em alguns casos o MRFO é mais eficiente que as demais formulações de Programação Linear Inteira e, além disso, quando G é muito denso, o método MRFO chega a se equiparar aos algoritmos de *branch-and-bound*.

Com exceção do método MRFO, com poucas adaptações, todas as abordagens apresentadas neste capítulo podem ser utilizadas também para resolver o problema de BICLIQUE DE PESO MÁXIMO. Nossos experimentos mostraram que não houve grande diferença no desempenho de nossas abordagens nesse caso mais geral do problema.

Acreditamos que em todos os métodos apresentados nesta seção existem possibilidades de aprimoramento. No caso dos métodos que envolvem Programação Linear Inteira, pode-se tentar encontrar inequações mais fortes que melhorem os limitantes obtidos pelas respectivas relaxações lineares. Com relação ao algoritmo B&B, assim como em todo algoritmo de *branch-and-bound*, existe um limiar entre qualidade dos limitantes e o tempo de processamento exigido para obtê-los. Portanto, para construir algoritmos de *branch-and-bound* mais eficientes do que B&B, deve-se considerar esses dois pontos.

Capítulo 3

Empacotamento de Bicliques

Neste capítulo, apresentamos nossos resultados para o problema de EMPACOTAMENTO DE BICLIQUES Os principais resultados apresentados neste capítulo foram extraídos dos trabalhos de Acuña *et al.* (2010) e Acuña *et al.* (2011), sendo que aqui explicamos nossa abordagem em mais detalhes.

A seguir, reapresentamos algumas definições dadas na Seção 1.2. Um *empacotamento* de bicliques em G é um conjunto P de bicliques em G, dois a dois disjuntos nos vértices que estão contidos em S, onde $S = V$ ou $S \in \{L, R\}$. Dado um grafo bipartido G e um inteiro positivo k, o problema de EMPACOTAMENTO DE BICLIQUES consiste em encontrar um empacotamento P de bicliques em G, tal que $|P| \leq k$ e $\sum_{B \in P} |E_B|$ seja máximo. Dado um biclique B, denotamos por S_B o conjunto dos vértices de $V_B \cap S$. Dizemos que duas arestas uv e pq de E são *incompatíveis* se $u \neq p$, $v \neq q$ e $uq \notin E$ ou $pv \notin E$. Denotamos por $\mathcal{I} = \{\{uv, pq\} \in E \times E \mid uv \text{ e } pq \text{ são incompatíveis}\}$ o conjunto de todos os pares de arestas incompatíveis de E.

Na Seção 3.1, apresentamos uma formulação para o problema de EMPACOTAMENTO DE BICLIQUES. Argumentamos que tal formulação não produz resultados práticos satisfatórios devido à grande quantidade de soluções simétricas. Na Seção 3.2, apresentamos uma outra formulação para o problema, na qual a técnica de *branch-and-price* é empregada. Por fim, na Seção 3.3 apresentamos experimentos computacionais realizados com grafos bipartidos gerados aleatoriamente e também com instâncias vindas de aplicações.

3.1 O Inconveniente de Formulações Com Simetria

Como o problema de BICLIQUE MÁXIMO é um caso especial do problema de EMPACOTAMENTO DE BICLIQUES, no qual $k = 1$, uma ideia natural seria modificar as formulações apresentadas no Capítulo 2, adaptando-as para resolver o problema de EMPACOTAMENTO DE BICLIQUES. Nesta seção, argumentamos que esta abordagem não produziria resultados práticos satisfatórios, devido ao problema de simetria que surge em tais formulações.

Para toda aresta $uv \in E$, seja $M(uv) = \{pq \in E \mid \{u, v\} \cap \{p, q\} \neq \emptyset \text{ e } \{u, v\} \cap \{p, q\} \subseteq S\}$ o conjunto de todas as arestas que possuem um vértice em comum com a aresta uv em S, incluindo a própria aresta uv. A seguir, apresentamos uma formulação para o problema de EMPACOTAMENTO DE BICLIQUES.

$$
\begin{array}{lll}
& \text{maximize} & \displaystyle\sum_{l=1}^{k}\sum_{uv\in E} x_{uv}^{l} \\
(\mathrm{PI}_{\mathrm{EB}}) & \text{sujeito a} & x_{uv}^{l} + x_{pq}^{l} \leq 1, \qquad \text{para todo } \{uv, pq\} \in \mathcal{I} \text{ e } l = 1, 2, \ldots, k \qquad (8.1) \\
& & x_{uv}^{l} + x_{pq}^{t} \leq 1, \qquad \text{para todo } uv \in E,\ pq \in M(uv) \text{ e } l \neq t \qquad (8.2) \\
& & x \in \{0,1\}^{|E|\times k} \qquad (8.3)
\end{array}
$$

A formulação ($\mathrm{PI}_{\mathrm{EB}}$) é uma generalização da formulação ($\mathrm{PI}_{\mathrm{BM}}$) apresentada da Seção 2.1. De fato, quando $k = 1$ a formulação ($\mathrm{PI}_{\mathrm{EB}}$) é exatamente igual a ($\mathrm{PI}_{\mathrm{BM}}$). Dada uma solução ótima x de ($\mathrm{PI}_{\mathrm{EB}}$), obtemos um empacotamento de bicliques $P = \{B_1, B_2, \ldots, B_k\}$ da seguinte maneira. Para todo $l \in \{1, 2, \ldots, k\}$, seja B_l o subgrafo de G induzido por $E_{B_l} = \{uv \in E \mid x_{uv}^{l} = 1\}$. A função objetivo e as restrições (8.1) garantem que B_l é um biclique, para todo $l \in \{1, 2, \ldots, k\}$. As restrições (8.2) garantem que $S_{B_l} \cap S_{B_t} = \emptyset$, para todo par de rótulos distintos l, t em $\{1, 2, \ldots, k\}$. Logo, P é um empacotamento de bicliques em G. Além disso, pela função objetivo, temos que $\sum_{B\in P} |E_B|$ é máximo. Portanto, ($\mathrm{PI}_{\mathrm{EB}}$) é uma formulação para o problema de EMPACOTAMENTO DE BICLIQUES.

A seguir, argumentamos que a formulação ($\mathrm{PI}_{\mathrm{EB}}$) apresenta um problema de simetria que a torna insatisfatória, de um ponto de vista prático. Para expor em mais detalhes quais são as dificuldades em questão, descrevemos superficialmente a forma como os resolvedores de Programação Linear Inteira funcionam. Por simplicidade, assumiremos que todas as variáveis inteiras do programa linear inteiro em questão são binárias. De forma geral, é executado um procedimento de *branch-and-bound* em que, para cada nó da árvore de busca, o valor de algumas variáveis binárias são fixados e o valor das demais variáveis são obtidos através da solução da relaxação linear correspondente. Se a solução da relaxação linear é inteira, temos uma solução ótima para o respectivo nó. Caso contrário, seleciona-se uma variável x com valor fracionário e são criados dois ramos a partir deste nó, sendo que em um deles o valor de x é fixado em 1 e no outro o valor de x é fixado em 0. Os limitantes inferiores são obtidos através de heurísticas e os limitantes superiores são obtidos através das relaxações lineares de cada nó explorado (no caso de problemas de minimização, invertem-se os papeis dos limitantes inferiores e superiores). Dessa forma, a relaxação linear de uma formulação desempenha um papel muito importante neste tipo de abordagem, pois quanto menor for a diferença dos valores na função objetivo entre uma solução inteira ótima e uma solução ótima da relaxação linear, menor tende a ser a quantidade de nós explorados durante a execução do *branch-and-bound*. Em muitos casos, o limitante obtido através da relaxação linear não é suficientemente bom para que resultados satisfatórios sejam obtidos. Por isso, os resolvedores implementam rotinas para encontrar planos-de-corte durante a fase de resolução das relaxações lineares, a fim de melhorar os limitantes superiores obtidos em cada nó. Ou seja, em vez de encontrar uma solução ótima para uma determinada relaxação linear, o resolvedor inclui novas inequações que eliminam soluções fracionárias que antes eram viáveis, fazendo com que o limitante superior encontrado seja menor. Este método de unir o *branch-and-bound* com a geração de planos-de-corte é conhecido como *branch-and-cut*. Para uma introdução ao assunto, veja os livros de Schrijver (1986) e Wolsey (1998). As rotinas para encontrar planos-de-corte implementadas nesses resolvedores são de caráter genérico. Ou seja, trata-se de rotinas que podem ser usadas para diversas formulações diferentes. O usuário também pode implementar rotinas adicionais para encontrar planos-de-corte que sejam específicas para a

formulação em questão.

Considere a relaxação linear de (PI_{EB}), denotada por (PL_{EB}). Observe que se atribuirmos o valor $\frac{1}{2}$ a cada uma das variáveis de (PL_{EB}), obtemos uma solução viável com valor $(|E|k)/2$ na função objetivo. Isso implica que o valor na função objetivo de uma solução ótima de (PL_{EB}) é no mínimo $(|E|k)/2$. Como cada aresta não pode pertencer a dois bicliques diferentes contidos em um mesmo empacotamento, temos que o valor na função objetivo de qualquer solução viável de (PI_{EB}) é no máximo $|E|$. Para $k \geq 2$, o limitante obtido através de (PL_{EB}) é maior ou igual a $|E|$. Consequentemente, este limitante não nos fornece nenhuma informação relevante sobre a instância do problema a ser resolvida. Portanto, para que possamos obter resultados práticos satisfatórios utilizando a formulação (PI_{EB}), necessitamos de boas rotinas para geração de planos-de-corte, sendo elas de caráter genérico ou específico.

Dizemos que duas soluções distintas de (PI_{EB}) são *simétricas* se elas induzem o mesmo agrupamento de arestas. Por exemplo, seja $P = \{B_1, B_2\}$ um empacotamento de bicliques em G. Sejam x e x' duas soluções viáveis de (PI_{EB}), tais que $x^1_{uv} = 1$ se e só se $uv \in E_{B_1}$, $x^2_{uv} = 1$ se e só se $uv \in E_{B_2}$, $x'^1_{uv} = 1$ se e só se $uv \in E_{B_2}$, $x'^2_{uv} = 1$ se e só se $uv \in E_{B_1}$. Dizemos então que x e x' são simétricas. Isto porque, apesar das soluções x e x' serem distintas, elas induzem o mesmo empacotamento. Observe que (PI_{EB}) possui uma grande quantidade de soluções simétricas. De fato, dada uma solução x de (PI_{EB}), considerando todas as permutações possíveis dos rótulos, conforme mostrado no exemplo anterior, existem $k! - 1$ soluções simétricas a x. Essa relação de simetria que existe entre as soluções de (PI_{EB}) também existe entre as soluções de (PL_{EB}). Logo, caso utilizemos algum plano-de-corte para eliminar uma determinada solução fracionária, devemos nos preocupar também com suas soluções simétricas. Esta propriedade, na maioria dos casos, torna impraticável o uso de formulções que apresentam essa relação de simetria entre suas soluções. Se adaptássemos as outras formulações apresentadas no Capítulo 2 para resolver o problema de EMPACOTAMENTO DE BICLIQUES, recairíamos no problema de simetria. Por isso, empregamos uma outra abordagem para resolver o problema de EMPACOTAMENTO DE BICLIQUES.

Campêlo *et al.* (2008) apresentam o conceito de *representantes*, utilizado para evitar o problema de simetria. O problema dessa abordagem é que o número de restrições adicionais necessárias para incorporar este conceito em uma formulação linear inteira é muito grande, resultando em um tempo de processamento muito maior para resolver sua respectiva relaxação linear. Como contrapartida, seria necessário que o limitante produzido pela relaxação linear fosse muito bom, o que é muito difícil de garantir na maioria dos casos. Poucos trabalhos utilizam o conceito de representantes.

Uma outra abordagem para a eliminação de simetria é a utilização da técnica de *geração de colunas*. Para uma introdução ao assunto, veja os trabalhos de Barnhart *et al.* (1998) e Wilhelm (2001). Essa técnica é bastante difundida e podemos encontrar na literatura trabalhos que apontam resultados bastante satisfatótios de sua aplicação a diversos tipos diferentes de problemas. Alguns exemplos clássicos podem ser encontrados nos trabalhos de Mehrotra e Trick (1996) e Ryan e Foster (1981). Por este motivo, decidimos aplicar a técnica de geração de colunas ao problema de EMPACOTAMENTO DE BICLIQUES, conforme apresentamos na próxima seção.

3.2 Utilizando a Técnica Geração de Colunas

Seja $\mathcal{B}$ o conjunto de todos os bicliques em G e, para todo $v \in V$, seja $\mathcal{B}(v) = \{B \in \mathcal{B} \mid v \in V_B\}$ o conjunto de todos os bicliques em G que contêm o vértice v. Definimos um vetor de variáveis binárias $x \in \{0,1\}^{|\mathcal{B}|}$ para representar o vetor característico de um empacotamento de bicliques. Ou seja, um biclique B em G está contido no empacotamento se e só se $x_B = 1$. A seguir, apresentamos uma formulação para o problema de EMPACOTAMENTO DE BICLIQUES.

$$
\begin{array}{lrcll}
 & \text{maximize} & \displaystyle\sum_{B\in\mathcal{B}} |E_B| \cdot x_B & & \\
(\text{PI2}_{\text{EB}}) & \text{sujeito a} & \displaystyle\sum_{B\in\mathcal{B}} x_B & \leq & k \qquad (9.1) \\
 & & \displaystyle\sum_{B\in\mathcal{B}(u)} x_B & \leq & 1, \qquad \text{para todo } u \in S \qquad (9.2) \\
 & & x & \in & \{0,1\}^{|\mathcal{B}|} \qquad (9.3)
\end{array}
$$

A restrição (9.1) garante que no máximo k bicliques são escolhidos. As restrições (9.2) garantem que os bicliques escolhidos são disjuntos nos vértices que estão contidos em S. Logo, qualquer solução viável de (PI2_{EB}) corresponde ao vetor característico de um empacotamento de bicliques em G. Reciprocamente, para cada empacotamento de bicliques em G, existe uma solução viável de (PI2_{EB}) correspondente. Considerando a função objetivo, temos que uma solução ótima de (PI2_{EB}) corresponde a um empacotamento de peso máximo. Portanto, (PI2_{EB}) é uma formulação para o problema de EMPACOTAMENTO DE BICLIQUES. Seja (PL2_{EB}) a relaxação linear de (PI2_{EB}), na qual a restrição (9.3) é substituída pela seguinte restrição: $x_B \geq 0$, para todo $B \in \mathcal{B}$. Nesta seção, mostramos apenas como encontrar uma solução ótima de (PL2_{EB}). Na seção seguinte, voltamos ao problema de encontrar uma solução ótima de (PI2_{EB}).

Observe que a quantidade de variáveis definidas em (PL2_{EB}), que corresponde à quantidade de bicliques em G, pode ser exponencial no tamanho do grafo de entrada, mesmo se considerássemos apenas os bicliques maximais (nos trabalhos de Ganter e Reuter (1991) e Alexe *et al.* (2004) são apresentados algoritmos para enumerar todos os bicliques maximais de G). Portanto, apenas para construir o programa linear (PL2_{EB}), seria necessário um consumo de espaço e tempo de processamento exponenciais no pior caso. De modo geral, utilizando a técnica de geração de colunas, consideramos inicialmente apenas um pequeno subconjunto das variáveis de (PL2_{EB}) e adicionamos as outras variáveis à medida em que elas são necessárias para aumentar o valor da função objetivo. Repetimos este procedimento até que não seja mais possível aumentar o valor da função objetivo. Na maioria dos casos, encontramos uma solução ótima de (PL2_{EB}) sem a necessidade de incluir explicitamente na formulação todas as variáveis. Conceitualmente, é como se atribuíssemos o valor zero para as variáveis que não foram incluídas durante o procedimento de geração de colunas. O nome *geração de colunas* vem do fato de que ao incluir uma nova variável na formulação, precisamos inserir (*gerar*) uma nova coluna na matriz dos coeficientes das restrições do programa linear em questão. Antes de mostrar o critério de parada utilizado no procedimento de geração de colunas, vamos expor alguns fatos sobre *dualidade*, conforme apresentamos a seguir.

Sejam $\beta \in \mathbb{R}$ e $\alpha \in \mathbb{R}^{|S|}$, tais que $\beta \geq 0$ e $\alpha_u \geq 0$, para todo $u \in S$. Suponha que a seguinte condição seja satisfeita: $\beta + \sum_{u\in S_B} \alpha_u \geq |E_B|$, para todo $B \in \mathcal{B}$. Neste caso, para qualquer solução

x^* viável de (PL2_{EB}), vale que $\sum_{B\in\mathcal{B}} |E_B| \cdot x^* \leq \sum_{B\in\mathcal{B}}(\beta + \sum_{u\in S_B} \alpha_u) \cdot x^*$. Como $\sum_{B\in\mathcal{B}} x^*_B \leq k$, então $\sum_{B\in\mathcal{B}} \beta x^*_B \leq k\beta$. Como $\sum_{B\in\mathcal{B}(u)} x^*_B \leq 1$, para todo $u \in S$, então $\sum_{B\in\mathcal{B}} \sum_{u\in S_B} \alpha_u x^*_B \leq \sum_{u\in S} \alpha_u$. Consequentemente, temos que $\sum_{B\in\mathcal{B}} |E_B| \cdot x^*_B \leq \sum_{B\in\mathcal{B}}(\beta + \sum_{u\in S_B} \alpha_u) \cdot x^*_B \leq \sum_{u\in S} \alpha_u + k\beta$ (ou seja, $\sum_{u\in S} \alpha_u + k\beta$ é um limitante superior para o valor de uma solução ótima de (PL2_{EB})). O melhor limitante que podemos obter utilizando esta observação são valores para β e α que minimizem $\sum_{u\in S} \alpha_u + k\beta$. Do teorema de *dualidade forte*, visto na Seção 1.1.1, segue que se x^* é uma solução ótima de (PI2_{EB}) e α e β minimizam $\sum_{u\in S} \alpha_u + k\beta$, obtemos a igualdade $\sum_{u\in S} \alpha_u + k\beta = \sum_{B\in\mathcal{B}} |E_B| \cdot x^*_B$. Para formalizar este conceito, apresentamos abaixo o programa linear *dual* de (PL2_{EB}).

(DPL2_{EB})

$$\text{minimize} \quad \sum_{u\in S} \alpha_u + k\beta$$

$$\text{sujeito a} \quad \beta + \sum_{u\in S_B} \alpha_u \geq |E_B|, \quad \text{para todo } B \in \mathcal{B} \tag{10.1}$$

$$\alpha_u \geq 0, \quad \text{para todo } u \in S \tag{10.2}$$

$$\beta \geq 0 \tag{10.3}$$

A variável dual β é associada à restrição (9.1) de (PL2_{EB}) e as variáveis duais α são associadas às restrições (9.2) de (PL2_{EB}). Dado um subconjunto de bicliques $\mathcal{B}' \subseteq \mathcal{B}$ em G, denotamos por ($\text{PL2}_{\text{EB}}(\mathcal{B}')$) o programa linear ($\text{PL2}_{\text{EB}}$) restrito a $\mathcal{B}'$. Ou seja, a formulação ($\text{PL2}_{\text{EB}}(\mathcal{B}')$) contém as mesmas restrições de (PL2_{EB}), porém em ($\text{PL2}_{\text{EB}}(\mathcal{B}')$) somente as variáveis correspondentes aos bicliques em $\mathcal{B}'$ são definidas. Oberve que, dada uma solução x viável de ($\text{PL2}_{\text{EB}}(\mathcal{B}')$), existe uma solução x' viável de (PL2_{EB}), tal que $x'_B = x_B$, para todo $B \in \mathcal{B}'$, e $x'_B = 0$, para todo $B \in \mathcal{B} \setminus \mathcal{B}'$. Por simplicidade, em vez de construir x', diremos que x é viável de (PL2_{EB}). Denotamos por ($\text{DPL2}_{\text{EB}}(\mathcal{B}')$) o programa linear dual de ($\text{PL2}_{\text{EB}}(\mathcal{B}')$), no qual as restrições (10.1) são definidas apenas para os bicliques de $\mathcal{B}'$. Do teorema de *dualidade forte*, segue o seguinte corolário.

Corolário 3.2.1. *Seja $\mathcal{B}'$ um subconjunto de $\mathcal{B}$. Dada uma solução ótima x^* de ($\text{PL2}_{\text{EB}}(\mathcal{B}')$) e sua correspondente solução dual complementar (β^*, α^*), temos que se (β^*, α^*) é viável de (DPL2_{EB}), então x^* é uma solução ótima de (PL2_{EB}) e (β^*, α^*) é uma solução ótima de (DPL2_{EB}).*

O Corolário 3.2.1 nos fornece uma propriedade que pode ser utilizada como condição de parada no processo de geração de colunas. Dada uma solução ótima x^* de ($\text{PL2}_{\text{EB}}(\mathcal{B}')$), basta verificar se sua correspondente solução dual complementar (β^*, α^*) viola alguma restrição de (DPL2_{EB}). Em caso afirmativo, uma tal restrição pode ser utilizada para gerar a próxima coluna, já que cada restrição de (DPL2_{EB}) corresponde a uma coluna de (PL2_{EB}). Caso contrário, x^* é uma solução ótima de (PL2_{EB}) e, portanto, o algoritmo para. Formalmente, estamos interessados em resolver o seguinte problema.

Problema 3.2.1. *Dado um grafo bipartido G, um número não-negativo β^* e um vetor $\alpha^* \in \mathbb{R}^{|S|}$, tal que $\alpha^* \geq 0$, o problema de* PRICING *consiste em encontrar um biclique B em G, tal que $\beta^* + \sum_{u\in S_B} \alpha^*_u < |E_B|$, ou fornecer um certificado de que $\beta^* + \sum_{u\in S_B} \alpha^*_u \geq |E_B|$, para todo biclique B em $\mathcal{B}$.*

Assuma que dispomos das rotinas descritas a seguir:

- a rotina RESOLVE-PRICING(β^*, α^*, G) recebe um grafo bipartido G, um número não-negativo β^* e um vetor $\alpha^* \in \mathbb{R}^{|S|}$, tal que $\alpha^* \geq 0$, e devolve um biclique B em G, tal que $\beta^* + \sum_{u \in S_B} \alpha^*_u < |E_B|$, ou devolve NULL caso tal biclique não exista.

- a rotina RESOLVE-PL2$_{\text{EB}}$-RESTRITO$(\mathcal{B}')$ recebe um subconjunto de bicliques $\mathcal{B}'$ em G e devolve uma tripla (x^*, α^*, β^*), onde x^* é uma solução ótima de (PL2$_{\text{EB}}(\mathcal{B}')$) e (β^*, α^*) é a solução dual complementar a x^*.

- a rotina ENCONTRA-BICLIQUES-INICIAIS(G) recebe um grafo bipartido G e devolve um conjunto de bicliques $\mathcal{B}' \subset \mathcal{B}$, tal que $\cup_{B \in \mathcal{B}'} S_B = S$.

A seguir, descrevemos um algoritmo de geração de colunas para encontrar uma solução ótima para (PL2$_{\text{EB}}$).

Algoritmo 6: RESOLVE-PL2$_{\text{EB}}(G)$

Entrada: um grafo bipartido G
Saída: uma solução ótima de (PL2$_{\text{EB}}$)

```
1 B' ← ENCONTRA-BICLIQUES-INICIAIS(G)
2 enquanto VERDADEIRO faça
3 |   (x*, α*, β*) ← RESOLVE-PL2_EB-RESTRITO(B')
4 |   B ← RESOLVE-PRICING(β*, α*, G)
5 |   se B = NULL então
6 |   |   devolva x*
7 |   fim
8 |   B' ← B' ∪ {B}
9 fim
```

Como a cada iteração utilizamos os valores de todas as variáveis duais para resolver o problema de PRICING, é necessário que cada restrição de (PL2$_{\text{EB}}(\mathcal{B}')$) tenha pelo menos um coeficiente não nulo, por isso a rotina ENCONTRA-BICLIQUES-INICIAIS devolve um conjunto de bicliques $\mathcal{B}' \subset \mathcal{B}$, tal que $\cup_{B \in \mathcal{B}'} S_B = S$. Além disso, o programa linear (PL2$_{\text{EB}}(\mathcal{B}')$) inicial deve ser viável. Esta propriedade é satisfeita para qualquer subconjunto $\mathcal{B}' \subset \mathcal{B}$. Observe que $\mathcal{B}' = \{B \in \mathcal{B} \mid E_B$ contém apenas uma aresta$\}$ é um subconjunto de bicliques que atende as duas propriedades mencionadas acima. Portanto, encontrar um conjunto $\mathcal{B}'$ inicial válido é um problema que pode ser resolvido de forma eficiente e simples. A escolha do subconjunto $\mathcal{B}'$ inicial desempenha um papel bastante importante neste procedimento, pois quanto mais próximos estiverem os valores da solução do programa linear inicial e uma solução ótima, menor tende a ser o número total de iterações. Na Seção 3.2.2, descrevemos como implementamos a função ENCONTRA-BICLIQUES-INICIAIS.

Observe que $|\mathcal{B}|$ é finito e, para todo biclique B de $\mathcal{B}$, temos que a coluna correspondente à variável x_B é inserida no máximo uma vez. Portanto, o procedimento termina em um número finito de iterações. Pelo Corolário 3.2.1, temos que o algoritmo RESOLVE-PL2$_{\text{EB}}$ encontra uma solução ótima de (PL2$_{\text{EB}}$). O algoritmo RESOLVE-PL2$_{\text{EB}}$ emprega a técnica de geração de colunas a um problema específico. Para uma explanação mais geral da técnica, veja o trabalho de Barnhart *et al.* (1998).

A seguir, exploramos em mais detalhes o problema de PRICING, que desempenha um papel

fundamental no algoritmo RESOLVE-PL2$_{\text{EB}}$. Dadas uma solução x^* ótima de (PL2$_{\text{EB}}(\mathcal{B}')$) e a solução (β^*, α^*) dual complementar a x^*, dizemos que $r(x_B) = \beta^* - f(B, \alpha^*)$ é o *custo reduzido* da variável x_B, onde $f(B, \alpha^*) = |E_B| - \sum_{v \in S_B} \alpha_v^*$. Observe que se a solução (β^*, α^*) não é viável de (DPL2$_{\text{EB}}$), então existe um biclique $\hat{B}$ em $\mathcal{B} \setminus \mathcal{B}'$ tal que

$$\beta^* + \sum_{u \in S_{\hat{B}}} \alpha_u^* < |E_{\hat{B}}|$$

$$\beta^* - f(\hat{B}, \alpha^*) < 0$$

$$r(x_{\hat{B}}) < 0.$$

Ou seja, existe uma variável $x_{\hat{B}}$ com custo reduzido negativo. Por outro lado, se para todo biclique B em $\mathcal{B} \setminus \mathcal{B}'$ temos que

$$\beta^* + \sum_{u \in S_B} \alpha_u^* \geq |E_B|$$

$$\beta^* - f(B, \alpha^*) \geq 0$$

$$r(x_B) \geq 0,$$

então a solução (β^*, α^*) é viável de (DPL2$_{\text{EB}}$) e, consequentemente, a solução x^* é ótima de (PL2$_{\text{EB}}$). Considere a versão de decisão do problema de PRICING, conforme enunciamos abaixo.

Problema 3.2.2. *Dados um grafo bipartido G, um número não-negativo β^* e um vetor $\alpha^* \in \mathbb{R}^{|S|}$, tal que $\alpha \geq 0$, decidir se existe um biclique B em G, tal que $r(x_B) < 0$ (ou seja, $f(B, \alpha^*) > \beta^*$).*

A seguir, mostramos que o problema de PRICING é NP-completo, fazendo uma redução da versão de decisão do problema de BICLIQUE MÁXIMO, enunciada abaixo.

Problema 3.2.3. *Dados um grafo bipartido G e um inteito positivo t, decidir se existe um biclique B em G, tal que $|E_B| \geq t$.*

Lema 3.2.1. *O Problema 3.2.2 é NP-completo.*

Demonstração. Dada uma instância (G, t) do Problema 3.2.3, construímos uma instância (G, α, β) do Problem 3.2.2, onde $\beta = t - 1$ e $\alpha_u = 0$, para todo $u \in S$. Observe que a resposta para a instância (G, t) é *sim* se e só se a resposta para a instância (G, α, β) é *sim*. Como o Problema 3.2.3 é NP-completo, temos que o Problema 3.2.2 é NP-completo. □

A cada iteração do algoritmo RESOLVE-PL2$_{\text{EB}}$ é resolvida uma instância do problema de PRICING, que é NP-completo, conforme mostramos acima. Isto pode dar a sensação de que este procedimento será bastante ineficiente, já que não temos garantias de que o número de iterações será pequeno. Na prática, podemos utilizar alguma heurística para tentar resolver o problema de PRICING e somente quando a heurística falhar é que precisamos resolver o problema de PRICING com um algoritmo exato. Além disso, o fato de o problema de PRICING ser NP-completo não nos tira a possibilidade de desenvolver métodos exatos que obtenham bons resultados práticos para resolvê-lo.

Uma forma de resolver o problema de PRICING é encontrar um biclique B em G com custo reduzido mínimo. Caso $r(x_B)$ seja negativo, então incluímos a variável x_B na formulação. Caso contrário, podemos parar o processo de geração de colunas. Por conveniência, enunciamos o problema de encontrar um biclique B em G com custo reduzido mínimo como um problema de maximização.

Problema 3.2.4. *Dado um grafo bipartido G e um vetor $\alpha \in \mathbb{R}^{|S|}$, o problema de* CUSTO REDUZIDO MÍNIMO *consiste em encontrar um biclique B em G que maximize $f(B, \alpha)$.*

Para completar a descrição do algoritmo RESOLVE-PL2$_{\text{EB}}$, que encontra uma solução ótima de (PL2$_{\text{EB}}$), basta que apresentemos um algoritmo para resolver o problema de CUSTO REDUZIDO MÍNIMO. De forma geral, não podemos garantir que uma solução ótima de (PL2$_{\text{EB}}$) seja inteira. Como desejamos encontrar uma solução ótima de (PI2$_{\text{EB}}$), apenas o procedimento de geração de colunas que descrevemos não é suficiente. Na seção seguinte, descrevemos um procedimento de *branch-and-price*, que é uma extensão do procedimento de geração de colunas e é responsável por encontrar uma solução ótima de (PI2$_{\text{EB}}$). Para isso, faremos algumas modificações na formulação (PL2$_{\text{EB}}$). Ao considerar tais modificações, teremos um novo problema de CUSTO REDUZIDO MÍNIMO. Na seção seguinte, apresentamos um algoritmo responsável por resolver essas duas variantes do problema de CUSTO REDUZIDO MÍNIMO.

3.2.1 Um Procedimento de *Branch-And-Price*

Para resolver (PI2$_{\text{EB}}$), empregaremos um procedimento de *branch-and-bound* no qual em cada nó da árvore de busca os valores de algumas variáveis binárias são fixados em 0 ou 1 e os valores das demais variáveis são obtidos através da relaxação linear correspondente ao nó em questão. Para resolver a relaxação linear correspondente a um determinado nó, utilizamos uma versão adaptada do algoritmo de geração de colunas, apresentado na seção anterior. Esta técnica é conhecida como *branch-and-price*. Para uma introdução ao assunto, veja os trabalhos de Barnhart *et al.* (1998), Vanderbeck e Wolsey (1996) e Wolsey (1998). Uma dificuldade que surge nesse procedimento é a inconveniência de utilizar as variáveis x como base para a construção da árvore de busca. Tal inconveniente consiste no fato de que, para resolver o problema de PRICING, estaríamos interessados em encontrar um biclique de custo reduzido mínimo dentre todos os bicliques de $\mathcal{B} \setminus \{B \in \mathcal{B} \mid$ a variável x_B tem valor fixado em $0\}$. Esta, aparentemente, pequena modificação no problema de PRICING o torna muito mais difícil de resolver. Para evitar esse inconveniente, usualmente o que se faz é definir variáveis "implícitas" que permitam implementar um procedimento de *branch-and-price* sem alterar de forma significante o problema de PRICING. A seguir, descrevemos em detalhes como definimos tais variáveis.

Dado um vetor $x \in [0,1]^{|\mathcal{B}|}$, seja $\omega : [0,1]^{|\mathcal{B}|} \to \mathbb{R}$ uma função, tal que $\omega(x) = \sum_{B \in \mathcal{B}} |E_B| \cdot x_B$, para todo $x \in [0,1]^{|\mathcal{B}|}$. Ou seja, $\omega(x)$ corresponde ao valor de x na função objetivo de (PL2$_{\text{EB}}$). Dada uma solução x viável de (PL2$_{\text{EB}}$), seja $\lambda_{uv} = \sum_{B \in \mathcal{B}(u) \cap \mathcal{B}(v)} x_B$, para todo par de vértices distintos $\{u, v\}$, tal que $u, v \in S$. Interpretamos os valores de λ da seguinte maneira: $\lambda_{uv} = 1$ se e só se os vértices u e v estão contidos no mesmo biclique, considerando o empacotamento induzido pela solução x. Dizemos que as variáveis λ são implícitas porque seus valores são obtidos a partir das variáveis x, que já estão definidas na formulação e, consequentemente, não há necessidade de incluir λ na formulação. Dada uma variável λ_{uv} com valor fracionário, digamos μ, podemos criar dois ramos diferentes na árvore de busca. Em um deles, a restrição $\lambda_{uv} \leq \lfloor \mu \rfloor = 0$ deve ser satisfeita. No outro, a restrição $\lambda_{uv} \geq \lceil \mu \rceil = 1$ deve ser satisfeita. Portanto, em cada nó da árvore de busca estamos interessados em resolver o seguinte problema.

Problema 3.2.5. *Dados conjuntos $F_1 \subset S \times S$ e $F_0 \subset S \times S$, encontrar uma solução x viável de* (PL2$_{\text{EB}}$)*, tal que $\lambda_{uv} = 1$, para todo $\{u, v\}$ em F_1, $\lambda_{uv} = 0$, para todo $\{u, v\}$ em F_0, e $\omega(x)$ é*

máximo.

Esta *regra de ramificação* proposta aqui é similar à introduzida por Ryan e Foster (1981) para resolver um problema de escalonamento. No trabalho de Vanderbeck e Wolsey (1996), podemos encontrar regras de ramificação de propósito mais geral.

Dado um conjunto $F_0 \subset S \times S$, seja $\mathcal{B}_{F_0} = \{B \in \mathcal{B} \mid \{u,v\} \not\subseteq S_B$, para todo $\{u,v\} \in F_0\}$ o conjunto de todos os bicliques em G que contêm no máximo um dos vértices u e v, para todo $\{u,v\} \in F_0$. A seguir, apresentamos uma formulação para o Problema 3.2.5.

$$
\begin{aligned}
& \text{maximize} && \sum_{B \in \mathcal{B}_{F_0}} |E_B| \cdot x_B && && \\
(\text{PL3}_{\text{EB}}) \qquad & \text{sujeito a} && \sum_{B \in \mathcal{B}_{F_0}(u) \cap \mathcal{B}_{F_0}(v)} x_B \geq 1, && \text{para todo } \{u,v\} \in F_1 && (11.1)\\
& && \sum_{B \in \mathcal{B}_{F_0}} x_B \leq k && && (11.2)\\
& && \sum_{B \in \mathcal{B}_{F_0}(u)} x_B \leq 1, && \text{para todo } u \in S && (11.3)\\
& && x_B \geq 0, && \text{para todo } B \in \mathcal{B}_{F_0} && (11.4)
\end{aligned}
$$

As restrições (11.2), (11.3) e (11.4) são as mesmas definidas na formulação (PL2_{EB}). Como a restrição $\lambda_{uv} = 0$, para todo $\{u,v\} \in F_0$, deve ser satisfeita, temos que $\sum_{B \in \mathcal{B} \setminus \mathcal{B}_{F_0}} x_B = 0$ em qualquer solução viável x. Por isso, incluímos na formulação (PL3_{EB}) apenas as variáveis correspondentes aos bicliques em $\mathcal{B}_{F_0}$. Por simplicidade, dada uma solução x viável de (PL3_{EB}), assumimos que $x \in [0,1]^{|\mathcal{B}|}$ e $x_B = 0$, para todo $B \in \mathcal{B} \setminus \mathcal{B}_{F_0}$. Pelas restrições (11.1) e (11.3), temos que $\lambda_{uv} = 1$, para todo $\{u,v\} \in F_1$. Portanto, (PL3_{EB}) é uma formulação para o Problema 3.2.5.

Dada uma solução x^* ótima de (PL3_{EB}), se x^* é inteira, então não há necessidade de criar novos ramos a partir deste nó que está sendo explorado na árvore de busca. Caso contrário, precisamos selecionar uma variável λ_{uv} com valor fracionário, para que possamos criar novos ramos a partir deste nó. No Lema 3.2.4, mostramos que se x^* é fracionária, então existe pelo menos uma variável λ_{uv} com valor fracionário. Ou seja, mostramos que nossa regra de ramificação nos conduz a uma solução ótima do problema de EMPACOTAMENTO DE BICLIQUES. Antes de mostrar o Lema 3.2.4, mostramos alguns lemas auxiliares.

Lema 3.2.2. *Dada uma solução x^* ótima de (PL3_{EB}), se x^* é fracionária então x^* possui pelo menos dois elementos com valores fracionários.*

Demonstração. Suponha, por absurdo, que x^* tem exatamente um elemento $x^*_{B'}$ com valor fracionário. Como as restrições (11.1) são satisfeitas com igualdade, temos que $\{u,v\} \not\subseteq S'_B$, para todo $\{u,v\} \in F_1$. Como todos os coeficientes das inequações (11.2) e (11.3) são inteiros, temos que $\sum_{B \in \mathcal{B}} x^*_B < k$ e $\sum_{B \in \mathcal{B}(u)} x^*_B = x^*_{B'}$, para todo $u \in S_{B'}$. Seja $x' \in [0,1]^{\mathcal{B}}$ um vetor, tal que $x'_B = x^*_B$, para todo $B \in \mathcal{B} \setminus \{B'\}$, e $x'_{B'} = x^*_{B'} + \epsilon$, onde $\epsilon = \min\{1 - x^*_{B'}, k - \sum_{B \in \mathcal{B}} x^*_B\}$. Observe que x' é uma solução viável de (PL3_{EB}), tal que $\omega(x') > \omega(x^*)$, o que é uma contradição. □

Lema 3.2.3. *Seja x^* uma solução ótima de (PL3_{EB}), tal que para cada par de variáveis distintas $x^*_{B_1}$ e $x^*_{B_2}$ com valores fracionários, temos que $S_{B_1} \cap S_{B_2} = \emptyset$ ou $S_{B_1} = S_{B_2}$. Então, existe uma solução inteira x^{**} ótima de (PL3_{EB}).*

Demonstração. Seja $\Psi : \mathbb{R}_+^{|\mathcal{B}|} \times \mathcal{B} \times \mathcal{B} \times \mathbb{R}_+ \to \mathbb{R}_+^{|\mathcal{B}|}$ uma função, tal que $\Psi(x, B_1, B_2, \epsilon) = x'$, onde $x'_{B_1} = x_{B_1} + \epsilon$, $x'_{B_2} = x_{B_2} - \epsilon$ e $x'_B = x_B$, para todo $B \in \mathcal{B} \setminus \{B_1, B_2\}$. Provamos o lema por indução em n, onde n é o número de variáveis com valor fracionário. Se $n = 0$, não há o que provar. Suponha que $n > 0$. Pelo Lema 3.2.2, temos que $n \geq 2$. Seja $x^*_{B_{max}}$ uma variável com valor fracionário, tal que $|E_{B_{max}}|$ é máximo. Pela hipótese do lema, um dos seguintes casos ocorre: (i) $S_{B_{max}} = S_{B_{eq}}$, para alguma outra variável $x^*_{B_{eq}}$ com valor fracionário; (ii) $S_{B_{max}} \cap S_B = \emptyset$, para cada uma das outras variáveis x^*_B, diferentes de $x^*_{B_{max}}$, com valor fracionário.

No caso (i), considere o vetor $x' = \Psi(x^*, B_{max}, B_{eq}, \epsilon)$, onde $\epsilon = \min\{1 - x^*_{B_{max}}, x^*_{B_{eq}}\}$. Como $S_{B_{max}} = S_{B_{eq}}$, então as variáveis $x^*_{B_{max}}$ e $x^*_{B_{eq}}$ possuem coeficientes não nulos nas mesmas restrições. Consequentemente, x' é uma solução viável de (PL2_{EB}). Observe que, como $x'_{B_{max}} = 1$ ou $x'_{B_{eq}} = 0$, temos que x' tem menos do que n elementos com valores fracionários. Além disso, pela escolha de B_{max}, segue que $\omega(x') \geq \omega(x^*)$, o que implica em $\omega(x') = \omega(x^*)$, já que x^* é uma solução ótima. Portanto, por hipótese de indução, existe uma solução inteira x^{**} viável de (PL3_{EB}), tal que $\omega(x^{**}) = \omega(x') = \omega(x^*)$.

No caso (ii), seja $x^*_{B_{neq}}$ uma outra variável, diferente de $x^*_{B_{max}}$, com valor fracionário. Como a propriedade (ii) é satisfeita e as restrições (11.1) são satisfeitas com igualdade, temos que a variável $x^*_{B_{max}}$ tem coeficiente nulo em todas as restrições (11.1). Pela propriedade (ii), temos que o valor da variável de folga correspondente ao vértice u na restrição (11.3) é $1 - x^*_{B_{max}}$, para todo $u \in S_{B_{max}}$. Portanto, a solução $x' = \Psi(x^*, B_{max}, B_{neq}, \epsilon)$ é viável de (PL3_{EB}), onde $\epsilon = \min\{1 - x^*_{B_{max}}, x^*_{B_{neq}}\}$. Observe que $x'_{B_{max}} = 1$ ou $x'_{B_{neq}} = 0$ e, consequentemente, o número de variáveis fracionárias em x' é menor do que em x^*. Pela escolha de B_{max} e usando a hipótese de indução, temos que existe uma soluação inteira x^{**} viável de (PL3_{EB}), tal que $\omega(x^{**}) = \omega(x') = \omega(x^*)$. □

Lema 3.2.4. *Dada uma solução x^* ótima de (PL3_{EB}), temos que existe uma variável λ_{uv} com valor fracionário ou então existe uma solução inteira x^{**} ótima de (PL3_{EB}).*

Demonstração. Se x^* é inteira então não há o que provar. Suponha que x^* é fracionária. Pelo Lema 3.2.2, existem pelo menos dois elementos com valores fracionários na solução x^*. Portanto, um dos seguintes casos ocorre: (i) para cada par de variáveis distintas $x^*_{B_1}$ e $x^*_{B_2}$ com valores fracionários, temos que $S_{B_1} \cap S_{B_2} = \emptyset$ ou $S_{B_1} = S_{B_2}$; (ii) existem duas variáveis distintas $x^*_{B_1}$ e $x^*_{B_2}$ com valores fracionários, tais que $u \in S_{B_1} \cap S_{B_2}$ e $v \in S_{B_1} \nabla S_{B_2}$, para algum para de vértices $u, v \in S$, onde $S_{B_1} \nabla S_{B_2} = (S_{B_1} \cup S_{B_2}) \setminus (S_{B_1} \cap S_{B_2})$.

No caso (i), pelo Lema 3.2.3, temos que existe uma solução inteira x^{**} ótima de (PL3_{EB}). No caso (ii), sem perda de generalidade, assuma que $v \in S_{B_1}$ e $v \notin S_{B_2}$. Como $B_1 \in \mathcal{B}(u) \cap \mathcal{B}(v)$, então $\sum_{B \in \mathcal{B}(u) \cap \mathcal{B}(v)} x^*_B \geq x^*_{B_1} > 0$. Como $B_2 \notin \mathcal{B}(u) \cap \mathcal{B}(v)$, $B_2 \in \mathcal{B}(u)$ e $\sum_{B \in \mathcal{B}(u)} x^*_B \leq 1$, então temos que $\sum_{B \in \mathcal{B}(u) \cap \mathcal{B}(v)} x^*_B \leq \sum_{B \in \mathcal{B}(u)} x^*_B - x^*_{B_2} < 1$. Consequentemente, temos que $0 < \lambda_{uv} < 1$. □

Do Lema 3.2.4, concluímos que nossa regra de ramificação nos conduz a uma solução ótima do problema de EMPACOTAMENTO DE BICLIQUES. Usualmente, os resolvedores de Programação Linear implementam o método *simplex*. Por isso, as soluções ótimas encontradas são sempre vértices dos respectivos poliedros. Neste caso, não há necessidade de aplicar o Lema 3.2.3, pois sempre que obtivermos uma solução fracionária, cairemos no caso (ii) do Lema 3.2.4. Analisando a prova do Lema 3.2.4, observa-se que no caso em que $S = V$, é possível escolher λ_{uv} fracionário com $uv \in E$. Por conveniência, no caso em que $S = V$, definimos λ_{uv} apenas para toda aresta $uv \in E$.

Como as formulações (PL2_{EB}) e (PL3_{EB}) não são iguais, temos que considerar quais modificações devem ser feitas no problema de PRICING. Abaixo, mostramos o programa linear dual de (PL3_{EB}).

$$
\begin{aligned}
& & \text{minimize} \quad & \sum_{u \in S} \alpha_u + k\beta - \sum_{\{u,v\} \in F_1} \gamma_{uv} & & & & \\
(\text{DPL3}_{\text{EB}}) \qquad & & \text{sujeito a} \quad & \beta + \sum_{u \in S_B} \alpha_u - \sum_{\{u,v\} \in F_1 | u,v \in S_B} \gamma_{uv} & \geq & |E_B|, & \forall B \in \mathcal{B}_{F_0} & \quad (12.1) \\
& & & \alpha_u & \geq & 0, & \forall u \in S & \quad (12.2) \\
& & & \gamma_{uv} & \geq & 0, & \forall \{u,v\} \in F_1 & \quad (12.3) \\
& & & \beta & \geq & 0 & & \quad (12.4)
\end{aligned}
$$

O vetor $\gamma \in \mathbb{R}^{|F_1|}$ de variáveis duais é associado às restrições (11.1) de (PL3_{EB}). A variável dual β e o vetor de variáveis duais $\alpha \in \mathbb{R}^{|S|}$ são associados às restrições (11.2) e (11.3) de (PL3_{EB}), respectivamente. Considerando (PL3_{EB}) e seu dual (DPL3_{EB}), temos que o custo reduzido de uma variável x_B é dado por

$$r'(x_B) = \beta + \sum_{v \in S_B} \alpha_v - |E_B| - \sum_{\{u,v\} \in F_1 | u,v \in S_B} \gamma_{uv}.$$

Seja $\Phi : \mathcal{B} \times \mathbb{R}^{|S|} \times \mathbb{R}^{|F_1|} \to \mathbb{R}$ uma função, tal que

$$\Phi(B, \alpha, \gamma) = |E_B| + \sum_{\{u,v\} \in F_1 | u,v \in S_B} \gamma_{uv} - \sum_{u \in S_B} \alpha_u.$$

Por conveniência, abaixo enunciamos o problema de encontrar uma variável de custo reduzido mínimo como um problema de maximização.

Problema 3.2.6. *Dado um grafo bipartido G, dois conjuntos $F_1 \subset S \times S$ e $F_0 \subset S \times S$, e dois vetores $\alpha \in \mathbb{R}^{|S|}$ e $\gamma \in \mathbb{R}^{|F_1|}$, tais que $\alpha \geq 0$ e $\gamma \geq 0$, o problema de* CUSTO REDUZIDO MÍNIMO *consiste em encontrar um biclique B contido em $\mathcal{B}_{F_0}$, tal que $\Phi(B, \alpha, \gamma)$ é máximo.*

No caso em que $F_1 = F_0 = \emptyset$, as duas variantes do problema de CUSTO REDUZIDO MÍNIMO (Problemas 3.2.4 e 3.2.6) são equivalentes, sendo que o Problema 3.2.4 é um caso particular do Problema 3.2.6. Daqui em diante, quando nos referirmos ao "problema de CUSTO REDUZIDO MÍNIMO", estaremos nos referindo ao Problema 3.2.6.

Observe que o problema de BICLIQUE MÁXIMO é um caso especial do problema de CUSTO REDUZIDO MÍNIMO em que $\alpha_v = 0$, para todo $v \in S$, e $F_1 = F_0 = \emptyset$. Esta semelhança entre esses dois problemas indica que pode haver também uma semelhança na forma de resolvê-los. Como o algoritmo de *branch-and-bound* apresentado na Seção 2.4 mostrou melhores resultados na resolução do problema de BICLIQUE MÁXIMO, faremos uma adaptação deste algoritmo para resolver o problema de CUSTO REDUZIDO MÍNIMO.

Daqui em diante, é conveniente distinguir os casos em que $S = L$, $S = R$ e $S = V$. Conforme mencionamos anteriormente, no caso em que $S = V$, para todo $\{u, v\}$ em $F_1 \cup F_0$, temos que u e v estão em classes distintas. Nos casos em que $S = L$ ou $S = R$, para todo $\{u, v\}$ em $F_1 \cup F_0$, temos que u e v estão na mesma classe. Dada uma instância do problema de CUSTO REDUZIDO MÍNIMO, no caso em que $S = V$, fazemos o seguinte pré-processamento: para todo par $\{u, v\}$ em F_0, removemos

a aresta uv de E. Observe que os únicos bicliques que deixam de existir em G após a remoção de uma dessas arestas uv são os bicliques que contêm uv. Portanto, tal pré-processamento não elimina nenhuma solução viável. Portanto, na resolução do problema de CUSTO REDUZIDO MÍNIMO, no caso em que $S = V$, podemos assumir que $F_0 = \emptyset$. Antes de apresentar nosso algoritmo de *branch-and-bound* para resolver o problema de CUSTO REDUZIDO MÍNIMO, introduzimos a notação que será utilizada e mostramos algumas propriedades importantes.

Dados um vértice u e um subconjunto de vértices V', seja $\Gamma(u, V') = \sum_{v \in V'} \gamma_{uv}$. Um subconjunto L' de L é dito *válido* se, para todo $\{u, v\}$ em F_0, vale que $\{u, v\} \not\subseteq L'$. Dado um subconjunto válido L' de L, seja $\mathcal{N}(L') = \{R' \subseteq N^{\cap}(L') \mid (L', R') \in \mathcal{B}_{F_0}\}$ e seja $N^*(L') = \arg\max_{R' \in \mathcal{N}(L')} \Phi((L', R'), \alpha, \gamma)$. Observe que, dado um vértice $v \in N^{\cap}(L')$, a contribuição do vértice v na função objetivo é $|L'|$, no caso em que $S = L$, ou $|L'| - \alpha_v + \Gamma(v, L')$, no caso em que $S = V$. Portanto, se $S = L$, temos que $N^*(L') = N^{\cap}(L')$, e se $S = V$, temos que $N^*(L') = \{v \in N^{\cap}(L') \mid |L'| - \alpha_v + \Gamma(v, L') > 0\}$. No caso em que $S = R$ e $F_1 = \emptyset$, temos que $N^*(L')$ é um conjunto independente de peso máximo no grafo $D = (V_D, E_D)$, onde $V_D = \{v \in N^{\cap}(L') \mid \alpha_v < |L'|\}$, $E_D = \{uv \mid \{u, v\} \in F_0\}$ e $peso(v) = |L'| - \alpha_v$, para todo $v \in V_D$. Como o problema de encontrar um conjunto independente de peso máximo em um grafo é NP-difícil, o conjunto $N^*(L')$ não pode ser construído de forma simples e eficiente como nos casos anteriores. Por essa razão, daqui em diante assumimos que $S = L$ ou $S = V$. No caso em que $S = R$, podemos inverter os papeis de L e R. O único inconveniente é que neste caso perdemos a propriedade de que $|L| \leq |R|$, o que deve causar um impacto negativo no tempo de processamento do algoritmo.

Nosso algoritmo de *branch-and-bound* explora o espaço de busca que corresponde a todos os possíveis subconjuntos válidos de L. Dados um subconjunto válido L' de L e um conjunto de vértices $C = \{u_1, u_2, \ldots, u_{|C|}\}$, tal que $C \subseteq L \setminus L'$, sejam $R' = \{v_1, \cdots, v_{|R'|}\} = N^{\cap}(L')$ e $\mathcal{B}_{F_0}(L', C) = \{B \in \mathcal{B}_{F_0} \mid L_B = L' \cup C'$, para algum $C' \subseteq C\}$. Sem perda de generalidade, assumimos que $|N(u_l) \cap R'| \leq |N(u_{l+1}) \cap R'|$, para $l = 1, \ldots, |C| - 1$, e $\alpha_{v_i} - \Gamma(v_i, L' \cup C) \leq \alpha_{v_{i+1}} - \Gamma(v_{i+1}, L' \cup C)$, para $i = 1, \ldots, |R'| - 1$. Denotamos por C_l o subconjunto $\{u_l, \ldots, u_{|C|}\}$ de C. A seguir, mostramos um limitante superior para o valor de $\Phi(B, \alpha, \gamma)$, para qualquer biclique B contido em $\mathcal{B}_{F_0}(L', C)$.

Lema 3.2.5. *O valor de* $LimSup(L', C) = \max\left\{\Phi((L', N^*(L')), \alpha, \gamma), \max_{1 \leq l \leq |C|} f(l)\right\}$ *é um limitante superior para o valor de* $\Phi(B, \alpha, \gamma)$*, para qualquer biclique* B *contido em* $\mathcal{B}_{F_0}(L', C)$*, onde*

$$f(l) = \begin{cases} (|L'| + |I(L', l)|) \cdot |N(u_l) \cap R'| - \displaystyle\sum_{u \in L' \cup I(L', l)} \left(\alpha_u - \frac{\Gamma(u, L' \cup C_l)}{2}\right), & \text{se } S = L \\ (|L'| + |C| - l + 1) \cdot T(l) - \displaystyle\sum_{u \in L'} \alpha_u - \sum_{i=1}^{T(l)} \left(\alpha_{v_i} - \Gamma(v_i, L' \cup C)\right), & \text{se } S = V \end{cases}$$

$$I(L', l) = \{u \in C_l \mid \alpha_u - \frac{\Gamma(u, L' \cup C_l)}{2} \leq |N(u_l) \cap R'|\},$$

$$T(l) = \min\{|N(u_l) \cap R'|, |\{v \in R' \mid \alpha_v - \Gamma(v, L' \cup C) \leq (|L'| + |C| - l + 1)\}|\}.$$

Demonstração. Seja B^* um biclique contido em $\mathcal{B}_{F_0}(L', C)$, tal que $\Phi(B^*, \alpha, \gamma)$ seja máximo. Como $LimSup(L', C) \geq \Phi((L', N^*(L')), \alpha, \gamma)$, no caso em que $L_{B^*} = L'$, temos que $LimSup(L', C) \geq \Phi(B^*, \alpha, \gamma)$. Suponha então que $L_{B^*} \supset L'$. Sejam l o menor índice, tal que $1 \leq l \leq |C|$ e $u_l \in L_{B^*}$, e C'_l o subconjunto de C_l tal que que $L_{B^*} = L' \cup C'_l$. Analisaremos os casos em que $S = L$ e $S = V$.

Suponha que $S = L$. Como $u_l \in L_{B^*}$, temos que $R_{B^*} \subseteq (N(u_l) \cap R')$ e, consequentemente,

$|R_{B^*}| \leq |N(u_l) \cap R'|$. Como $\Phi(B^*, \alpha, \gamma)$ é máximo, temos que $C'_l \subseteq \{u \in C_l \mid \alpha_u - \frac{\Gamma(u, L' \cup C'_l)}{2} \leq |R_{B^*}|\}$. Como $\Gamma(u, L' \cup C'_l) \leq \Gamma(u, L' \cup C_l)$ e $|R_{B^*}| \leq |N(u_l) \cap R'|$, temos que $C'_l \subseteq I(L', l)$ e, consequentemente, $|L_{B^*}| \leq |L'| + |I(L', l)|$ e $\sum_{u \in I(L',l)} \left(\alpha_u - \frac{\Gamma(u, L' \cup C_l)}{2}\right) \leq \sum_{u \in C'_l} \left(\alpha_u - \frac{\Gamma(u, L' \cup C'_l)}{2}\right)$. Portanto, temos que

$$\Phi(B^*, \alpha, \gamma) = (|L'| + |C'_l)|) \cdot |R_{B^*}| - \sum_{u \in L' \cup C'_l} \left(\alpha_u - \frac{\Gamma(u, L' \cup C'_l)}{2}\right) \leq$$

$$(|L'| + |I(L', l)|) \cdot |N(u_l) \cap R'| - \sum_{u \in L' \cup I(L',l)} \left(\alpha_u - \frac{\Gamma(u, L' \cup C_l)}{2}\right) \leq LimSup(L', C),$$

como queríamos mostrar (o termo $\frac{1}{2}$ é utilizado na equação acima para que cada termo γ_{uv}, tal que $u, v \in L_{B^*}$, seja somado uma única vez no valor de $\Phi(B^*, \alpha, \gamma)$).

Suponha que $S = V$. Como $C'_l \subseteq C_l$, temos que $|L'| + |C'_l| \leq |L'| + |C_l| = |L'| + |C| - l + 1$. Como $\Phi(B^*, \alpha, \gamma)$ é máximo, temos que $R_{B^*} \subseteq \{v \in N^{\cap}(L_{B^*}) \mid \alpha_v - \Gamma(v, L' \cup C'_l) \leq (|L'| + |C'_l|)\}$. Como $\Gamma(v, L' \cup C) \geq \Gamma(v, L' \cup C'_l)$ e $|L'| + |C'_l| \leq |L'| + |C| - l + 1$, temos que $\{v \in N^{\cap}(L_{B^*}) \mid \alpha_v - \Gamma(v, L' \cup C'_l) \leq (|L'| + |C'_l|)\} \subseteq \{v \in R' \mid \alpha_v - \Gamma(v, L' \cup C) \leq (|L'| + |C| - l + 1)\}|\}$ e, consequentemente, $|R_{B^*}| \leq T(l)$. Como $\alpha_{v_i} - \Gamma(v_i, L' \cup C) \leq \alpha_{v_{i+1}} - \Gamma(v_{i+1}, L' \cup C)$, para $i = 1, \ldots, |R'| - 1$, temos que $\sum_{i=1}^{T(l)} (\alpha_{v_i} - \Gamma(v_i, L' \cup C)) \leq \sum_{v \in R_{B^*}} (\alpha_v - \Gamma(v, L' \cup C)) \leq \sum_{v \in R_{B^*}} (\alpha_v - \Gamma(v, L' \cup C'_l))$. Portanto, temos que

$$\Phi(B^*, \alpha, \gamma) = (|L'| + |C'_l)|) \cdot |R_{B^*}| - \sum_{u \in L' \cup C'_l} \alpha_u - \sum_{v \in R_{B^*}} \left(\alpha_v - \Gamma(v, L' \cup C'_l)\right) \leq$$

$$(|L'| + |C| - l + 1) \cdot T(l) - \sum_{u \in L'} \alpha_u - \sum_{i=1}^{T(l)} \left(\alpha_{v_i} - \Gamma(v_i, L' \cup C)\right) \leq LimSup(L', C),$$

como queríamos mostrar.

Concluímos, portanto, que o valor de $LimSup(L', C)$ é um limitante superior para o valor de $\Phi(B, \alpha, \gamma)$, para qualquer biclique B contido em $\mathcal{B}_{F_0}(L', j)$. □

Nosso algoritmo de *branch-and-bound* faz a enumeração implícita de todos os subconjuntos válidos de L, no sentido de que com o uso do Lema 3.2.5 alguns subconjuntos não precisam ser enumerados explicitamente durante o processo de enumeração. Dividimos o procedimento em três subrotinas. O Algoritmo 7 apenas inicializa as variáveis globais utilizadas durante o procedimento e faz a primeira chamada ao Algoritmo 8, que por sua vez é responsável pelo procedimento de *branch-and-bound* propriamente dito.

Algoritmo 7: RESOLVE-PRICING(β, α, G)

Entrada: um grafo bipartido G, um número β e um vetor α.
Saída: resolve o problema de PRICING.

```
1 L' ← ∅
2 B* ← (∅, ∅)
3 B&B-PRICING(L)
4 se φ(B*, α) ≤ β então
5 |   devolva NULL
6 fim
7 devolva B*
```

Algoritmo 8: B&B-PRICING(C)

```
1  u ← arg min_{u∈C} |N(u) ∩ N^∩(L')|
2  L' ← L' ∪ {u}
3  P ← {v ∈ C | {u, v} ∈ F_0}
4  C ← C \ (P ∪ {u})
5  Seja B' = (L', N*(L'))
6  se φ(B', α) > φ(B*, α) então
7  |   B* ← B'
8  fim
9  se C ≠ ∅ e LimSup(L', C) > β e LimSup(L', C) > φ(B*, α) então
10 |   B&B-PRICING(C)
11 fim
12 L' ← L' \ {u}
13 C ← C ∪ P
14 se C ≠ ∅ e LimSup(L', C) > β e LimSup(L', C) > φ(B*, α) então
15 |   B&B-PRICING(C)
16 fim
```

Na linha 2 do Algoritmo 7, podemos inicializar a variável B^* com qualquer biclique de B_{F_0}. O ideal é que se disponha de uma boa heurística, de forma que o valor de $\phi(B^*, \alpha)$ seja bastante próximo do valor ótimo. Em nossa implementação, utilizamos a heurística descrita na Seção 3.2.2. Na linha 5 do Algoritmo 8, o conjunto $N^*(L')$ pode ser construído de forma simples e eficiente, conforme descremos anteriormente. Nas linhas 9 e 14 do Algoritmo 8, o teste $LimSup(L', C) > \beta$ é feito porque estamos interessados apenas em bicliques com custo reduzido negativo.

3.2.2 Considerações Sobre a Implementação

Nas seções anteriores, descrevemos como resolver o problema de EMPACOTAMENTO DE BICLIQUES através de um procedimento de *branch-and-price*, que combina as técnicas de *branch-and-bound* e geração de colunas. A seguir, abordamos sucintamente algumas questões importantes com relação à implementação deste procedimento.

Como encontrar um subconjunto de variáveis a ser incluído no programa linear inicial

No algoritmo de geração de colunas, iniciamos o procedimento com um programa linear restrito a um subconjunto de variáveis e a cada iteração, após resolver o programa linear, incluímos uma nova variável com custo reduzido negativo, considerando os valores das variáveis duais. Para que possamos dar início a este procedimento, o programa linear inicial deve ser viável e em cada restrição deve haver pelo menos uma variável com coeficiente não nulo. Diremos que $(\text{PL2}_{\text{EB}}(\mathcal{B}'))$ é válido se essas duas condições foram satisfeitas. Além disso, a escolha do subconjunto de variáveis iniciais desempenha um papel bastante importante neste procedimento, pois quanto mais próximos estiverem os valores de uma solução ótima do programa linear inicial e uma solução ótima do programa linear completo, menor tende a ser o número total de iterações. Em nossa implementação, optamos por fazer a seguinte heurística.

Algoritmo 9: ENCONTRA-BICLIQUES-INICIAIS(G)

```
1  B' ← ∅
2  Seja G' = (V', E') uma cópia do grafo G = (V, E)
3  enquanto E' ≠ ∅ faça
4  |  B&B(G') /* encontra um biclique máximo em G' */
5  |  B' ← B' ∪ {B}
6  |  G' ← G'[V' \ S_B]
7  fim
8  Seja S' o subconjunto de vértices de S que não estão contidos em nenhum biclique de B'
9  enquanto S' ≠ ∅ faça
10 |  escolha uma aresta uv ∈ E tal que {u, v} ∩ S' ≠ ∅
11 |  B' ← B' ∪ G[{u, v}]
12 |  S' ← S' \ {u, v}
13 fim
14 devolva B'
```

Na linha 4 do Algoritmo 9, podemos utilizar a heurística apresentada na Seção 2.5 para encontrar um biclique, não necessariamente máximo, em G'. Observe que após o laço das linhas 3 a 7 o subconjunto dos k primeiros bicliques inseridos em $\mathcal{B}'$ é um empacotamento de bicliques em G. Consequentemente, além de obter um subconjunto de bicliques iniciais tal que $(\text{PL2}_{\text{EB}}(\mathcal{B}'))$ é válido, obtemos também uma solução viável do problema de EMPACOTAMENTO DE BICLIQUES, que pode ser usada como um limitante inferior no valor de uma solução ótima. Chamaremos esta heurística de encontrar um empacotamento viável de HEURÍSTICA-GULOSA. Uma outra vantagem do Algoritmo 9 é que, em muitos casos, o valor de uma solução ótima do programa linear inicial está bastante próximo do valor de uma solução ótima do programa linear completo, conforme mostramos na Seção 3.3.

Observe que precisamos tratar também o problema de encontrar um subconjunto de variáveis a ser incluído no programa linear inicial, agora considerando a formulação $(\text{PL3}_{\text{EB}}(\mathcal{B}'))$, que corresponde à formulação (PL3_{EB}) restrita às variáveis correspondentes ao subconjunto $\mathcal{B}'$. O que difere $(\text{PL3}_{\text{EB}}(\mathcal{B}'))$ de $(\text{PL2}_{\text{EB}}(\mathcal{B}'))$ é que em $(\text{PL3}_{\text{EB}}(\mathcal{B}'))$ são incluídas as restrições (11.1) $(\sum_{B \in \mathcal{B}_{F_0}(u) \cap \mathcal{B}_{F_0}(v)} x_B \geq 1$, para todo $\{u, v\} \in F_1)$ e as variáveis são definidas apenas para os bi-

cliques de $\mathcal{B}_{F_0}$. Dependendo da escolha de F_1, o programa linear ($\mathrm{PL3}_{\mathrm{EB}}$) pode ser inviável. Portanto, nem sempre é possível encontrar um programa linear ($\mathrm{PL3}_{\mathrm{EB}}(\mathcal{B}')$) inicial válido. O que fazemos neste caso é introduzir variáveis "artificiais" formando uma matriz identidade, considerando a matriz de coeficientes das restrições (11.1), sendo que tais variáveis possuem um coeficiente suficientemente pequeno na função objetivo ($-k \cdot |E|$, por exemplo). Dessa forma, se alguma das variáveis artificiais possuirem valor positivo na solução ótima encontrada pelo procedimento de geração de colunas, então temos que ($\mathrm{PL3}_{\mathrm{EB}}$) é inviável.

Heurísticas para resolver o problema de PRICING

Não há necessidade de encontrar uma variável com custo reduzido mínimo a cada iteração do algoritmo de geração de colunas. Basta que encontremos uma variável com custo reduzido negativo. Além disso, podemos gerar mais de uma nova coluna a cada iteração. Em nossa implementação, utilizamos a heurística HEUR-PRICING descrita abaixo para tentar encontrar um conjunto de variáveis com custo reduzido negativo. Só executamos o algoritmo B&B-PRICING quando a heurística HEUR-PRICING não encontra nenhuma variável com custo reduzido negativo.

Algoritmo 10: HEUR-PRICING(C)

```
1  B' ← ∅
2  para todo u' ∈ L faça
3  |  L' ← {u'}
4  |  C ← L \ ({u'} ∪ {v ∈ C | {u', v} ∈ F_0})
5  |  Seja B' = (L', N*(L'))
6  |  se φ(B', α) > β então
7  |  |  B ← B ∪ {B'}
8  |  fim
9  |  enquanto C ≠ ∅ e N∩(L') ≠ ∅ faça
10 |  |  u ← arg max_{u∈C} Φ(L' ∪ {u}, α, β)
11 |  |  L' ← L' ∪ {u}
12 |  |  C ← C \ ({u} ∪ {v ∈ C | {u, v} ∈ F_0})
13 |  |  Seja B' = (L', N*(L'))
14 |  |  se φ(B', α) > β então
15 |  |  |  B ← B ∪ {B'}
16 |  |  fim
17 |  fim
18 fim
19 devolva B
```

Reaproveitamento das variáveis incluídas

A cada iteração do procedimento de *branch-and-price* escolhemos algum nó da árvore de busca a ser explorado. Em vez de começarmos todo o procedimento de geração de colunas desde o início, para o nó em questão, podemos reaproveitar muitas das variáveis incluídas anteriormente, considerando o último programa linear resolvido. Quando o conjunto F_0 referente ao nó que está sendo explorado

não possui novos elementos, com relação ao conjunto F_0 do último programa linear resolvido, podemos reaproveitar todas as variáveis de tal programa linear. Já no caso em que o novo conjunto F_0 possui novos elementos, reaproveitamos apenas as variáveis correspondentes ao subconjunto $\mathcal{B}' \cap \mathcal{B}_{F_0}$.

Gerenciamento da quantidade de variáveis incluídas ao longo do procedimento

Quanto maior for o tamanho de $\mathcal{B}'$, maior tende a ser o consumo de tempo e memória necessários para resolver $(\mathrm{PL3}_{\mathrm{EB}}(\mathcal{B}'))$. Para evitar o consumo excessivo de memória e tempo de processamento, toda vez que $|\mathcal{B}'|$ exceder um valor pré-estipulado, removemos algumas das variáveis que possuem os custos reduzidos mais altos. É importante estabelecer corretamente a quantidade máxima de variáveis incluídas, pois caso o valor desse parâmetro não seja grande o suficiente, corre-se o risco do algoritmo nunca parar. Por outro lado, se esse valor for muito alto, não obteremos o efeito desejado.

Heurísticas para obter limitantes superiores

A qualquer momento podemos obter um empacotamento de bicliques em G através de uma solução ótima de $(\mathrm{PI2}_{\mathrm{EB}})$ restrito às variáveis correspondentes a $\mathcal{B}'$. Como observamos em nossos experimentos computacionais, tal solução pode ser obtida com pouco tempo de processamento, pois o valor de uma solução ótima de $(\mathrm{PL2}_{\mathrm{EB}}(\mathcal{B}'))$ muitas vezes está bastante próximo do valor de uma solução ótima do programa inteiro restrito. Antes de executarmos este procedimento, realizamos o seguinte pré-processamento: para cada par de variáveis x_{B_1} e x_{B_2} com valor positivo na solução da relaxação linear atual, tal que $S_{B_1} \cap S_{B_2} \neq \emptyset$, adicionamos em $\mathcal{B}'$ os bicliques B_1' e B_2', onde $V_{B_1'} = V_{B_1} \setminus (S_{B_1} \cap S_{B_2})$ e $V_{B_2'} = V_{B_2} \setminus (S_{B_1} \cap S_{B_2})$. Para cada par de variáveis selecionado, sabemos que nenhum empacotamento possui ambos os bicliques B_1 e B_2. Por isso, incluímos os bicliques B_1' e B_2', que podem estar contidos num mesmo empacotamento que B_2 e B_1, respectivamente. Nos referimos a este procedimento, quando aplicado à raiz da árvore de busca (ou seja, quando terminamos o processo de geração de colunas para $F_0 = F_1 = \emptyset$), como HEURÍSTICA-DA-SOLUÇÃO-INTEIRA-RESTRITA. Conforme mostramos em nossos experimentos computacionais, em muitos casos a solução encontrada por essa heurística é ótima.

Seleção do próximo nó a ser explorado e criação dos novos ramos

Uma dificuldade em termos de implementação é que os resolvedores de Programação Linear Inteira não disponibilizam bibliotecas que auxiliem na implementação do método *branch-and-price*. Existem alguns *frameworks*, como por exemplo o SCIP (http://scip.zib.de/), que fazem isso e podem ser utilizados em conjunto com os principais resolvedores de Programação Linear Inteira. Porém, esses *frameworks* ainda não dispõem de boa documentação e não são fáceis de usar. Por isso, decidimos implementar nosso algoritmo sem o auxílio de nenhum *framework*.

Como estamos tratando de um problema de maximização, a política de seleção de nós da árvore de busca afeta mais diretamente o limitante superior global. Optamos por selecionar um nó com o maior limitante superior. Esta escolha visa tentar diminuir o mais rápido possível o limitante superior global. No momento de criar novos ramos na árvore de busca, escolhemos uma variável

implícita com valor mais próximo de $\frac{1}{2}$. Com isso, verificamos empiricamente que a altura da árvore de busca tende a não ser muito grande.

Controlando o efeito de *tailling-off*

Muitas vezes, grande parte das últimas variáveis incluídas durante o processo de geração de colunas não resulta em um ganho significativo na função objetivo. Este sintoma é conhecido na literatura como efeito de *tailling-off*. Além disso, quanto mais o processo de geração de colunas se aproxima do final, mais difícil tende a ser o problema de PRICING. A seguir, descrevemos uma forma de tratar esse problema.

Dada uma solução x viável de (PL3_{EB}) e sua correspondente solução dual complementar (β, α, γ), seja B^* um biclique de $\mathcal{B}_{F_0}$ com custo reduzido $r'(B^*)$ mínimo. Seja $z(\beta, \alpha, \gamma) = \sum_{u \in S} \alpha_u + k\beta - \sum_{\{u,v\} \in F_1} \gamma_{uv}$ o valor de (β, α, γ) na função objetivo de (DPL3_{EB}). Observe que, para todo biclique B contido em $\mathcal{B}_{F_0}$, vale que

$$-r'(B) = \beta + \sum_{u \in S_B} \alpha_u - \sum_{\{u,v\} \in F_1 | u,v \in S_B} \gamma_{uv} - |E_B| \geq -r'(B^*),$$

e, consequentemente, temos que

$$r'(B^*) + \beta + \sum_{u \in S_B} \alpha_u - \sum_{\{u,v\} \in F_1 | u,v \in S_B} \gamma_{uv} \geq |E_B|.$$

Portanto, $(\beta + r'(B^*), \alpha, \gamma)$ é uma solução viável de (DPL3_{EB}) com valor $z(\beta + r'(B^*), \alpha, \gamma) = k \cdot r'(B^*) + z(\beta, \alpha, \gamma)$ na função objetivo. Do lema de *dualidade fraca*, temos que $z(\beta + r'(B^*), \alpha, \gamma)$ é um limitante superior no valor de uma solução ótima de (PL3_{EB}). Muitas vezes estamos interessados em resolver (PL3_{EB}) apenas para obter um bom limitante superior para o valor de uma solução ótima inteira. No caso em que o valor de $\lfloor z(\beta + r'(B^*), \alpha, \gamma) \rfloor - \lfloor z(\beta, \alpha, \gamma) \rfloor$ é próximo de zero, podemos encerrar o processo de geração de colunas antes de resolver (PL3_{EB}).

3.3 Experimentos Computacionais

Nesta seção, apresentamos os resultados de experimentos computacionais realizados com grafos gerados aleatoriamente, conforme descrito na Seção 2.7.1, e também com as mesmas instâncias vindas de aplicações que foram utilizadas nos experimentos apresentados na Seção 2.7.2. Utilizamos o CPLEX 12.2 como o resolvedor de Programação Linear Inteira, implementamos o algoritmo de *branch-and-price* descrito nas seções anteriores em linguagem C++ e realizamos os experimentos em um computador com 65 GB de memória RAM e um processador com velocidade de 1.6 GHz.

Apresentamos nas Tabelas 3.3.1 a 3.3.4 os resultados dos testes feitos com instâncias geradas aleatoriamente. Para cada densidade em $\{0.1, 0.2, \ldots, 0.9\}$, geramos 100 instâncias diferentes e calculamos a média aritmética dos resultados. Na primeira coluna mostramos se $S = L$, $S = R$ ou $S = V$, na segunda coluna mostramos o valor do parâmetro k e na terceira coluna mostramos a densidade do grafo G. Estipulamos um limite máximo de 5 minutos para o tempo de processamento. Nas Tabelas 3.3.1 e 3.3.2 mostramos os resultados referentes às instâncias que foram resolvidas dentro do limite máximo de tempo e nas Tabelas 3.3.3 e 3.3.4 mostramos os resultados referentes às instâncias em que o programa foi interrompido após exceder o limite máximo de tempo. Mostramos

na coluna "%Inst." o percentual de instâncias resolvidas, no caso das Tabelas 3.3.1 e 3.3.2, ou o percentual de instâncias para as quais ocorreu a interrupção, no caso das Tabelas 3.3.3 e 3.3.4. Utilizamos as seguintes siglas: "HrG" refere-se à HEURÍSTICA-GULOSA, que corresponde á primeira etapa de todo o processo, na qual são construídos um empacotamento viável e o programa linear restrito ($PL2_{EB}(\mathcal{B}')$) inicial; "GerCol" refere-se ao procedimento de geração de colunas para resolver o programa linear ($PL2_{EB}$), partindo do programa linear restrito ($PL2_{EB}(\mathcal{B}')$) inicial construído na etapa anterior; "HrSIR" refere-se à HEURÍSTICA-DA-SOLUÇÃO-INTEIRA-RESTRITA, partindo de uma solução ótima de ($PL2_{EB}$) calculada nas etapas anteriores; "B&P" refere-se ao procedimento de *branch-and-price* para resolver o programa inteiro ($PI2_{EB}$), partindo de uma solução ótima de ($PL2_{EB}$) e do menor limitante superior obtido, ambos calculados nas etapas anteriores.

Na coluna "#Col." mostramos o número total de colunas geradas durante todo o processo de geração de colunas e *branch-and-price*. Na coluna "#Nós" mostramos o número de nós explorados na árvore de busca. Dados dois números inteiros não negativos $LimSup$ e $LimInf$, que correspondem a um limitante superior e um limitante inferior no valor de um empacotamento ótimo, respectivamente, dizemos que o *gap* entre $LimSup$ e $LimInf$ é o valor dado por $gap = 100 \cdot \frac{(LimSup - LimInf)}{LimSup}$. O valor de *gap* é não-negativo e funciona como uma medida de proximidade entre os limitantes inferior e superior, sendo que quanto mais próximo de zero estiver este valor, mais próximos estão os dois limitantes em questão. Nas duas últimas colunas das Tabelas 3.3.1 e 3.3.2 mostramos os *gaps* entre as soluções encontradas pelas heurísticas e a solução ótima.

Tabela 3.3.1. Grafos bipartidos balanceados gerados aleatoriamente ($|E| = 500, \alpha = 1$).

				Tempo de Processamento							Gap	
S	k	Dns	%Inst.	HrG	GerCol	HrSIR	B&P	Total	#Col.	#Nós	HrG	HrSIR
		$[1,3]$	100.00%	0.016	0.131	0.007	0.003	0.157	325	0	1.27%	0.00%
	4	$[4,6]$	100.00%	0.006	0.112	0.010	0.084	0.212	313	0	5.00%	0.05%
		$[7,9]$	100.00%	0.000	0.394	0.145	1.781	2.320	1324	3	15.31%	0.22%
		$[1,3]$	100.00%	0.021	0.151	0.007	0.001	0.180	306	0	2.19%	0.00%
L	6	$[4,6]$	100.00%	0.007	0.110	0.012	0.100	0.229	349	0	7.51%	0.04%
		$[7,9]$	100.00%	0.000	0.236	0.095	0.485	0.816	1050	2	27.37%	0.12%
		$[1,3]$	100.00%	0.025	0.147	0.007	0.004	0.183	302	0	3.15%	0.00%
	8	$[4,6]$	100.00%	0.008	0.098	0.013	0.039	0.157	360	0	10.49%	0.03%
		$[7,9]$	100.00%	0.000	0.169	0.058	0.126	0.353	901	0	39.99%	0.04%
		$[1,3]$	100.00%	0.014	0.419	0.047	1.248	1.729	498	2	3.50%	0.18%
	4	$[4,6]$	78.33%	0.007	3.022	0.565	42.244	45.837	1019	14	7.65%	0.93%
		$[7,9]$	23.33%	0.000	1067.842	24.595	8.968	1101.405	11525	0	5.83%	0.10%
		$[1,3]$	94.00%	0.018	1.130	0.397	27.600	29.145	1033	14	7.43%	0.36%
V	6	$[4,6]$	19.33%	0.010	6.327	1.985	111.898	120.219	1543	23	11.54%	1.19%
		$[7,9]$	19.33%	0.000	1080.603	20.970	8.871	1110.444	11237	0	5.19%	0.03%
		$[1,3]$	48.67%	0.022	1.774	1.314	93.196	96.306	1630	31	10.55%	0.50%
	8	$[4,6]$	1.00%	0.003	31.447	7.353	157.987	196.790	2485	7	13.05%	0.68%
		$[7,9]$	17.67%	0.000	1000.303	25.536	2.399	1028.238	11634	0	5.34%	0.00%

Tabela 3.3.2. Grafos bipartidos desbalanceados gerados aleatoriamente ($|E| = 500, \alpha = \frac{1}{2}$).

				Tempo de Processamento							Gap	
S	k	Dns	%Inst.	HrG	GerCol	HrSIR	B&P	Total	#Col.	#Nós	HrG	HrSIR
		[1, 3]	100.00%	0.007	0.026	0.000	0.000	0.033	96	0	0.12%	0.00%
	4	[4, 6]	100.00%	0.000	0.019	0.001	0.003	0.023	138	0	1.97%	0.01%
		[7, 9]	100.00%	0.000	0.061	0.020	0.055	0.136	523	1	17.63%	0.09%
		[1, 3]	100.00%	0.013	0.026	0.000	0.000	0.039	92	0	0.12%	0.00%
L	6	[4, 6]	100.00%	0.000	0.019	0.002	0.007	0.028	153	0	3.90%	0.00%
		[7, 9]	100.00%	0.000	0.039	0.013	0.015	0.067	433	0	33.77%	0.02%
		[1, 3]	100.00%	0.013	0.027	0.000	0.000	0.040	89	0	0.29%	0.00%
	8	[4, 6]	100.00%	0.001	0.020	0.001	0.002	0.025	162	0	7.20%	0.00%
		[7, 9]	100.00%	0.000	0.027	0.010	0.005	0.041	379	0	49.73%	0.01%
		[1, 3]	97.67%	0.033	1.413	0.360	14.466	16.272	1751	6	5.96%	0.22%
	4	[4, 6]	100.00%	0.013	0.980	0.121	4.242	5.355	984	4	9.93%	0.17%
		[7, 9]	99.00%	0.007	2.537	1.137	17.238	20.920	3563	6	15.23%	0.39%
		[1, 3]	92.67%	0.037	1.684	0.597	25.874	28.192	1996	8	9.22%	0.17%
R	6	[4, 6]	99.67%	0.014	0.889	0.188	6.313	7.405	1109	6	15.24%	0.17%
		[7, 9]	99.00%	0.008	1.322	0.643	9.016	10.988	2663	6	25.77%	0.24%
		[1, 3]	88.33%	0.040	1.690	0.618	22.191	24.539	1854	8	11.96%	0.15%
	8	[4, 6]	100.00%	0.015	0.747	0.171	3.617	4.550	1063	4	20.39%	0.08%
		[7, 9]	99.67%	0.008	0.926	0.419	3.642	4.994	2175	3	36.53%	0.13%
		[1, 3]	99.00%	0.006	0.144	0.136	6.738	7.024	499	9	5.68%	0.38%
	4	[4, 6]	80.67%	0.000	0.771	0.920	42.509	44.200	1073	16	8.41%	1.18%
		[7, 9]	50.00%	0.000	40.657	23.260	47.111	111.027	9281	2	4.62%	0.60%
		[1, 3]	75.00%	0.009	0.300	0.721	50.195	51.225	1068	24	9.98%	0.58%
V	6	[4, 6]	8.00%	0.000	1.447	3.837	136.750	142.034	1797	27	13.03%	1.60%
		[7, 9]	52.33%	0.000	38.118	24.018	56.687	118.823	8914	3	4.41%	0.80%
		[1, 3]	26.67%	0.013	0.473	1.971	94.489	96.946	1343	27	12.04%	0.71%
	8	[4, 6]	2.67%	0.000	3.406	7.279	133.624	144.309	2282	14	14.23%	1.36%
		[7, 9]	49.67%	0.000	46.497	29.534	54.379	130.410	9783	2	3.97%	0.71%

Conforme mostrado nas Tabelas 3.3.1 e 3.3.2, obtivemos resultados bastante satisfatórios no caso em que $S = L$. Neste caso, foram encontradas soluções ótimas para todas as instâncias, sendo que para a grande maioria das instâncias não houve a necessidade de iniciar a fase de *branch-and-price*. No caso em que $S = R$, o número de nós necessários para encontrar uma solução ótima foi pequeno em todos os casos, sendo que para algumas instâncias não foram encontradas soluções ótimas dentro do tempo limite estabelecido. Neste caso, o tempo de processamento foi muito maior do que no caso em que $S = L$. Isso se deve ao fato de nosso algoritmo para resolver o problema de *pricing* explorar o espaço de busca dos vértices em S e, consequentemente, quanto mais desbalanceado for o grafo de entreda, maior dende a ser a diferença de desempenho entre os casos $S = L$ e $S = R$. O caso em que nosso algoritmo obteve o pior desempenho foi quando $S = V$. Neste caso, uma parte consideravelmente grande das instâncias não foi resolvida dentro do tempo limite estabelecido. Além disso, tanto o número de colunas geradas como o número de nós explorados foi maior do que nos casos anteriores. Observando a estrutura de algumas soluções, pudemos notar que os bicliques contidos em um empacotamento ótimo tendem a ser mais balanceados no caso em que $S = V$ do que nos casos em que $S = L$ ou $S = R$ e, consequentemente, são mais difíceis de serem encontrados.

Considerando todos os dados mostrados nessas duas primeiras tabelas, o número de nós explorados foi pequeno, o que demonstra que a formulação ($PI2_{EB}$) é bastante forte. Além disso, o *gap* máximo obtido pela heurística HrSIR foi de 1.6%, o que consideramos bastante satisfatório. Apesar de os *gaps* obtidos pela heurística HrG terem sido maiores do que os obtidos por HrSIR, o tempo de processamento de HrG foi muito menor do que o de HrSIR. Observe que HrSIR só pode ser executada após o término da etapa de geração de colunas, que costuma consumir muito mais tempo do que a fase inicial, que corresponde á execução de HrG. O valor do parâmetro k influenciou bastante na qualidade das soluções encontradas por HrG. Em praticamente todos os casos, quanto maior o valor de k, maior foi o *gap* obtido por HrG.

Atentamos o leitor para o fato de que as interrupções por excesso de tempo de processamento foram efetuadas apenas após a etapa de geração de colunas. Optamos por assim proceder pois julgamos conveniente que pelo menos essa etapa fosse executada até o final. Em alguns casos, a etapa de geração de colunas durou mais do que o tempo limite estabelecido, fazendo com que a média de tempo ficasse acima dos 5 minutos estabelecidos como tempo máximo. A seguir, nas Tabelas 3.3.3 e 3.3.4 mostramos os resultados das instâncias para as quais nosso programa foi interrompido após exceder o tempo limite estabelecido, dentro das condições mencionadas acima.

Tabela 3.3.3. Tempo limite excedido – Grafos balanceados ($|E| = 500, \alpha = 1$).

				Tempo de Processamento							
S	k	Dns	%Inst.	HrG	GerCol	HrSIR	B&P	Total	#Col.	#Nós	Gap final
		[1, 3]	0%	-	-	-	-	-	-	-	-
	4	[4, 6]	21.67%	0.001	15.829	3.372	287.639	306.840	1647	28	2.35%
		[7, 9]	76.67%	0.000	371.839	18.797	97.059	487.695	2212	1	5.41%
		[1, 3]	5.67%	0.014	1.797	1.034	302.007	304.851	2588	70	1.45%
V	6	[4, 6]	80.67%	0.005	31.168	12.545	271.515	315.233	1412	18	4.34%
		[7, 9]	80.67%	0.000	388.142	19.023	88.132	495.297	2270	1	5.02%
		[1, 3]	51.33%	0.017	5.966	2.913	296.123	305.019	1881	50	1.88%
	8	[4, 6]	99.00%	0.006	37.548	17.842	263.693	319.090	1315	9	5.30%
		[7, 9]	82.33%	0.000	408.890	19.730	86.184	514.804	2420	1	4.89%

Conforme mostramos nas Tabelas 3.3.3 e 3.3.4, em todos os casos o *gap* final obtido foi pequeno. O caso em que $S = V$ foi o que teve mais instâncias para as quais nosso programa foi interrompido, e também nesse caso os *gaps* obtidos foram os maiores.

Na Tabela 3.3.5, mostramos o resultado de experimentos computacionais realizados com instâncias vindas das aplicações de *empacotamento de produtos* (EP) e *redes metabólicas* (RM). As instâncias utilizadas para gerar a Tabela 3.3.5 são as mesmas utilizadas na Seção 2.7.2 e no trabalho de Acuña *et al.* (2011). Nos casos em que $S = R$ ou $S = V$ não foi possível encontrar uma solução ótima nem mesmo para ($PL2_{EB}$), dentro de um tempo limite de 48 horas de processamento. Portanto, consideramos apenas o caso em que $S = L$.

Tabela 3.3.4. Tempo limite excedido – Grafos desbalanceados ($|E| = 500, \alpha = 1$).

				Tempo de Processamento							
S	k	Dns	%Inst.	HrG	GerCol	HrSIR	B&P	Total	#Col.	#Nós	Gap final
		[1, 3]	2.00%	0.038	1.987	1.458	308.498	311.982	4425	35	1.96%
	4	[4, 6]	0.00%	-	-	-	-	-	-	-	-
		[7, 9]	1.00%	0.010	1.867	1.140	301.403	304.420	4947	52	0.38%
		[1, 3]	7.00%	0.053	2.280	1.202	302.350	305.885	4048	46	1.53%
R	6	[4, 6]	0.33%	0.010	0.800	0.670	303.290	304.770	3497	86	0.49%
		[7, 9]	1.00%	0.010	1.290	1.053	303.237	305.590	4622	70	0.38%
		[1, 3]	11.33%	0.063	2.217	1.679	302.163	306.122	3857	46	1.21%
	8	[4, 6]	0.00%	-	-	-	-	-	-	-	-
		[7, 9]	0.33%	0.010	0.640	0.650	301.160	302.460	3826	101	0.36%
		[1, 3]	1.00%	0.010	0.153	0.847	301.900	302.910	2249	85	1.65%
	4	[4, 6]	19.33%	0.000	1.724	3.554	300.608	305.886	1805	39	2.11%
		[7, 9]	50.00%	0.000	7.889	18.205	293.803	319.896	2445	11	3.55%
		[1, 3]	24.67%	0.009	0.323	1.624	302.274	304.229	2081	73	1.55%
V	6	[4, 6]	92.00%	0.000	2.792	16.306	294.669	313.766	1432	21	4.08%
		[7, 9]	47.67%	0.000	8.664	21.897	289.803	320.364	2491	10	3.57%
		[1, 3]	73.33%	0.010	0.674	4.779	299.559	305.023	1549	43	2.12%
	8	[4, 6]	97.33%	0.000	3.481	25.085	291.502	320.068	1379	11	5.04%
		[7, 9]	50.33%	0.000	7.270	22.006	294.384	323.660	2403	10	3.76%

Tabela 3.3.5. Experimentos realizados com instâncias vindas de aplicações ($S = L$).

					Tempo de Processamento					Gaps Heurísticas	
Inst.	α	$\|E\|$	Dens.	k	HrG	GerCol	HrSIR	Total	#Col.	HrG	HrSIR
				4	0.11	0.63	0.03	0.77	746	10.24%	0.00%
EP1	$\frac{1}{24}$	9449	0.4	6	0.13	0.54	0.04	0.71	768	8.29%	0.00%
				8	0.14	0.51	0.04	0.69	750	8.72%	0.00%
				4	1.69	11.44	0.11	13.24	1494	8.80%	0.00%
EP2	$\frac{1}{10}$	15519	0.3	6	2.06	7.51	0.11	9.68	1546	7.85%	0.00%
				8	2.52	12.93	0.12	15.57	1548	6.93%	0.00%
				4	0.02	0.03	0.02	0.07	94	4.13%	0.00%
RM1	$\frac{1}{290}$	41010	0.6	6	0.01	0.04	0.02	0.07	96	6.11%	0.00%
				8	0.02	0.04	0.03	0.09	92	5.85%	0.00%
				4	0.12	0.24	0.04	0.40	242	0.47%	0.00%
RM2	$\frac{1}{116}$	71140	0.4	6	0.15	0.23	0.05	0.43	264	3.50%	0.00%
				8	0.18	0.25	0.05	0.48	254	4.95%	0.00%

Conforme mostramos na Tabela 3.3.5, o número de colunas geradas para cada uma das instâncias foi pequeno, comparado com o tamanho da entrada e, além disso, as soluções ótimas foram encontradas sem a necessidade de iniciar a etapa de *branch-and-price*, pois para todas essas instâncias a heurística HrSIR obteve uma solução ótima. Em muitos casos, o custo-benefício de executar

apenas o procedimento de geração de colunas seguido de alguma heurística análoga a HrSIR pode ser bastante vantajoso. Isto porque a implementação da etapa de *branch-and-price* é bastante trabalhosa e sua execução pode ter um custo computacional adicional muito grande para, muitas vezes, encontrar uma solução com valor ligeiramente maior do que a encontrada pela heurística.

Capítulo 4

Um Problema de Programação Bilinear Inteira

Neste capítulo, apresentamos um método para resolver um problema de Programação Bilinear Inteira e, através de experimentos computacionais, comparamos nossa abordagem com uma técnica de linearização existente na literatura. O conteúdo deste capítulo está baseado no artigo de Freire *et al.* (2011).

Na Seção 2.3, apresentamos o método MRFO para resolver o problema de BICLIQUE MÁXIMO. No entanto, o MRFO foi desenvolvido originalmente para resolver o seguinte problema:

$$
\begin{aligned}
& \text{maximize} && \left(\sum_i \alpha_i x_i\right)\left(\sum_j \beta_j y_j\right) && && \\
(\mathrm{P_B}) \qquad & \text{sujeito a} && \sum_{i=1}^{n} a_{ki}x_i + \sum_{j=1}^{m} b_{kj}y_j \;\leq\; d_k, && \text{para } k = 1,\ldots,K && (13.1)\\
& && x \;\in\; \mathbb{N}^n && && (13.2)\\
& && y \;\in\; \mathbb{N}^m && && (13.3)
\end{aligned}
$$

onde os coeficientes α_i e β_j são inteiros não-negativos (podemos considerar coeficientes racionais, desde que multiplicados por um escalar apropriado).

Na Seção 4.1, apresentamos uma técnica de linearização existente na literatura. Na Seção 4.2, apresentamos nossa abordagem para resolver ($\mathrm{P_B}$), que corresponde à versão original do método MRFO. Na Seção 4.3, descrevemos dois casos especiais de ($\mathrm{P_B}$) e mostramos sua relação com o problema de BICLIQUE MÁXIMO. Por fim, na Seção 4.4, mostramos os resultados de experimentos computacionais realizados com as abordagens apresentadas neste capítulo.

4.1 Técnica de Decomposição Binária

Se todas as variáveis de ($\mathrm{P_B}$) forem limitadas, podemos transformar ($\mathrm{P_B}$) em um Programa Linear Inteiro, utilizando alguma das técnicas conhecidas de linearização apresentadas nos trabalhos de Al-Khayyal e Falk (1983); Harjunkoski *et al.* (1997); Ibaraki (1976); McCormick (1983); Padberg (1989); Sherali e Adams (1999). Uma dessas técnicas consiste em substituir cada variável x_i por sua decomposição binária $\sum_{r=0}^{\lceil \lg s_i \rceil} 2^r w_{ir}$, onde w_{ir} é uma variável binária e s_i é um limitante superior para o valor de x_i. Além disso, adiciona-se variáveis inteiras z_{ijr}, com a interpretação de que $z_{ijr} =$

$w_{ir} \cdot y_j$. Como esta restrição é não-linear, não podemos incluí-la explicitamente na formulação, sendo necessário então introduzir novas restrições. A seguir, apresentamos a linearização da formulação ($\mathrm{P_B}$), utilizando a técnica de decomposição binária.

$$
\begin{array}{lll}
\text{maximize} & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{r=0}^{\lceil \lg s_i \rceil} \alpha_i\beta_j 2^r \cdot z_{ijr} & \\
\text{sujeito a} & z_{ijr} \leq y_j, & \forall 1 \leq j \leq m, 1 \leq i \leq n \text{ e } 1 \leq r \leq \lceil \lg s_i \rceil \quad (14.1)\\
& z_{ijr} \leq s_j \cdot w_{ir}, & \forall 1 \leq j \leq M,\ 1 \leq i \leq N \text{ e } 1 \leq r \leq \lceil \lg s_i \rceil \quad (14.2)\\
& y_j - s_j \cdot (1 - w_{ir}) \leq z_{ijr}, & \forall 1 \leq j \leq m, 1 \leq i \leq n \text{ e } 1 \leq r \leq \lceil \lg s_i \rceil \quad (14.3)\\
& \displaystyle\sum_{i=1}^{n}\sum_{r=0}^{\lceil \lg s_i \rceil} 2^r a_{ki} w_{ir} + \sum_{j=1}^{m} b_{kj} y_j \leq d_k, & \forall 1 \leq k \leq K \quad (14.4)\\
& z_{ijr} \geq 0, & \forall 1 \leq j \leq m,\ 1 \leq i \leq n \text{ e } 1 \leq r \leq \lceil \lg s_i \rceil \quad (14.5)\\
& w_{ir} \in \{0,1\}, & \forall 1 \leq i \leq n \text{ e } 1 \leq r \leq \lceil \lg s_i \rceil \quad (14.6)\\
& y \in \mathbb{N}^m & (14.7)
\end{array}
\qquad (\mathrm{P_{LIN}})
$$

As restrições (14.1), (14.2) e (14.3) garantem que $z_{ijr} = w_{ir} \cdot y_j$, como queríamos. Na formulação ($\mathrm{P_{LIN}}$) são definidas $O(nms^*)$ variáveis, onde $s^* = \max_{i=1,\ldots,n} \lg(s_i)$. Aplicando a decomposição binária nas variáveis y, em vez de x, obteríamos uma outra linearização da formulação ($\mathrm{P_B}$), porém com uma quantidade de variáveis assintoticamente superior à quantidade de variáveis de ($\mathrm{P_{LIN}}$).

4.2 Método de Relaxação da Função Objetivo

Na Seção 2.3, mostramos no Lema 2.3.1 que se $a_1 + a_2 \geq b_1 + b_2$ e $a_1 \cdot a_2 < b_1 \cdot b_2$, então $\min\{a_1, a_2\} < \min\{b_1, b_2\}$, onde a_1, a_2, b_1, b_2 são números positivos. Utilizando o Lema 2.3.1, podemos resolver ($\mathrm{P_B}$) da seguinte maneira. Em vez de maximizar a função não-linear $(\sum_{i=1}^{n} \alpha_i x_i)(\sum_{j=1}^{m} \beta_j y_j)$, maximizamos a função linear $(\sum_{i=1}^{n} \alpha_i x_i) + (\sum_{j=1}^{m} \beta_j y_j)$, impondo um limitante inferior LI para $\sum_{i=1}^{n} \alpha_i x_i$ e $\sum_{j=1}^{m} \beta_j y_j$. Observe que, para algum valor de LI, temos que uma solução ótima do problema original é também uma solução ótima do problema relaxado. Dado um inteiro positivo LI, considere a seguinte formulação.

$$
\begin{array}{lll}
\text{maximize} & \displaystyle\sum_{i=1}^{n} \alpha_i x_i + \sum_{j=1}^{m} \beta_j y_j & \\
\text{sujeito a} & \displaystyle\sum_{i=1}^{n} \alpha_i x_i \geq \text{LI}, & (15.1)\\
& \displaystyle\sum_{j=1}^{m} \beta_j y_j \geq \text{LI}, & (15.2)\\
& \displaystyle\sum_{i=1}^{n} a_{k,i} x_i + \sum_{j=1}^{m} b_{k,j} y_j \leq d_k, \quad \text{para } k = 1, \ldots, K & (15.3)\\
& x \in \mathbb{N}^n & (15.4)\\
& y \in \mathbb{N}^m & (15.5)
\end{array}
\qquad (\mathrm{P_{MRFO}})
$$

Para obter uma solução ótima de ($\mathrm{P_B}$), executamos o procedimento descrito a seguir: (1) resolve-

mos (P_{MRFO}) com $\text{LI} = 1$; (2) atualizamos o valor de LI com $\text{LI} = \min\{\sum_{i=1}^{n} \alpha_i x_i^*, \sum_{j=1}^{m} \beta_j y_j^*\} + 1$, onde (x^*, y^*) é uma solução ótima de (P_{MRFO}), considerando o valor anterior de LI; (3) resolvemos então (P_{MRFO}) com o novo valor de LI; (4) se (P_{MRFO}) for viável, voltamos para o passo (2), caso contrário, encerramos o procedimento. Denotamos este procedimento por MRFO. Utilizando o Lema 2.3.1, é possivel mostrar que uma solução encontrada por MRFO que maximize $(\sum_{i=1}^{n} \alpha_i x_i)(\sum_{j=1}^{m} \beta_j y_j)$ é uma solução ótima de (P_{B}). É importante observar que estamos assumindo que o valor de uma solução ótima de (P_{B}) é estritamente maior do que zero. Neste caso, (P_{B}) é inviável ou ilimitado se e somente se (P_{MRFO}) é inviável ou ilimitado, respectivamente. Portanto, podemos seguramente inicializar o procedimento com $\text{LI} = 1$. No caso em que o valor de uma solução ótima de (P_{B}) é igual a zero, temos que na primeira iteração (P_{MRFO}) é inviável. Portanto, para considerar este caso, basta checar se existe uma solução em que $x_i = 0$ para todo i tal que $\alpha_i > 0$, ou $y_j = 0$ para todo j tal que $\beta_j > 0$.

Observe que a cada iteração de MRFO encontramos uma nova solução cujo valor na função objetivo não necessariamente é maior do que o das soluções anteriormente encontradas. Seria desejável se pudéssemos evitar as iterações "infrutíferas", ou pelo menos garantir que a cada iteração o valor de LI aumentará em no mínimo T unidades, para um certo parâmetro T. A seguir, apresentamos uma formulação que pode substituir (P_{MRFO}) em MRFO, onde ϕ ó valor da melhor solução encontrada até o momento e T é um inteiro positivo.

$$
\begin{aligned}
& & \text{maximize} \quad & \sum_{i=1}^{n} \alpha_i x_i + \sum_{j=1}^{m} \beta_j y_j & & & \\
(\mathrm{P}_{\text{MRFO+}}) \qquad & & \text{sujeito a} \quad & \sum_{i=1}^{n} \alpha_i x_i \;\geq\; \sum_{t=0}^{T} (\text{LI} + t) \cdot p_t & & & (16.1)\\
& & & \sum_{j=1}^{m} \beta_j y_j \;\geq\; \sum_{t=0}^{T} (\text{LI} + t) \cdot p_t & & & (16.2)\\
& & & \sum_{i=1}^{n} \alpha_i x_i + \sum_{j=1}^{m} \beta_j y_j \;\geq\; \sum_{t=0}^{T-1} \left((\text{LI} + t) + \left\lceil \frac{\phi + 1}{\text{LI} + t} \right\rceil \right) \cdot p_t & & & (16.3)\\
& & & \sum_{t=0}^{T} p_t \;=\; 1 & & & (16.4)\\
& & & \sum_{i=1}^{n} a_{ki} x_i + \sum_{j=1}^{m} b_{kj} y_j \;\leq\; d_k, & & \text{para } k = 1, \ldots, K & (16.5)\\
& & & x \;\in\; \mathbb{N}^n & & & (16.6)\\
& & & y \;\in\; \mathbb{N}^m & & & (16.7)\\
& & & p \;\in\; \{0,1\}^{T+1} & & & (16.8)
\end{aligned}
$$

O objetivo de introduzir as variáveis p é encontrar uma solução melhor do que as anteriores a cada iteração ou então garantir que LI aumenta em T unidades. Seja (x^*, y^*, p^*) uma solução ótima de ($\mathrm{P}_{\text{MRFO+}}$) e seja t^* o índice tal que $p_{t^*}^* = 1$. As restrições (16.1), (16.2) e (16.4) garantem que cada termo $\sum_{i=1}^{n} \alpha_i x_i^*$ e $\sum_{j=1}^{m} \beta_j y_j^*$ são maiores ou iguais a $\text{LI} + t^*$, onde $1 \leq t^* \leq T$. A restrição (16.3) garante que, caso $t^* < T$, a solução encontrada terá maior valor na função objetivo de (P_{B}) do que as soluções anteriores. Esta condição é satisfeita devido ao fato de os termos $\sum_{i=1}^{n} \alpha_i x_i^*$ e $\sum_{j=1}^{m} \beta_j y_j^*$ serem maiores ou iguais a $\text{LI} + t^*$ e a soma desses dois termos ser maior ou igual a $(\text{LI} + t^*) + \lceil \frac{\phi+1}{\text{LI}+t^*} \rceil$.

Logo, utilizando o Lema 2.3.1, temos que $(\sum_{i=1}^{n} \alpha_i x_i^*)(\sum_{j=1}^{m} \beta_j y_j^*) \geq (\text{LI}+t^*)\cdot\lceil\frac{\phi+1}{\text{LI}+t^*}\rceil > \phi$. Caso não exista solução viável de (P_B) com valor maior que ϕ na qual os termos $\sum_{i=1}^{n} \alpha_i x_i^*$ e $\sum_{j=1}^{m} \beta_j y_j^*$ sejam menores ou iguais a $\text{LI} + T$, então temos que $t^* = T$ e, consequentemente, $\sum_{i=1}^{n} \alpha_i x_i^*$ e $\sum_{j=1}^{m} \beta_j y_j^*$ são limitados inferiormente por $\text{LI} + T$. Para T suficientemente grande, a cada iteração de MRFO é encontrada uma solução com maior valor na função objetivo. Dependendo dos coeficientes α_i e β_j, o valor de T pode ser exponencial no tamanho da entrada para garantir o incremento na função objetivo em todas as iterações.

Denotamos por MRFO+ o procedimento análogo a MRFO no qual utilizamos $(\text{P}_{\text{MRFO+}})$ no lugar de (P_MRFO). Em MRFO+, para obter uma solução ótima de (P_B), iniciamos o procedimento com $\phi = 0$ e $\text{LI} = 1$. A cada iteração, atualizamos os valores de ϕ e LI de acordo com os seguintes casos: (i) caso tenha sido encontrada uma solução com valor maior do que ϕ, então atualizamos ϕ e LI com $\phi = (\sum_{i=1}^{n} \alpha_i x_i^*)(\sum_{j=1}^{n} \beta_j y_j^*)$ e $\text{LI} = \min\{\sum_{i=1}^{n} \alpha_i x_i^*, \sum_{j=1}^{m} \beta_j y_j^*\} + 1$; (ii) caso contrário, atualizamos apenas LI com $\text{LI} = \text{LI} + T$.

4.3 Aplicações

Para comparar os desempenhos de MRFO e (P_LIN), utilizamos instâncias dos problemas descritos a seguir:

- BiLinGB: dado um grafo bipartido completo $G = (L \cup R, E)$, para cada vértice v associamos um preço $p_v \in \mathbb{N}$ e um custo $c_v \in \mathbb{N}$, e para cada aresta uv associamos uma capacidade $d_{uv} \in \mathbb{N}$. Deseja-se encontrar vetores $x \in \mathbb{N}^{|L|}$ e $y \in \mathbb{N}^{|R|}$, tais que $c_u x_u + c_v y_v \leq d_{uv}$, para cada aresta uv de E, e a função $(\sum_{u\in L} p_u x_u)(\sum_{v\in R} p_v y_v)$ é maximizada. Denotamos este problema por BiLinGB.

- EmpProd: na Seção 1.3 descrevemos o problema de EMPACOTAMENTO DE PRODUTOS, no qual são dados um grafo bipartido $G = (L \cup R, E)$ e uma função $d : L \times R \to \mathbb{N}$, tal que $d_{uv} = 0$, para todo $uv \notin E$. Consideramos aqui o caso simplificado em que deseja-se encontrar vetores $x \in \mathbb{N}^{|L|}$ e $y \in \mathbb{N}^{|R|}$, tais que $x_u \cdot y_v \leq d_{uv}$, para todo $u \in L$ e $v \in R$, e a função $(\sum_{u\in L} x_u)(\sum_{v\in R} y_v)$ é maximizada. No caso em que d é uma função binária, tal que $d_{uv} = 1$ se e somente se $uv \in E$, temos o problema de BICLIQUE MÁXIMO.

Observe que no problema EmpProd temos uma restrição não-linear a ser satisfeita. Formalmente, o problema EmpProd consiste em resolver o programa não-linear inteiro (P_EP) descrito abaixo.

$$
\begin{aligned}
& & \text{maximize} \quad & \sum_{u\in L} x_u \cdot \sum_{v\in R} y_v & & & \\
(\text{P}_\text{EP}) \qquad & & \text{sujeito a} \quad & x_u \cdot y_v \leq d_{uv}, & \text{para todo } uv \in E & & (17.1)\\
& & & x \in \mathbb{N}^{|L|} & & & (17.2)\\
& & & y \in \mathbb{N}^{|R|} & & & (17.3)
\end{aligned}
$$

Neste caso, para aplicar o método MRFO, utilizamos a técnica de linearização apresentada na Seção 4.1, que consiste em substituir as restrições (17.1) por

$$y_v \leq \left\lfloor \frac{d_{uv}}{\kappa} \right\rfloor + \left(s_v - \left\lfloor \frac{d_{uv}}{\kappa} \right\rfloor\right) \left(|b(\kappa)| - \sum_{j:b(\kappa)_j=1} w_{vj}\right), \qquad \forall u \in L, \forall v \in R,\ \kappa = 1, \ldots, s_v \qquad (17.4)$$

$$x_v = \sum_{i=0}^{\lceil \lg s_v \rceil} 2^i w_{vi}, \qquad \forall v \in R \qquad (17.5)$$

$$w_{vi} \in \{0, 1\}, \qquad \forall v \in R, i = 1, \ldots, \lceil \lg s_v \rceil \qquad (17.6)$$

onde $b(\kappa)$ é o vetor que contém a representação binária do inteiro κ (ou seja, $\kappa = \sum_{j=0}^{\lceil \lg \kappa \rceil} 2^{b(\kappa)_j}$) e $|b(\kappa)|$ corresponde à quantidade de bits não-nulos em $b(\kappa)$. As restrições (17.4), (17.5) e (17.6) garantem que $x_v = \kappa$ se e somente se $(|b(\kappa)| - \sum_{j:b(\kappa)_j=1} w_{p,j}) = 0$. Logo, temos que se $x_u = \kappa$ então $y_v \leq \lfloor \frac{d_{uv}}{\kappa} \rfloor$ e, consequentemente, $x_u \cdot y_v \leq d_{uv}$.

No caso da formulação (P_{LIN}), na qual os termos $x_u \cdot y_v$ já estão linearizados, substituimos as restrições (17.1) por $\sum_{r=0}^{\lceil \lg s_v \rceil} 2^r z_{uvr} \leq d_{uv}$, para todo $uv \in E$.

4.4 Experimentos Computacionais

Implementamos as abordagens (P_{LIN}), MRFO e MRFO+, utilizando a linguagem de programação C++ e o CPLEX 12.2 como o resolvedor de Programação Linear Inteira. O computador utilizado em nossos testes possui um processador Intel Xeon e 8Gb de memória RAM. O valor utilizado para o parâmetro T na formulação (P_{MRFO+}) foi $T = 50$. Intuitivamente, quanto maior o valor de T, obtém-se um menor número de iterações, porém o tempo total de processamento tende a ser maior. Escolhemos $T = 50$ porque foi este o valor com o qual obtivemos melhores resultados para o conjunto de dados utilizado em nossos experimentos computacionais.

Na Tabela 4.4.1, mostramos os resultados de experimentos computacionais realizados com instâncias do problema BiLinGB geradas aleatoriamente, com $|L| = |R| = 50$. Para cada vértice v de $L \cup R$, selecionamos aleatoriamente custos c_v e preços p_v seguindo uma distribuição poisson de média 4. A capacidade d_{uv} de cada aresta $uv \in E$ foi selecionada aleatoriamente seguindo uma distribuição de probabilidade poisson de média λ, onde λ é igual a 8, 16 ou 32. Observe que quanto maior o valor de λ, maior o espaço de soluções, supostamente tornando as instâncias mais difíceis de resolver. Para cada valor fixado para λ, geramos 10 instâncias e calculamos a média aritmética dos resultados.

Tabela 4.4.1. Instâncias do problema BiLinGB geradas aleatoriamente, com $|L| = |R| = 50$

	(P_{LIN})	MRFO		MRFO+	
λ	Tempo	Tempo	Num. Iter.	Tempo	Num. Iter.
8	6s	1s	5	3s	3
16	1h16m24s	15s	32	11s	20
32	LME	27m17s	91	21m09s	64

Conforme mostrado na Tabela 4.4.1, a formulação (P_{LIN}) consumiu muito mais tempo de processamento do que as demais abordagens em todos os casos. Além disso, nas instâncias com $\lambda = 32$, a formulação (P_{LIN}) não conseguiu obter uma solução ótima devido ao problema de *limite de memória excedido* (LME), lembrando que o valor de λ influencia na quantidade de variáveis e restrições de (P_{LIN}). Confirmando nossas expectativas, MRFO necessitou de um número maior de iterações do que

MRFO+. Porém, o tempo de processamento médio por iteração foi maior em MRFO+ do que em MRFO. Na maioria dos casos, MRFO+ foi mais eficiente do que MRFO.

Na Tabela 4.4.2, mostramos os resultados de experimentos computacionais realizados com instâncias do problema EMPPROD geradas aleatoriamente, com $|L| = 10$ e $|R| = 30$. Para gerar tais instâncias, primeiro fixamos $d_{uv} = 0$ com probabilidade uniforme ρ. Em seguida, para cada aresta uv, tal que $d_{uv} > 0$, escolhemos d_{uv} seguindo uma distribuição de probabilidade poisson de média λ.

Tabela 4.4.2. Instâncias do problema EMPPROD geradas aleatoriamente, com $|L| = 10$ e $|R| = 30$

		(P_{LIN})	MRFO		MRFO+	
ρ	λ	Tempo	Tempo	Num. Iter.	Tempo	Num. Iter.
0.2	10	4m47s	1m11s	18	20s	4
0.2	50	1h42m40s	35m56s	52	49m42s	9
0.2	200	LME	98h01m29s	108	31h43m39s	12
0.5	10	19s	17s	11	5s	3
0.5	50	16m37s	4m37s	28	1m10s	4
0.5	200	1h31m51s	7h34m58s	77	3h01m14s	7
0.8	10	1s	3s	9	1s	2
0.8	50	8s	40s	19	6s	2
0.8	200	25s	4m22s	35	55s	4

Conforme mostrado na Tabela 4.4.2, as instâncias em que G é mais esparso são mais fáceis de resolver (observe que a densidade esperada de G é $1-\rho$). De modo geral, os resultados observados nas Tabelas 4.4.1 e 4.4.2 são similares, no sentido de que na maioria dos casos, considerando o consumo total de tempo, as abordagens ficaram classificadas em MRFO+, seguida de MRFO e por último (P_{LIN}). No entanto, nos casos em que $\lambda = 200$, observa-se que (P_{LIN}) obteve os melhores resultados (exceto quando $\rho = 0.2$). Conforme observamos em nossos experimentos, tal comportamento deve-se ao fato de que nesses casos a relaxação linear de (P_{LIN}) obteve um limitante melhor do que nos outros casos.

Na Tabela 4.4.3, mostramos os resultados de experimentos computacionais realizados com dados reais obtidos através de uma grande distribuidora de alimentos localizada no Chile. Em cada instância, selecionamos subconjuntos de produtos e clientes (originalmente, o grafo G tem tamanho $|L| = 497$, $|R| = 760$ e $\max d_{uv} = 1.000.000$), e comparamos os resultados obtidos por cada uma das abordagens apresentadas neste capítulo.

Tabela 4.4.3. Instâncias vindas da aplicação de empacotamento de produtos

			(P_{LIN})	MRFO		MRFO+	
Instância	$\|L\| \times \|R\|$	$\max d_{uv}$	Tempo	Tempo	Num. Iter.	Tempo	Num. Iter.
w5x41m	5 × 41	99	27s	44s	31	8s	5
w5x41	5 × 41	433	1m32s	2m07s	47	33s	12
w10x60	10× 60	2402	39m53s	55h31m47s	129	8h06m53s	9
w10x60d	10× 60	201	5m57s	7m02s	35	1m47s	3
w32x311	32× 311	26	LME	19m35s	14	3m13s	6

Os resultados observados na Tabela 4.4.3 são similares aos resultados observados nas tabelas anteriores. No entanto, para a instância w10x60, a formulação (P_{LIN}) obteve um desempenho muito

melhor do que as demais abordagens. Uma possível explicação para o resultado observado seria o fato de que a linearização feita conforme mostramos na Seção 4.3 afetou de forma negativa o desempenho de MRFO e MRFO+. Tanto MRFO como MRFO+ encontraram uma solução ótima para a instância w10x60 na primeira iteração e a maior parte do tempo total de processamento foi consumida para mostrar que a solução encontrada era de fato ótima. As instâncias w10x60 e w10x60d consistem no mesmo grafo de entrada, com as demandas em w10x60d iguais às demandas em w10x60 divididas por 12. Considerando a instância w10x60d, o desempenho das três abordagens voltou a se assemelhar ao observado na maioria dos casos, o que demonstra o papel fundamental de $\max d_{uv}$.

Capítulo 5

Conclusão e Trabalhos Futuros

Nesta tese, estudamos principalmente os problemas de EMPACOTAMENTO DE BICLIQUES e BICLIQUE MÁXIMO. Introduzimos três abordagens diferentes para o problema de BICLIQUE MÁXIMO, sendo que duas delas utilizam formulações de Programação Linear Inteira e uma delas utiliza a técnica de *branch-and-bound*. Através de experimentos computacionais, comparamos nossas abordagens com um algoritmo existente na literatura para resolver o problema de CLIQUE MÁXIMO, e concluímos que nosso algoritmo de *branch-and-bound* foi a abordagem que obteve melhor desempenho, principalmente para grafos desbalanceados. Argumentamos sobre os problemas de formulações com simetria e, para evitar este problema, introduzimos um algoritmo de *branch-and-price* para resolver o problema de EMPACOTAMENTO DE BICLIQUES, no qual o problema de *pricing* consiste em um caso mais geral do problema de BICLIQUE MÁXIMO, permitindo que com poucas adaptações pudéssemos utilizar a mesma abordagem para ambos os problemas. Consideramos que obtivemos contribuições satisfatórias para esses dois problemas, sendo que muitas dessas contribuições foram publicadas nos artigos de Acuña *et al.* (2010) e Acuña *et al.* (2011). Por fim, introduzimos um método para resolver um problema de Programação Bilinear Inteira, do qual o problema de BICLIQUE MÁXIMO é um caso especial. Mostramos que em muitos casos nossa abordagem supera a técnica de representação binária. Nosso método foi publicado no artigo de Freire *et al.* (2011).

Com relação a trabalhos futuros, temos as seguintes sugestões:

- os problemas de BICLIQUE MÁXIMO e EMPACOTAMENTO DE BICLIQUES podem ser formulados para grafos arbitrários (sem a suposição de que G é bipartido). Apesar de não termos encontrado na literatura aplicações desses problemas, a investigação deles nos parece interessante.

- nas abordagens apresentadas para o problema de BICLIQUE MÁXIMO, comentamos quais modificações são necessárias para tratar também o problema de BICLIQUE DE PESO MÁXIMO. No entanto, acreditamos que este problema deve ser investigado em mais detalhes, tanto do ponto de vista teórico como experimental.

- encontramos na literatura um problema relacionado ao de EMPACOTAMENTO DE BICLIQUES que consiste em encontrar uma cobertura de cardinalidade mínima das arestas de E por bicliques (a cardinalidade de uma tal cobertura é conhecida como a *bipartite dimension* do grafo G). É possível utilizar a técnica de *branch-and-price* para resolver este problema, de forma que o problema de *pricing* resultante consiste em encontrar um biclique de peso máximo. É difícil saber, porém, se os resultados obtidos com tal abordagem seriam similares aos obtidos

por nosso algoritmo de *branch-and-price* para o problema de EMPACOTAMENTO DE BICLIQUES.

Além dos temas abordados nesta tese, estudamos durante o doutorado os problemas de MAPEAMENTO DE GRAFOS (a maioria dos nossos resultados para esse problema foram obtidos durante o mestrado) e COBERTURA DE MATRIZES POR RAIOS EXTREMAIS, descritos nas Seções 1.5.1 e 1.5.3, respectivamente. No caso do problema de COBERTURA DE MATRIZES POR RAIOS EXTREMAIS, iniciamos nosso trabalho com esse tema aproximadamente na metade do ano de 2011 e ainda estamos trabalhando juntamente com nossos colaboradores para tentar obter novos resultados e, consequentemente, novas publicações. Após o término do doutorado, pretendemos iniciar um pós-doutoramento na UNICAMP, supervisionado pelo Dr. Cid Carvalho de Souza, no qual daremos continuidade à pesquisa sobre esses temas e, possivelmente, também outros temas. Pretendemos manter um forte contato com os pesquisadores que colaboraram em nossas pesquisas durante o doutorado e também expandir nossa rede de contatos, através de participações em conferências e workshops dentro e fora do Brasil (em outubro de 2012, a convite do Eduardo Moreno, participaremos do *IMSA Latin-American Workshop* que acontecerá no Chile).

Referências Bibliográficas

Acuña *et al.***(2010)** V. Acuña, C. E. Ferreira, A. S. Freire e E. Moreno. The biclique k-clustering problem in bipartite graphs and its application in bioinformatics. *Electronic Notes in Discrete Mathematics*, 36:159 – 166. ISSN 1571-0653. ISCO 2010 - International Symposium on Combinatorial Optimization. Citado na pág. 10, 11, 12, 41, 73

Acuña *et al.***(2011)** V. Acuña, C.E. Ferreira, A.S. Freire e E. Moreno. Solving the maximum edge biclique packing problem on unbalanced bipartite graphs. *Discrete Applied Mathematics*. ISSN 0166-218X. doi: 10.1016/j.dam.2011.09.019. URL http://www.sciencedirect.com/science/article/pii/S0166218X11003556. Citado na pág. 10, 11, 12, 17, 37, 41, 61, 73

Al-Khayyal e Falk(1983) F.A. Al-Khayyal e J.E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8:273–286. Citado na pág. 65

Alexe *et al.***(2004)** Gabriela Alexe, Sorin Alexe, Yves Crama, Stephan Foldes, Peter L. Hammer e Bruno Simeone. Consensus algorithms for the generation of all maximal bicliques. *Discrete Applied Mathematics*, 145(1):11 – 21. ISSN 0166-218X. doi: 10.1016/j.dam.2003.09.004. URL http://www.sciencedirect.com/science/article/pii/S0166218X04000629. Citado na pág. 8, 44

Ambühl *et al.***(2011)** Christoph Ambühl, Monaldo Mastrolilli e Ola Svensson. Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut. *SIAM J. Comput.*, 40(2):567–596. Citado na pág. 9

Barnhart *et al.***(1998)** C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh e P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operational Research*, 46(3):316–329. Citado na pág. 13, 43, 46, 48

Berman e Schnitger(1992) P. Berman e G. Schnitger. On the complexity of approximating the independent set problem. *Information and Computation*, 96(1):77 – 94. Citado na pág. 8, 16

Campêlo *et al.***(2008)** Manoel Campêlo, Victor A. Campos e Ricardo C. Corrêa. On the asymmetric representatives formulation for the vertex coloring problem. *Discrete Applied Mathematics*, 156(7):1097 – 1111. GRACO 2005 - 2nd Brazilian Symposium on Graphs, Algorithms and Combinatorics. Citado na pág. 43

Carvalho *et al.***(2001)** M. H. Carvalho, M. R. Cerioli, R. Dahab, P. Feofiloff, C. G. Fernandes, C. E. Ferreira, K. S. Guimar aes, F. K. Miyazawa, J. C. Pina Jr., J. A. R. Soares e Y. Wakabayashi. *Uma Introdução Sucinta a Algoritmos de Aproximação*. Texto preparado para um curso no vigésimo terceiro Colóquio de Matemática - IMPA, Rio de Janeio - RJ. URL http://www.ime.usp.br/~cris/aprox/. Citado na pág. 1

Chvátal(1983) Vasek Chvátal. *Linear Programming*. W.H. Freeman. Citado na pág. 1

Dawande *et al.***(1997)** M. Dawande, P. Keskinocak e S. Tayur. On the biclique problem in bipartite graphs. Relatório técnico, Carnegie-Mellon University. Citado na pág. 7, 8, 9, 15

Ding *et al.***(2006)** Chris Ding, Ya Zhang, Tao Li e Stephen R. Holbrook. Biclustering protein complex interactions with a biclique finding algorithm. Em *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, páginas 178–187, Washington, DC, USA. IEEE Computer Society. ISBN 0-7695-2701-9. doi: 10.1109/ICDM.2006.27. URL http://dx.doi.org/10.1109/ICDM.2006.27. Citado na pág. 7, 8

Feige e Kogan(2004) U. Feige e S. Kogan. Hardness of approximation of the balanced complete bipartite subgraph problem. Relatório técnico, Department of Computer Science and Applied Mathematics - Weizmann Institute, Israel. Citado na pág. 9, 10

Ferreira *et al.***(2011a)** C.E. Ferreira, A.S. Freire e G.A. Puglia. A dynamic programming algorithm for finding a minimum cost tree mapping. Submetido para revista. Citado na pág. 10, 11

Ferreira *et al.***(2011b)** C.E. Ferreira, A.S. Freire e G.A. Puglia. A dynamic programming algorithm for the tree mapping problem. *Electronic Notes in Discrete Mathematics*, 37(0):147 – 152. LAGOS'11 - VI Latin-American Algorithms, Graphs and Optimization Symposium. Citado na pág. 10, 11

Freire *et al.***(2010)** A.S. Freire, R.M. Cesar e C.E. Ferreira. A column generation approach for the graph matching problem. Em *Proceedings of the 20th International Conference on Pattern Recognition (ICPR)*, páginas 1088–1091. Citado na pág. 10, 11

Freire *et al.***(2011)** A.S. Freire, E. Moreno e J.P. Vielma. An integer linear programming approach for bilinear integer programming. *Operational Research Letters*. Citado na pág. 10, 12, 19, 65, 73

Freire *et al.***(2012)** A.S. Freire, V. Acuña, P. Crescenzi, C.E. Ferreira, V. Lacroix, P.V. Milreu, E. Moreno e M.-F. Sagot. Minimum ratio cover of matrix columns by extreme rays of its induced cone. Em *2nd International Symposium on Combinatorial Optimization (ISCO), Lecture Notes in Computer Science (LNCS)*. Citado na pág. 10, 13

Ganter e Reuter(1991) Bernhard Ganter e Klaus Reuter. Finding all closed sets: A general approach. *Order*, 8:283–290. ISSN 0167-8094. URL http://dx.doi.org/10.1007/BF00383449. 10.1007/BF00383449. Citado na pág. 44

Garey e Johnson(1979) M. R. Garey e D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York. Citado na pág. 7, 8

Haemers(2001) Willem H. Haemers. Bicliques and eigenvalues. *Journal of Combinatorial Theory, Series B*, 82(1):56 – 66. Citado na pág. 7

Harjunkoski *et al.***(1997)** I. Harjunkoski, R. Pörn, T. Westerlund e H. Skrifvars. Different Strategies for Solving Bilinear Integer Non-Linear Programming Problems with Convex Transformations. *Computers and Chemical Engineering*, 21:S487–S492. Citado na pág. 65

Hochbaum(1998) Dorit S. Hochbaum. Approximating clique and biclique problems. *Journal of Algorithms*, 29(1):174 – 200. Citado na pág. 7

Ibaraki(1976) T. Ibaraki. Integer programming formulation of combinatorial optimization problems. *Discrete Mathematics*, 16:39–52. Citado na pág. 65

Karp(1972) R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, páginas 85–103. Citado na pág. 8

Konc e Janezic(2007) J. Konc e D. Janezic. An improved branch and bound algorithm for the maximum clique problem. *MATCH Communications in Mathematical and in Computer Chemistry*, 58:569–590. Citado na pág. 17, 27

Lanka e Goerdt(2004) André Lanka e Andreas Goerdt. An approximation hardness result for bipartite clique. *Electronic Colloquium on Computational Complexity (ECCC)*, páginas –1–1. Citado na pág. 9

Madeira e Oliveira(2004) S. C. Madeira e A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45. Citado na pág. 7

McCardle *et al.***(2007)** Kevin F. McCardle, Kumar Rajaram e Christopher S. Tang. Bundling retail products: Model and analysis. *European Journal of Operational Research*, 177:1197–1217. Citado na pág. 6

McCormick(1983) G.P. McCormick. *Nonlinear programming: theory, algorithms and applications*. John Wiley & sons. Citado na pág. 65

Mehrotra e Trick(1996) A. Mehrotra e M. A. Trick. A column generation approach for graph coloring. *Informs Journal on Computing*, 8(4):344 – 354. Citado na pág. 43

Notebaart *et al.***(2008)** R. A. Notebaart, B. Teusink, R.J. Siezen e B. Papp. Co-regulation of metabolic genes is better explained by flux coupling than by network distance. *PLoS Comput Biol*, 4(1):e26. Citado na pág. 6

Nussbaum *et al.***(2010)** Doron Nussbaum, Shuye Pu, Jörg-Rüdiger Sack, Takeaki Uno e Hamid Zarrabi-Zadeh. Finding maximum edge bicliques in convex bipartite graphs. Em My Thai e Sartaj Sahni, editors, *Computing and Combinatorics*, volume 6196 of *Lecture Notes in Computer Science*, páginas 140–149. Springer Berlin / Heidelberg. Citado na pág. 8

Östergård(2002) P. R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120:197–207. Citado na pág. 17, 27

Padberg(1989) M. Padberg. The boolean quadric polytope: some characteristics, facets and relatives. *Mathematical Programming*, 45:139–172. Citado na pág. 65

Peeters(2003) R. Peeters. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, páginas 651–654. Citado na pág. 8

Ryan e Foster(1981) D. M. Ryan e B. A. Foster. An integer approach to scheduling. *Computer Scheduling of Public Transport*, páginas 269–280. Citado na pág. 43, 49

Schrijver(1986) Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley. Citado na pág. 1, 42

Schuster e Hilgetag(1994) S. Schuster e C. Hilgetag. On elementary flux modes in biochemical reaction systems at steady state. *Journal of Biological Systems*, 2(2):165–182. Citado na pág. 6

Sherali e Adams(1999) H.D. Sherali e W. P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Kluwer Academic Publishers. Citado na pág. 65

Stremersch e Tellis(2002) Stefan Stremersch e Gerard J. Tellis. Strategic bundling of products and prices: a new synthesis for marketing. *Journal of Marketing*, 66:55–72. Citado na pág. 6

Swaminathan e Tayur(1998) J. M. Swaminathan e S. R. Tayur. Managing broader product lines through delayed differentiation using vanilla boxes. *Manage. Sci.*, 44(12):161–172. Citado na pág. 6, 7

Tan(2008) J. Tan. Inapproximability of maximum weighted edge biclique and its applications. Em *TAMC'08: Proceedings of the 5th international conference on Theory and applications of models of computation*, páginas 282–293, Berlin, Heidelberg. Springer-Verlag. Citado na pág. 10

Terzer e Stelling(2008) M. Terzer e J. Stelling. Large-scale computation of elementary flux modes with bit pattern trees. *Bioinformatics*, 24(19):2229–2235. Citado na pág. 6

Tomita e Seki(2003) E. Tomita e T. Seki. *Discrete Mathematics and Theoretical Computer Science*, volume 2731/2003 of *Lecture Notes in Computer Science*, chapter An Efficient Branch-and-Bound Algorithm for Finding a Maximum Clique, páginas 279–289. Springer Berlin / Heidelberg. Citado na pág. 17, 27

Vanderbeck e Wolsey(1996) F. Vanderbeck e L. A. Wolsey. An exact algorithm for IP column generation. *Operations Research Letters*, 19(4):151 – 159. Citado na pág. 48, 49

Warren e Hicks(2007) Jeffrey S. Warren e Illya V. Hicks. Combinatorial branch-and-bound for the maximum weight independent set problem, 2007. Citado na pág. 17

Wilhelm(2001) W. E. Wilhelm. A technical review of column generation in integer programming. *Optimization and Engineering*, páginas 159–200. Citado na pág. 13, 43

Wolsey(1998) Laurence A. Wolsey. *Integer Programming*. Wiley - Interscience Series in Discrete Mathematics and Optimization. Citado na pág. 1, 42, 48

Yannakakis(1978) M. Yannakakis. Node-and edge-deletion NP-complete problems. Em *Proceedings of the tenth annual ACM symposium on Theory of computing*, STOC '78, páginas 253–264, New York, NY, USA. ACM. Citado na pág. 7