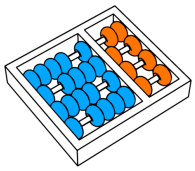
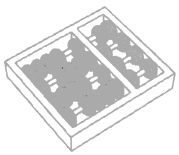


Eduardo Theodoro Bogue

"O Problema da Máxima Interseção de k-Subconjuntos"

CAMPINAS
2014



Universidade Estadual de Campinas
Instituto de Computação

Eduardo Theodoro Bogue

"O Problema da Máxima Interseção de k-Subconjuntos"

Orientador(a): **Prof. Dr. Cid Carvalho de Souza**
Co-Orientador(a): **Prof. Dr. Eduardo Candido Xavier**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Ciência da Computação.

Este exemplar corresponde à versão final da Dissertação defendida por Eduardo Theodoro Bogue, sob orientação de Prof. Dr. Cid Carvalho de Souza.

Assinatura do Orientador(a)

CAMPINAS
2014

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

B634p

Bogue, Eduardo Theodoro, 1990-
O problema da máxima interseção de k-subconjuntos / Eduardo Theodoro Bogue. – Campinas, SP : [s.n.], 2014.

Orientador: Cid Carvalho de Souza.
Coorientador: Eduardo Candido Xavier.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização combinatória. 2. Programação linear. 3. Algoritmos em grafos. I. Souza, Cid Carvalho de,1963-. II. Xavier, Eduardo Candido,1979-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Maximum k-subset problem
Palavras-chave em inglês:
Combinatorial optimization
Linear programming
Graph algorithms
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Cid Carvalho de Souza [Orientador]
Yuri Abitbol de Menezes Frota
Fabio Luiz Usberti
Data de defesa: 27-03-2014
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Defesa de Dissertação de Mestrado em Ciência da Computação, apresentada pelo(a) Mestrando(a) **Eduardo Theodoro Bogue**, aprovado(a) em **27 de março de 2014**, pela Banca examinadora composta pelos Professores Doutores:

Prof(a). Dr(a). Yuri Abitbol de Menezes Frota
Titular

Prof(a). Dr(a). Fabio Luiz Usberti
Titular

Prof(a). Dr(a). Cid Carvalho de Souza
Presidente

Instituto de Computação
Universidade Estadual de Campinas

O Problema da Máxima Interseção de k-Subconjuntos

Eduardo Theodoro Bogue[1]

27 de março de 2014

Banca Examinadora:

- Prof. Dr. Cid Carvalho de Souza (Supervisor/*Orientador*)
- Prof. Dr. Yuri Abitbol de Menezes Frota
  Instituto de Ciência da Computação - Universidade Federal Fluminense
- Prof. Dr. Fabio Luiz Usberti
  Instituto de Computação - Universidade Estadual de Campinas
- Prof. Dr. Orlando Lee
  Instituto de Computação - Universidade Estadual de Campinas (*Suplente Interno*)
- Prof. Dr. Luidi Simonetti
  Instituto de Ciência da Computação - Universidade Federal Fluminense (*Suplente Externo*)

[1]Suporte Financeiro: Bolsa FAPESP (processo 2012/08298-6) 2012–2014

Abstract

In this project, we study the Maximum k-Subset Intersection problem (kMIS). Given an integer k, a ground set U and a collection $\mathcal{S}$ of subsets of U, the kMIS problem is to select k distinct subsets $S_1, S_2, \ldots, S_k$ in $\mathcal{S}$ whose intersection size $|S_1 \cap S_2 \cap \cdots \cap S_k|$ is maximum. The kMIS problem is NP-hard and hard to approximate and occurs in areas of applications such as computational biology and data privacy. To the best of our knowledge, no exact method was proposed to solve this problem.

In this work, we introduce five integer linear programming formulations for the problem, three using a simple *Branch-and-Bound* method and two using a *Branch-and-Cut* method. We also present a greedy heuristic and a metaheuristic $GRASP$ developed in order to generate good lower bounds. The heuristic $GRASP$ developed was able to find solutions very close to the optimal ones. Furthermore, we introduce a very efficient preprocessing procedure to reduce the size of the input. Computational experiments were performed in order to analyze the performance of the integer linear programming models in question, showing that the *Branch-and-Cut* models performed better.

Resumo

Neste projeto, nós estudamos o Problema da Máxima Interseção de k-Subconjuntos (kMIS). Dado um inteiro k, um conjunto base U e uma coleção S de subconjuntos de U, o problema kMIS consiste em selecior k subconjuntos distintos $S_1, S_2, \ldots, S_k$ em S cujo tamanho da interseção de $|S_1 \cap S_2 \cap \ldots \cap S_k|$ seja máxima. Trata-se de um problema $\mathcal{NP}$-difícil e difícil de ser aproximado que ocorre em aplicações de áreas como biologia computacional e privacidade de dados. Até o nosso conhecimento, nenhum método exato foi proposto para resolver este problema.

Neste trabalho, introduzimos cinco formulações de programação linear inteira para o problema, sendo três baseadas no método de *Branch-and-Bound* e duas no método de *Branch-and-Cut*. Além disso, uma heurística gulosa e uma meta-heurística $GRASP$ foram desenvolvidas com o intuito de gerar bons limitantes inferiores. A heurística $GRASP$ desenvolvida foi capaz de encontrar soluções muito próximas da solução ótima. Ademais, introduzimos um método muito eficiente de pré-processamento para reduzir o tamanho da entrada. Experimentos computacionais foram realizados de forma a analisar o desempenho dos modelos de programação linear inteira em questão, demonstrando que os modelos baseados no método de *Branch-and-Cut* obtiveram melhores resultados.

Agradecimentos

Primeiramente gostaria de agradecer meus orientadores, Prof. Dr. Cid Carvalho de Souza e Prof. Dr. Eduardo Candido Xavier, que me guiaram durante o mestrado para que eu pudesse chegar a conclusão desse etapa e me fizeram evoluir não somente como profissional mas também como pessoa. Ao Prof. Dr. Alexandre Freire, que colaborou com o desenvolvimento desse trabalho.

Aos meus amigos que acompanharam minha trajetória durante esses dois anos, sempre me apoiando e proporcionando momentos de descontração, muitas vezes necessários em momentos difíceis do curso. Não há como listar todos, porém, abro uma exceção para os amigos Leandro, Hudson, Anderson e Daniela (Campo Grande), Adriano, Ana, Magnum, Pedro, Maycon e Carla (Campinas) e Melissa (Inglaterra).

Com muito carinho, gostaria de agradecer também aos meus pais, que sempre fizeram o possível e o impossível por mim, se sacrificando para que eu pudesse ter uma boa educação e me apoiando em todos os momentos da minha vida.

Por fim, gostaria de agradecer a minha namorada, Jussara, que sempre foi uma fonte inesgotável de força e palavras de apoio, principalmente na reta final desse trabalho.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Problemas de otimização combinatória ocorrem em diversas situações práticas do dia a dia, como por exemplo em áreas de logística de transporte e distribuição, alocação de recursos, entre outras. Problemas de corte e empacotamento e do caixeiro viajante são dois exemplos de problemas clássicos e muito bem estudados em otimização combinatória. Novas variantes destes problemas surgem à medida que aplicações exigem restrições adicionais ou relaxam algumas restrições do problema. Em particular, neste trabalho estamos interessados em uma variante do Problema de Subconjunto de Interseção Máxima [Clifford and Popa(2011)], denominado o Problema da Máxima Interseção de k-Subconjuntos (kMIS).

O kMIS é definido do seguinte modo: dada uma coleção $\{S_1, \ldots, S_n\}$ de n subconjuntos sobre um conjunto finito de elementos $\{e_1, \ldots, e_m\}$, e um inteiro positivo k, o objetivo é selecionar exatamente k subconjuntos cujo tamanho da interseção destes subconjuntos seja máxima. Utilizando a terminologia de teoria dos grafos, o problema pode ser descrito da seguinte maneira: dado um grafo bipartido $G = (V = L \cup R, E)$, em que L representam os subconjuntos, R os elementos e uma aresta liga um vértice i associado a $S_i \in L$ a um vértice j correspondente a $e_j \in R$ se e somente se $e_j \in S_i$, o objetivo é encontrar uma biclique (um subgrafo bipartido completo) B, subgrafo de G, tal que $|V(B) \cap L| = k$ e $|V(B) \cap R|$ seja máxima.

Na Figura 1.1 é mostrado o grafo bipartido referente à instância do problema em que $k = 2$, $L = \{S_1, S_2, S_3, S_4\}$ e $R = \{1, 2, 3, 4, 5\}$, onde $S_1 = \{2, 3\}$, $S_2 = \{1, 2, 3\}$, $S_3 = \{1, 2, 3, 4, 5\}$ e $S_4 = \{2, 3, 5\}$.

A motivação inicial para investigar o kMIS origina-se de sua aplicação no controle de dados de pacientes em hospitais [Vinterbo(2002)]. Seja L o conjunto de pacientes de um hospital e R um conjunto de dados dos pacientes. Para evitar que dados divulgados de pacientes não possam ser utilizados para identificá-los, é permitido revelar algum dado somente se k pacientes possuam este mesmo dado, de maneira que k seja grande o su-

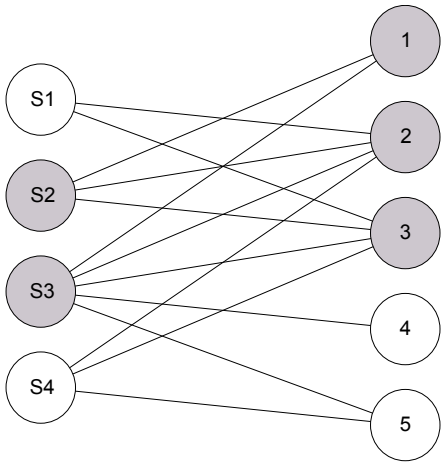


Figura 1.1: Para $k = 2$, a solução ótima é obtida com os subconjuntos S_2 e S_3, dada por $\{1, 2, 3\}$, ou S_3 e S_4, dada por $\{2, 3, 5\}$.

ficiente para garantir a preservação da privacidade. Com isto, o objetivo do problema torna-se em encontrar k pacientes que possuam a maior quantidade possível de dados em comum.

Em [Nussbaum et al.(2010)Nussbaum, Pu, Sack, Uno, and Takeaki], uma aplicação do kMIS na análise de DNA Microarray é mencionada. Dados de Microarray são usualmente apresentados como uma matriz bidimensional em que as linhas representam genes e as colunas condições de teste. Cada entrada $[i, j]$ da matriz binária representa o nível de expressão de um dado gene i sobre uma determinada condição j. Para melhor entender processos biológicos de uma célula, biólogos geralmente tentam encontrar relações entre subconjuntos de genes e subconjuntos de condições. Contudo, esta análise se torna inviável de ser realizada sem a utilização de recursos computacionais. Este problema pode ser modelado como uma instância do problema kMIS de forma que L é o conjunto de genes e R o conjunto de condições, e o objetivo é selecionar k genes que possuam o maior número de condições em comum.

Neste trabalho, a seguinte aplicação para o kMIS foi considerada: cada vértice em L representa um artista musical (cantor, banda, etc...), e cada vértice em R representa uma pessoa, de modo que existe uma aresta ligando um vértice $j \in R$ a um vértice $i \in L$ se a pessoa j é fã do artista musical i. O objetivo do problema é encontrar um conjunto de k artistas musicais (para fazer parte de um show de rock, por exemplo) com o maior número de fãs em comum, o que permite a realização de um show em que as pessoas gostem da maioria das bandas escolhidas.

A proposta desta dissertação é investigar o kMIS do ponto de vista algorítmico. Mais precisamente, deseja-se avaliar a adequabilidade de algoritmos exatos baseados em

técnicas de Programação Linear Inteira (PLI) e de heurísticas como algoritmos Gulosos e *Greedy Randomized Adaptive Search Procedure* (*GRASP*) para resolver o problema. Além da motivação oriunda das aplicações práticas já mencionadas acima, a direção de pesquisa escolhida nesta dissertação se justifica pelas seguintes razões. Primeiramente, o problema foi provado ser $\mathcal{NP}$-difícil por Vinterbo [Vinterbo(2002)] e recentemente Xavier [Xavier(2012)] demonstrou que o problema é difícil de ser aproximado. Ademais, não foi encontrado na literatura nenhum trabalho onde tenha sido feito um tratamento algorítmico do problema, não havendo, portanto, qualquer previsão sobre para qual tamanho de instância se pode resolvê-lo na prática.

O restante deste capítulo é dedicado à apresentação da terminologia e de conceitos básicos a serem utilizados no decorrer desta dissertação. Na Seção 1.1 fazemos uma breve revisão bibliográfica de problemas relacionados com o kMIS. Na Seção 1.2 caracterizamos um problema de Programação Linear (PL) e sua diferença para um problema de Programação Linear Inteira (PLI). Na Seção 1.3 apresentamos o algoritmo de *Branch-and-Bound* (**B**&**B**), um dos principais algoritmos usados para resolver problemas de otimização combinatória e PLI. Na Seção 1.4 são apresentados alguns resultados básicos de teoria poliedral com o intuito de se avaliar a força das desigualdades utilizadas em um modelo de PLI. Na Seção 1.5 descrevemos o algoritmo de planos de corte, outro método muito utilizado para resolver problemas de PLI. Por fim, na Seção 1.7 apresentamos a organização do texto restante desta dissertação.

1.1 Problemas Relacionados

Nesta seção é feita uma revisão dos trabalhos anteriores que apresentam problemas relacionados ao kMIS. Segundo nossa pesquisa bibliográfica, foi [Vinterbo(2002)] o primeiro a citar este problema na literatura. Em relação à complexidade computacional, em [Vinterbo(2002)] e [Xavier(2012)] é demonstrado que o problema é $\mathcal{NP}$-difícil e em [Xavier(2012)], é também demonstrado que o problema é inaproximável, de modo que o kMIS não admite algoritmo $\frac{1}{N^\epsilon}$-aproximado, sendo N o tamanho da entrada e ϵ uma constante. Para provar a complexidade do problema, [Xavier(2012)] realiza uma redução a partir do problema *Maximum Edge Biclique (MEB)*, provado ser $\mathcal{NP}$-difícil por [Peeters(2003)]. O problema MEB é definido do seguinte modo: dado um grafo bipartido $G = (L \cup R, E)$, o objetivo do problema é encontrar uma biclique B, subgrafo de G, que possua cardinalidade máxima de arestas. A Figura 1.2 exibe uma instância do problema MEB.

Uma ideia natural para resolvermos o problema MEB é resolvermos o kMIS, para $k = 1, 2, \ldots, |L|$, e retornarmos a solução que possuir o número máximo de arestas. Este processo nos leva intuitivamente a solução ótima do problema MEB. Nota-se também que

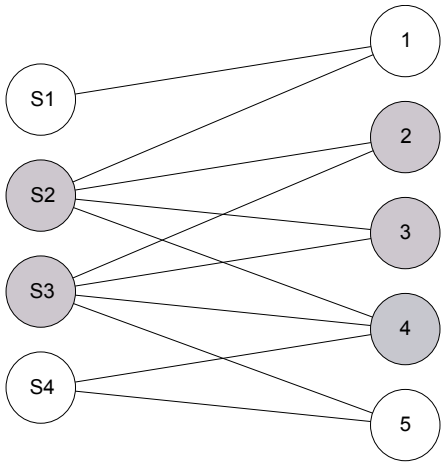


Figura 1.2: Instância do problema MEB em que os vértices em cinza representam a maior biclique presente no grafo.

o problema kMIS é um caso particular do problema MEB com a restrição adicional de que $|V(B) \cap L| = k$.

Em [Acuña et al.(2014)Acuña, Ferreira, Freire, and Moreno] são propostas formulações em programação linear inteira para o problema MEB. A partir das formulações propostas são realizados experimentos computacionais utilizando instâncias provenientes de aplicações em bioinformática e instâncias geradas aleatoriamente.

Em [Nussbaum et al.(2010)Nussbaum, Pu, Sack, Uno, and Takeaki] é apresentado um algoritmo para o problema MEB no caso especial onde $G = (L \cup R, E)$ é um grafo *convexo* em R (G é dito ser *convexo* em R se existir uma ordenação de vértices em R tal que, para qualquer vértice $v \in L$, os vértices adjacentes a v são consecutivos em R). O algoritmo apresentado possui complexidade $O(m \log^3 m \log \log m)$ em tempo e $O(m)$ em espaço, onde $m = |E|$.

Em [Alexe et al.(2004)Alexe, Alexe, Crama, Foldes, Hammer, and Simeone] e [Dias et al.(2005)Dias, de Figueiredo, and Szwarcfiter], os autores apresentam algoritmos exponenciais capazes de gerar todas as bicliques maximais de um grafo G, tornando assim possível encontrar uma solução para o problema MEB através de uma simples busca pela biclique de maior número de arestas. Contudo, optamos por estudar o problema através de modelos de programação linear inteira em virtude que testes preliminares mostraram resultados superiores dos modelos PLI em relação aos algoritmos exponencias que geram todas as bicliques maximais de um grafo. Dado o estreito vínculo entre o kMIS e o MEB, veremos no Capítulo 2 três modelos de PLI para o kMIS baseado nas formulações discutidas em [Acuña et al.(2014)Acuña, Ferreira, Freire, and Moreno].

1.2 Conceitos Básicos de Programação Linear Inteira

O problema de programação linear consiste em maximizar ou minimizar uma função linear, denominada de função objetivo, sobre uma região no espaço descrita por um conjunto de desigualdades também lineares, chamadas restrições do problema. Uma instância do problema de programação linear (PL) pode ser descrito da seguinte maneira:

$$\begin{aligned} &\text{Maximize} && cx \\ \text{sujeito a} \quad &&& Ax \leq b, && (1.1) \\ &&& x \in \mathbb{R}^n_+ && (1.2) \end{aligned}$$

onde A é uma matriz m por n, c um vetor linha n-dimensional, b um vetor coluna m-dimensional e x um vetor coluna n-dimensional de variáveis. Adicionando a restrição de que as variáveis em x devem assumir valores inteiros, ou seja, $x \in \mathbb{Z}^n_+$, temos então uma instância do problema de problema de programação linear inteira (PLI). Como as soluções do PL devem satisfazer um sistema linear, sabemos que o conjunto de soluções viáveis forma um poliedro.

O problema de PLI é mais difícil que o problema de PL, visto que uma instância do problema de PL pode ser resolvida por algoritmos polinomiais, como por exemplo o método dos pontos interiores [Karmarkar(1984)], enquanto o problema de PLI é $\mathcal{NP}$-difícil.

Seja P um conjunto de pontos de coordenadas inteiras que são viáveis para um modelo de PLI. A envoltória convexa de P é definida como o menor poliedro que contém todos os pontos de P. A Figura 1.3 mostra um exemplo em que os pontos em cinza representam soluções viáveis para um problema, sendo o menor polígono que contém todos os pontos em cinza a envoltória convexa e o polígono externo uma formulação válida para o problema. Uma formulação válida é aquela que contém todos os pontos em P e não contém nenhum ponto inteiro externo a P.

Um dos algoritmos mais utilizados para se resolver problemas de PLI é o algoritmo de *Branch-and-Bound* (**B**&**B**), que se vale da relaxação linear do problema para gerar sucessivos limitantes superiores e inferiores para o problema até que os dois atinjam o mesmo valor, chegando assim a uma solução ótima do problema. A relaxação linear de um problema de PLI corresponde à substituição das restrições de integralidade, $x \in \mathbb{Z}^n_+$, por restrições $x \in \mathbb{R}^n_+$, gerando assim seu problema em PL correspondente. Esta transformação é conveniente, pois vimos que PLs podem ser computados em tempo polinomial, o que permite calcular rapidamente limitantes duais para o problema que se quer resolver.

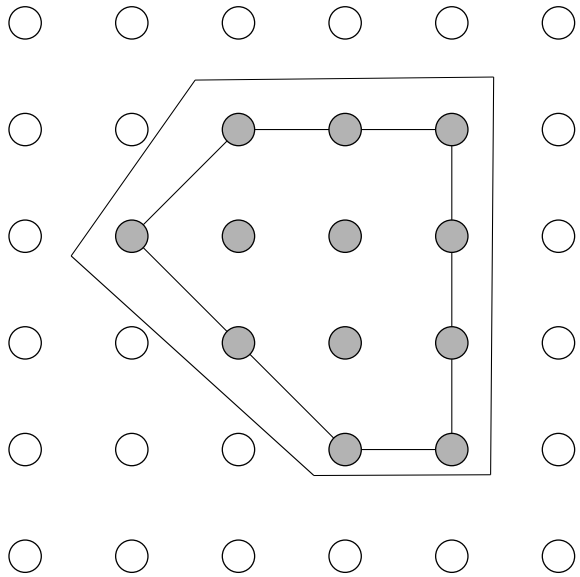


Figura 1.3: Envoltória convexa e formulações válidas de um problema.

1.3 Método de *Branch-and-Bound*

Como dito anteriormente, o paradigma de **B**&**B** é um dos principais métodos para resolvermos problemas de otimização combinatória, sendo comumente utilizado na resolução de modelos de PLI. Considere o seguinte problema:

$$z = \max\{cx : x \in S\} \tag{1.3}$$

Seja $S = S_1 \cup \ldots \cup S_c$ uma decomposição de S em conjuntos menores, $z^p = \max\{cx : x \in S_p\}$ para $p = 1, \ldots, c$ e $z = \max_p z^p$. O algoritmo baseia-se na observação de que a enumeração das soluções inteiras do conjunto S possui uma estrutura de árvore. Podemos considerar, por exemplo, um modelo que possui uma variável inteira x_1, e uma variável binária x_2, tal que $1 \leq x_1 \leq 3$ e $0 \leq x_2 \leq 1$. A Figura 1.4 exibe a enumeração de todas as soluções para estas variáveis.

A estrutura da Figura 1.4 pode ser vista como uma árvore onde inicialmente o conjunto S é decomposto nos subconjuntos $S_1 = \{x \in S : x_1 = 1\}$, $S_2 = \{x \in S : x_1 = 2\}$ e $S_3 = \{x \in S : x_1 = 3\}$, então $S_{10} = \{x \in S_1 : x_2 = 0\}$, e $S_{11} = \{x \in S_1 : x_2 = 1\}$, e assim por diante. Os nós folhas representam todas as soluções enumeradas, possuindo 6 delas no total: (3 possíveis valores de x_1) $\times$ (2 possíveis valores de x_2).

A ideia principal do algoritmo de *branch-and-bound* consiste em minimizar o crescimento da árvore de enumeração, à medida que esta é construída, através da realização de podas em nós em que se é possível saber previamente que estes não nos levarão a uma

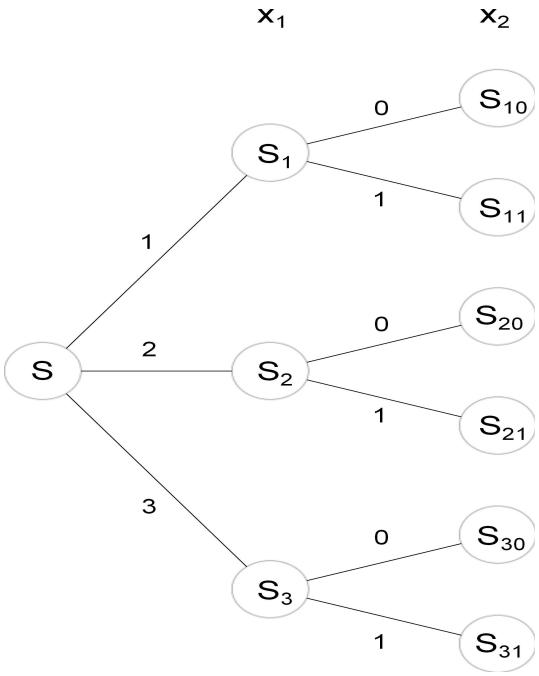


Figura 1.4: Exemplo de árvore completa de enumeração.

solução ótima do problema. Isto é desejável pois a construção da árvore de enumeração completa é normalmente inviável devido ao número exponencial de soluções. Seja $\overline{z}^p$ e $\underline{z}^p$ um limitante superior e inferior para z_p, então temos que os valores de $\overline{z} = \max_p \overline{z}^p$ e $\underline{z} = \max_p \underline{z}^p$ representam limitantes superiores e inferiores em z, respectivamente.

Podemos podar um nó da árvore de enumeração se este estiver em algum dos três casos descritos em 1.3.1, 1.3.2 e 1.3.3.

1.3.1 Poda por Otimalidade:

Analisando a árvore de enumeração da Figura 1.5, podemos observar que o limitante superior e inferior em S_2 são ambos iguais a 10, o que implica que S_2 pode ser podado, visto que conhecemos uma solução de custo 10 e a melhor solução possível para S_2 também possui valor 10.

1.3.2 Poda por limitante:

Na árvore de enumeração da Figura 1.6, temos que a solução de maior custo em S_1 possui no máximo o valor 15, enquanto analisando o nó S_2 verificamos que já é conhecida uma solução de ao menos custo 18. Logo, o nó S_1 pode ser podado.

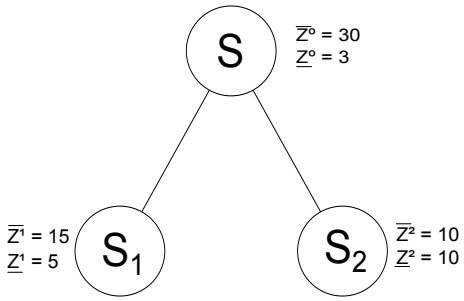


Figura 1.5: Exemplo em que o nó S_2 pode ser podado da árvore por otimalidade.

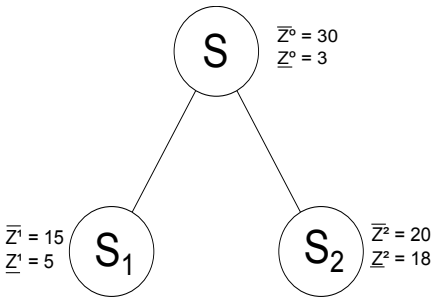


Figura 1.6: Exemplo onde o nó S_1 pode ser podado da árvore por limitante.

1.3.3 Poda por inviabilidade:

Dada a desigualdade $10x_1+5x_2 \leq 9$ e a árvore de enumeração da Figura 1.7, se x_1 assumir o valor 1 temos que a desigualdade é violada. Logo, podemos podar o nó S_2 da árvore de enumeração, visto que todas as soluções provenientes deste nó não são soluções viáveis do problema.

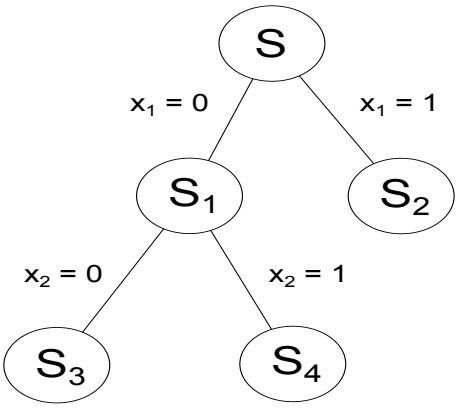


Figura 1.7: Exemplo em que o nó S_2 pode ser podado da árvore.

Em um dado momento da execução do algoritmo, os nós da árvore que não estão

podados são chamados de nós ativos. O algoritmo encerra sua execução quando não existem mais nós ativos na árvore, retornando assim uma solução de valor ótimo z do problema.

1.4 Conceitos da Teoria Poliedral

Nesta seção apresentamos alguns conceitos básicos de teoria poliedral a serem utilizados durante o decorrer desta dissertação. Inicialmente formalizamos conceitos como dimensão, face e faceta de um poliedro e em seguida descrevemos dois métodos para se avaliar a força de uma desigualdade.

1.4.1 Conceitos Básicos

DefiniÃ§Ã$o 1.1. *Um conjunto de pontos $x^1, \ldots, x^p \in \mathbb{R}^n$ é linearmente independente se a única solução para $\sum_{i=1}^{p} \alpha_i x_i = 0$ é $\alpha_i = 0$ para todo $i \in \{1, \ldots, p\}$.*

DefiniÃ§Ã$o 1.2. *Seja A uma matriz de dimensão $m \times n$, o posto de A corresponde ao número de linhas (ou colunas) linearmente independentes.*

DefiniÃ§Ã$o 1.3. *Um conjunto de pontos $x^1, \ldots, x^p \in \mathbb{R}^n$ é afim-independente se a única solução para $\sum_{i=1}^{p} \alpha_i x_i = 0$, $\sum_{i=1}^{p} \alpha_i = 0$ é $\alpha_i = 0$ para todo $i \in \{1, \ldots, p\}$.*

DefiniÃ§Ã$o 1.4. *Seja $P = \{x^1, x^2, \ldots, x^p\}$ um conjunto de pontos em $\mathbb{R}^n$. A envoltória convexa de P é dada por:*

$$conv(P) = \{\sum_{i=1}^{p} \alpha_i x^i : \sum_{i=1}^{p} \alpha_i = 1, x^i \in P, \alpha_i \in \mathbb{R}_+, i = 1, \ldots, p\} \tag{1.4}$$

DefiniÃ§Ã$o 1.5. *Um poliedro $P \subseteq \mathbb{R}^n$ é dito ser monótono se e somente se para todo $x \in P$ e todo $y \in \mathbb{R}^n_+$ com $y \leq x$, tem-se que $y \in P$.*

DefiniÃ§Ã$o 1.6. *Uma desigualdade $\pi x \leq \pi_0$ é válida para um poliedro se todos os pontos deste poliedro a satisfazem.*

DefiniÃ§Ã$o 1.7. *Um conjunto $D \subseteq \mathbb{R}^n$ possui dimensão d se a cardinalidade de todo subconjunto maximal de vetores afim independentes em D é $n + 1$. Se $dim(D) = dim(\mathbb{R}^n) = n$, dizemos então que D possui dimensão cheia.*

DefiniÃ§Ã$o 1.8. *Seja $(A^=, b^=)$ o conjunto de igualdades de um poliedro $P \subseteq \mathbb{R}^n$. A dimensão de P é dada por $dim(P) = n - posto(A^=, b^=)$.*

DefiniÃ§Ã$o 1.9. *Se $\pi x \leq \pi_0$ e $\mu x \leq \mu_0$ são duas igualdades válidas para um poliedro P, dizemos que $\pi x \leq \pi_0$ domina $\mu x \leq \mu_0$ se existir $u > 0$ tal que $\pi \geq u\mu$ e $\pi_0 \leq u\mu_0$.*

DefiniÃ§Ã$o 1.10. *Seja $\alpha x \leq \alpha_0$ uma desigualdade válida para um poliedro P. O conjunto $F = \{x \in P : \alpha x = \alpha_0\}$ é chamado de face de P (dizemos que a desigualdade $\alpha x \leq \alpha_0$ define face F em P). Uma face é dita ser própria de $F \neq \emptyset$ e $F \neq P$. Se $dim(F) = dim(P) - 1$ então F é dita ser uma faceta de P.*

A importância de se conhecer facetas de um poliedro P advém do fato de que, um sistema linear minimal descrevendo P deve conter exatamente uma desigualdade linear para cada faceta de P. Pela definição acima, percebe-se que a dimensão de uma faceta é a maior possível para uma face qualquer. Assim, associa-se a força de uma desigualdade à dimensão da face associada. Nesse sentido, as desigualdades definidoras de facetas são as mais fortes que se pode obter, de modo que uma desigualdade que define faceta não é dominada por nenhuma outra. O sucesso do método de *Branch-and-Cut*, utilizado na resolução de problemas de PLI, é ligado a utilização de desigualdades fortes.

Um dos métodos mais usuais para se caracterizar se uma desigualdade define faceta para um certo poliedro P deriva da própria definição de dimensão. É o denominado método direto, que consiste em exibir exatamente $dim(P)$ vetores afim-independentes que satisfaçam a inequação na igualdade. Nesta dissertação, utilizamos o método direto em todas as provas de caracterização de facetas.

Contudo, encontrar pontos afim-independentes para caracterizar que uma desigualdade define uma faceta pode ser uma tarefa difícil. Neste caso, outro método muito utilizado na literatura para definir se certa inequação define uma faceta é o método indireto. O método indireto baseia-se no Teorema 1.1.

Teorema 1.1. *Seja $(A^=, b^=)$ o conjunto de igualdades de um poliedro $P \subseteq \mathbb{R}^n$ e seja $F = \{x \in P : \pi x = \pi_0\}$ uma face própria de P. As seguintes afirmações são equivalentes:*

1. *F é uma faceta de P.*
2. *Se $\lambda x = \lambda_0$ para todo $x \in F$, então*

$$(\lambda, \lambda_0) = (\alpha\pi + \upsilon A^=, \alpha\pi_0 + \upsilon b^=) \text{ para algum } \alpha \in \mathbb{R}_+ \text{ e algum } \upsilon \in \mathbb{R}^{|A^=|}$$

Todavia, em casos em que o poliedro P não possui dimensão cheia o método indireto pode apresentar dificuldades, visto que uma faceta pode ser definida por um número infinito de hiperplanos. Devido a isto, quando o poliedro não possui dimensão cheia opta-se geralmente por estudar o poliedro monótono $P' \supset P$, no qual toda solução ótima de P é também uma solução ótima de P', sendo P' de dimensão cheia.

1.5 Algoritmo de Planos de Corte

A ideia do algoritmo de planos de corte é adicionar desigualdades que removam pontos não-inteiros do poliedro correspondente à relaxação linear do modelo de PLI associado ao problema que se quer resolver. Com a adição de sucessivas desigualdades válidas, ocorre um fortalecimento desta relaxação na medida em que, a cada desigualdade adicionada, o poliedro da relaxação reduz de tamanho, aproximando-se iterativamente da envoltória convexa das soluções inteiras. Um algoritmo de **B**&**B** que utiliza-se da adição de planos de corte para remover soluções não-inteiras do problema é denominado de *Branch-and-Cut*.

Seja P o conjunto de soluções viáveis de um problema de PLI, $\mathcal{F}$ um conjunto de desigualdades fortes com respeito a $conv(P)$ e uma rotina, denominada rotina de separação, cujo objetivo é determinar se existe alguma desigualdade em $\mathcal{F}$ que seja violada por um ponto que não esteja em $conv(P)$. A ideia do algoritmo de **B**&**C** é baseada na utilização do método de **B**&**B**, de modo que a cada nó da árvore de enumeração é executado um algoritmo de planos de corte.

Seja s^i a solução ótima obtida pela relaxação linear do nó i na árvore de enumeração. Temos então que se s^i contém elementos fracionários, podemos utilizar a rotina para encontrar uma desigualdade válida em $\mathcal{F}$ que a solução s^i não satisfaça. Em caso positivo, a desigualdade violada é inserida na formulação e a relaxação linear do nó é resolvida novamente. Este processo pode ser repetido enquanto for possível encontrar uma desigualdade válida que corte a solução fracionária obtida pela relaxação linear de um nó ou enquanto a adição desta desigualdade esteja melhorando consideravelmente o limitante superior do problema. Caso nenhuma desigualdade seja adicionada, escolhe-se uma variável fracionária em s^i para a realização do *branching* na árvore de enumeração.

A Figura 1.8 mostra um exemplo de uma desigualdade que remove o ponto não-inteiro x^* da solução mantendo todos os pontos inteiros viáveis do problema.

Decidir se uma desigualdade representa um plano de corte em alguns casos é um problema $\mathcal{NP}$-difícil. Na prática, são utilizadas famílias de desigualdades válidas e, ao verificarmos que o ponto obtido pela relaxação de um nó não é inteiro, verificamos através de uma heurística se o ponto viola alguma desigualdade válida desta família.

1.6 Notações e Definições

Nesta seção, introduzimos as notações e definições utilizadas nesta dissertação.

Seja $G = (V, E)$ um grafo bipartido e L e R as duas partes da bipartição de V (ou seja, $V = L \cup R$, $L \cap R = \emptyset$ e $E \subseteq L \times R$). Por simplicidade, dizemos que G é (L, R)-bipartido e, dada uma aresta $\{u, v\} \in E$, supomos, por convenção, que $u \in L$, $v \in R$ e escrevemos uv (sem chaves, ou seja, o rótulo a esquerda corresponde ao vértice

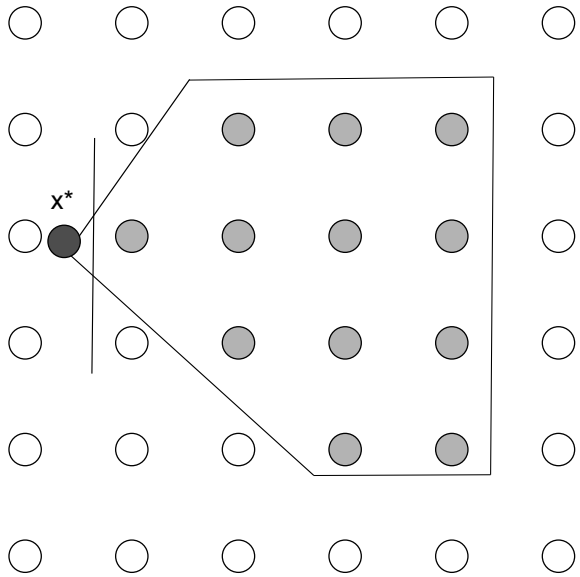


Figura 1.8: Exemplo de um plano de corte removendo a solução ótima não-inteira da relaxação linear.

em L, e o vértice a direita corresponde ao vértice em R). Dado um subgrafo B de G, denotamos seu conjunto de vértices por V_B, seu conjunto de arestas por E_B e dizemos que B é um grafo (L_B, R_B)-bipartido, em que $L_B = V_B \cap L$ e $R_B = V_B \cap R$. Dado um subconjunto de vértices $S \subset V$, o subgrafo de G *induzido* por S é o grafo $G' = (V', E')$, tal que $V' = S$ e $E' = \{uv \in E \mid u \in V' \text{ e } v \in V'\}$. Dado um vértice u em V, seja $N(u) = \{v \in V \mid \{u, v\} \in E\}$ o conjunto de *vizinhos* de u e $\overline{N}(u)$ o conjunto de vértices que não são vizinhos de u na parte oposta da bipartição (ou seja, se $u \in L$ então $\overline{N}(u) = R \setminus N(u)$, caso contrário $\overline{N}(u) = L \setminus N(u)$). Dado um subconjunto S de V, $\bigcap(S) = \bigcap_{u \in S} N(u)$ é o conjunto de vizinhos que todos os vértices em S tem em comum. Denotamos por $\overline{E} = \{\{u, v\} \mid u \in L \text{ e } v \in \overline{N}(u)\}$ o conjunto de arestas que faltam para G se tornar um grafo bipartido completo. Dizemos que B é uma *biclique* se $uv \in E_B$, para todo $u \in L_B$ e $v \in R_B$. Denotamos por $\chi(B)$ o vetor *incidência* de B, em que $\chi(B) \in \{0, 1\}^{|V|}$ e $\chi(B)_u = 1$ se e somente se $u \in V_B$. Para todo $u \in V$, denotamos por B_u a biclique que contém apenas o vértice u e seja $\mathcal{B} = \{\chi(B_u) \in \{0, 1\}^{|L|+|R|} \mid u \in V\}$.

Dada uma instância (G, k) do kMIS, denotamos por $opt(G, k)$ o valor ótimo da função objetivo para aquela instância.

1.7 Organização do Texto

Os capítulos restantes desta dissertação estão organizados da seguinte maneira. O Capítulo 2 descreve os algoritmos exatos desenvolvidos para o kMIS. O estudo poliédrico realizado na Seção 2.4 teve como objetivo estudar desigualdades fortes para o problema. As heurísticas apresentadas no Capítulo 3 foram desenvolvidas para o problema com o intuito de fornecer bons limitantes inferiores iniciais para os modelos de PLI, visto que bons limitantes iniciais podem levar a um maior número de podas na árvore de **B**&**B**, agilizando assim a resolução de instâncias do problema. Além disso, utilizamos o valor da solução heurística para gerar o modelo de cliques descrito na Seção 2.5. Devido aos bons resultados obtidos pelas heurísticas desenvolvidas, em muitos casos a dificuldade em se resolver o problema concentra-se em provar a otimalidade de uma solução e não em encontrá-la. A formulação descrita na Seção 2.6 foi motivada com o intuito de produzir limitantes capazes de provar de maneira mais rápida a otimalidade de uma solução. Apresentamos também na Seção 3.3 dois pré-processamentos utilizados nas instâncias com o intuito de reduzir o tamanho da entrada do problema.

No Capítulo 4 são apresentados os resultados experimentais das heurísticas e dos algoritmos exatos desenvolvidos. Por fim, no Capítulo 5, apresentamos comentários sobre o trabalho realizado nesta dissertação, suas contribuições e direções de trabalho futuro.

Capítulo 2

Métodos Exatos

Neste capítulo, apresentamos as formulações de PLI desenvolvidas para o kMIS. Inicialmente, apresentamos nas Seções 2.1 e 2.2 duas formulações baseadas em PLI para o problema MEB descritas em [Acuña et al.(2014)Acuña, Ferreira, Freire, and Moreno], denominadas (MEB V) e (MEB L), e mostramos como é possível adaptá-las para o kMIS. Na Seção 2.3 introduzimos uma nova formulação de PLI para o kMIS. A Seção 2.4 é dedicada à realização de um estudo poliédrico do problema, de modo que apresentamos uma formulação ligeiramente diferente e descrevemos uma classe de desigualdades que define faceta para este politopo. Por último, introduzimos na Seção 2.6 uma nova formulação iterativa baseada em PLI para o kMIS.

2.1 Formulação *(MEB V)* adaptada para o kMIS

Para a formulação *(MEB V)*, as seguintes variáveis são definidas:

y_u = quantidade de vértices vizinhos de u que pertencem à solução, caso u também pertença à solução, e 0 caso contrário, sendo que $u \in L$.

$z_u = \begin{cases} 1, \text{ caso o vértice } u \text{ seja escolhido para estar na solução, sendo que } u \in V \\ 0, \text{ c.c} \end{cases}$

A seguir, apresentamos a formulação *(MEB V)*.

$$
\begin{aligned}
\text{Maximize} \quad & \frac{1}{k}\sum_{u\in L} y_u & & \\
\text{sujeito a} \quad & z_u + z_v \leq 1, & \forall u \in L \text{ e } \forall v \in \overline{N}(u) & \quad (2.1)\\
& y_u \leq |N(u)| z_u, & \forall u \in L & \quad (2.2)\\
& y_u \leq \sum_{v\in N(u)} z_v, & \forall u \in L & \quad (2.3)\\
& \sum_{u\in L} z_u = k & & \quad (2.4)\\
& y_u \geq 0, & \forall u \in L & \quad (2.5)\\
& z_u \in \{0,1\}, & \forall u \in V & \quad (2.6)
\end{aligned}
$$

A restrição (2.1) assegura que caso um vértice $u \in L$ seja escolhido, nenhum vértice de $v \in R$ que não seja vizinho de u pode ser selecionado e vice-versa. A restrição (2.2) força que a quantidade máxima de vértices vizinhos de $u \in L$ pertencentes a solução seja $|N(u)|$, caso u tenha sido selecionado, e 0 caso contrário. A restrição (2.3) previne que y_u seja maior que a quantidade de vértices vizinhos de u que foram escolhidos. A restrição (2.4) foi adicionada à formulação original e assegura que k vértices em L sejam escolhidos para a solução; em outras palavras, a quantidade de subconjuntos escolhidos no problema kMIS é igual a k, em concordância com a especificação deste problema. As restrições (2.6) são restrições de integralidade. Note que a integralidade das variáveis z associada à forma da função objetivo e à restrição (2.2) já garantem a integralidade das variáveis y. Para estas, apenas por completude, impomos somente a restrição de não-negatividade (2.5). A função objetivo visa maximizar a quantidade de elementos na interseção das vizinhanças dos subconjuntos escolhidos. Na formulação (MEB V), existem $O(|V|)$ variáveis e $O(|L||R|)$ restrições. A formulação (MEB L) descrita a seguir corresponde a uma modificação da formulação (MEB V) para reduzirmos a quantidade de variáveis binárias para $O(|L|)$.

2.2 Formulação *(MEB L)* adaptada para o kMIS

Para a variante da formulação *(MEB L)* utilizada para o problema kMIS, são utilizadas as mesmas variáveis descritas na formulação mostrada em 2.1, exceto que as variáveis z são definidas apenas para os vértices de L e as variáveis y apenas para os vértices em R. Note que na formulação (MEB V) as variáveis y eram definidas apenas para L.

$$
\begin{aligned}
\text{Maximize} \quad & \frac{1}{k}\sum_{v\in R} y_v & & \\
\text{sujeito a} \quad & y_v \leq \sum_{u\in N(v)} z_u, & \forall v \in R & (2.7)\\
& y_v \leq (1 - z_u)|N(v)|, & \forall u \in L, \forall v \in \overline{N}(u) & (2.8)\\
& \sum_{u\in L} z_u = k & & (2.9)\\
& y_v \geq 0, & \forall v \in R & (2.10)\\
& z_u \in \{0, 1\}, & \forall u \in L & (2.11)
\end{aligned}
$$

A restrição (2.7) força que y_v não seja maior que a quantidade de vértices vizinhos de v que foram escolhidos. A restrição (2.8) assegura que, caso um vértice $u \in L$ seja escolhido, nenhum vértice $v \in R$ que não seja vizinho de u possa ser escolhido. A restrição (2.9) corresponde à mesma restrição da formulação da Seção 2.1, forçando que k vértices em L sejam escolhidos para a solução, satisfazendo assim a restrição para o problema kMIS em que k subconjuntos devem ser selecionados. As restrições (2.11) são restrições de integralidade. Note novamente que a integralidade das variáveis z já garantem a integralidade das variáveis y, de modo que apenas por completude, impomos somente a restrição de não-negatividade (2.10).

2.3 Formulação M_{ARESTA}

Nesta seção apresentamos uma nova formulação para o kMIS. As variáveis utilizadas na formulação estão definidas abaixo:

$$x_i = \begin{cases} 1, \text{ caso o vértice } i \text{ seja escolhido para estar na solução, sendo que } i \in R \\ 0, \text{ c.c} \end{cases}$$

$$y_j = \begin{cases} 1, \text{ caso o vértice } j \text{ seja um dos } k \text{ subconjuntos da solução, sendo que } j \in L \\ 0, \text{ c.c} \end{cases}$$

Com isto, o modelo fica sendo:

$$
\begin{aligned}
& & \text{Maximize} \quad & \sum_{i \in R} x_i & & \\
& \text{(IP1)} & \text{sujeito a} \quad & \sum_{j \in L} y_j = k & & (2.12) \\
& & & x_i + y_j \leq 1, & \forall j \in L, \forall i \in \overline{N}(j) & (2.13) \\
& & & x_i \in \{0,1\}, & \forall i \in R & (2.14) \\
& & & y_j \in \{0,1\}, & \forall j \in L & (2.15)
\end{aligned}
$$

A restrição (2.12) força que k subconjuntos sejam escolhidos. A restrição (2.13) assegura que e_i não pode estar na interseção caso S_j tenha sido escolhido quando $e_i \notin S_j$. As restrições (2.14) e (2.15) forçam integralidade das variáveis. A função objetivo visa maximizar a quantidade de elementos selecionados.

2.4 Estudo Poliédrico do Politopo do kMIS

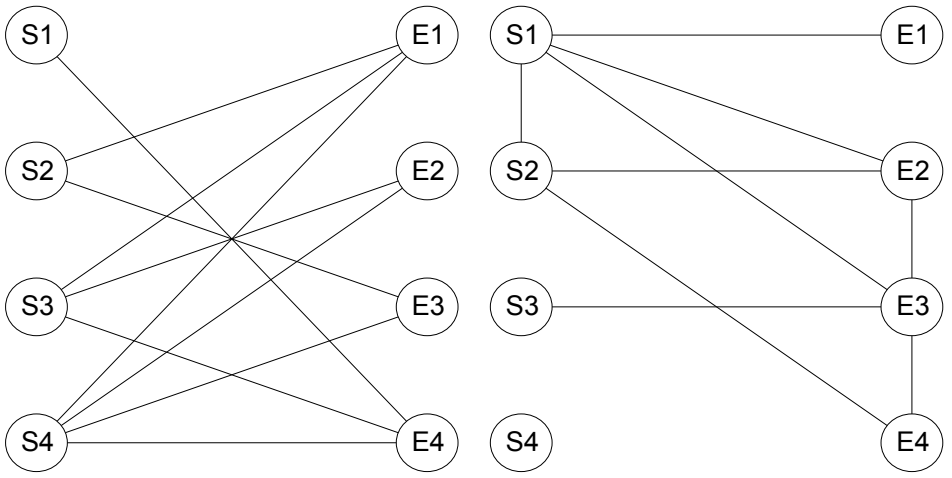
Nesta seção, tomando como ponto de partida a formulação (IP1) da seção anterior e supondo conhecido um limitante primal, derivamos um novo modelo PLI para o kMIS. O poliedro correspondente à envoltória convexa deste novo modelo é monótono. Como mencionado anteriormente, esta característica é conveniente para o estudo facial do poliedro, tornando mais simples as demonstrações dos resultados. Com isto, nesta seção, fazemos esta investigação na qual, além de mostrar que algumas restrições do modelo monótono definem facetas, também obtivemos uma nova classe de facetas que terminaram por nos conduzir à uma formulação ainda mais forte para o kMIS.

2.4.1 Problema kMIS monótono

Face à maior dificuldade em se provar facetas quando um poliedro Q definido pela envoltória convexa do problema de PLI não tem dimensão cheia, é prática usual optar por estudar a qualidade das desigualdades para um poliedro monótono $Q' \supset Q$, no qual toda solução ótima de Q é também uma solução ótima de Q', sendo este último de dimensão cheia. Assim, seja $\mathcal{P}_{\text{IP1}}(G,k,\lambda) = conv\{x \in \mathbb{R}^{|V|} | x \text{ satisfaz } (2.12), (2.13), (2.14) \text{ e } (2.15)\}$. Devido à restrição de igualdade (2.12), o politopo associado a $\mathcal{P}_{\text{IP1}}(G,k,\lambda)$ não possui dimensão cheia. Como resultado, por conveniência técnica, optamos por fazer a investigação facial do politopo associado à formulação monótona do problema kMIS, o qual pode ser descrito simplesmente substituindo-se a igualdade em (2.12) por uma restrição de forma "$\leq$". Contudo, para tornar o texto mais consistente com a implementação com-

putacional que foi feita nesta pesquisa e que será apresentada mais adiante, esta discussão será realizada considerando a existência de um limitante primal conhecido.

Assim, suponha que temos um limitante inferior λ para $opt(G, k)$. Seja $\mathcal{L}(\lambda)$ o conjunto de pares de vértices de L que possuem no máximo $\lambda - 1$ vizinhos em comum, ou seja, $\mathcal{L}(\lambda) = \{\{u, v\} \subset L \ : \ |N(u) \cap N(v)| < \lambda\}$. Analogamente, para um inteiro t, nós definimos $\mathcal{R}(t) = \{\{u, v\} \subset R \ : \ |N(u) \cap N(v)| < t\}$, isto é, como o conjunto de pares de vértices de R cuja interseção da vizinhança é menor do que t vértices. Seja $\mathcal{F}(t, \lambda) = \mathcal{L}(\lambda) \cup \mathcal{R}(t) \cup \overline{E}$. Note que, por construção, qualquer um dos pares em $\mathcal{F}(k, \lambda)$ não pode fazer parte de qualquer solução ótima do kMIS. Seja, então, G' um grafo de forma que todo par de vértices contido em $\mathcal{F}(k, \lambda)$ possui uma aresta entre si. As Figuras 2.1(a) e 2.1(b) mostram um exemplo de transformação do grafo G de uma instância do kMIS quando $k = 2$ e $\lambda = 1$ para o grafo G'.



(a) Exemplo do problema kMIS

(b) Grafo G' construído através do exemplo da Figura 2.1(a)

Figura 2.1: Exemplo de transformação de uma instância do kMIS para o grafo G'.

Introduzimos a seguir uma formulação monótona para uma variante ligeiramente diferente do problema kMIS, denotada por kMIS(λ), em que consideramos apenas soluções viáveis que não contêm pares em $\mathcal{F}(k, \lambda)$. Como todas as soluções ótimas são preservadas, a formulação (IP2) pode ser usada para encontrar uma solução ótima para o kMIS. As variáveis utilizadas nesta formulação são as mesmas variáveis descritas para a formulação $\mathrm{M}_{\mathrm{ARESTA}}$.

$$\text{Função Objetivo} \quad \max \sum_{i \in R} x_i + (|R|+1) \sum_{j \in L} y_j$$

$$\text{(IP2)} \qquad \text{sujeito a} \quad \sum_{j \in L} y_j \leq k \tag{2.16}$$

$$\sum_{j=\{u,v\} \cap L} y_j + \sum_{i=\{u,v\} \cap R} x_i \leq 1, \qquad \forall \{u,v\} \in \mathcal{F}(k,\lambda) \tag{2.17}$$

$$x_i \in \{0,1\}, \qquad \forall i \in R \tag{2.18}$$

$$y_j \in \{0,1\}, \qquad \forall j \in L \tag{2.19}$$

A restrição (2.16) força que no máximo k subconjuntos sejam escolhidos. A restrição (2.17) assegura que apenas soluções que não contêm pares em $\mathcal{F}(k,\lambda)$ sejam escolhidas. As restrições (2.18) e (2.19) são restrições de integralidade. Com relação à função objetivo, temos que cada subconjunto que pertence a uma solução ótima gera um aumento em $|R|+1$ unidades em seu valor. Como no máximo $|R|$ elementos podem fazer parte da solução ótima, temos que sempre é mais vantajoso adicionar um subconjunto ao invés de inserir elementos na solução. Uma vez que o número máximo de subconjuntos que podemos adicionar é igual a k, toda solução ótima conterá uma biclique de k subconjuntos que maximize o número de elementos nesta biclique, exatamente como requerido na solução do problema da formulação original. Seja $\mathcal{P}_{\text{IP2}}(G,k,\lambda) = \text{conv}\{x \in \mathbb{R}^{|V|} \mid x \text{ satisfaz } (2.16), (2.17), (2.18) \text{ e } (2.19)\}$. O Teorema 2.1 demonstra que o poliedro monótono $\mathcal{P}_{\text{IP2}}(G,k,\lambda)$ possui dimensão cheia. Porém, antes de enunciá-lo é interessante observar os fatos relatados no próximo parágrafo.

Seja $G' = (V', E')$ o grafo obtido por $V' = V$ e $E' = \mathcal{F}(k,\lambda)$. Note que toda solução viável do kMIS corresponde a um conjunto independente ou estável em G', isto é, um subconjunto de vértices não adjacentes dois a dois. Porém, o contrário não é verdadeiro, pois os conjuntos estáveis em G' com mais de k vértices em L não correspondem a uma solução viável do kMIS. Deste modo, uma pergunta natural seria: quais propriedades do politopo do conjunto independente correspondente a G' valem para $\mathcal{P}_{\text{IP2}}(G,k,\lambda)$? Nós exibimos uma resposta parcial para esta pergunta mostrando que algumas desigualdades, como as desigualdades de cliques e de não-negatividade, que são conhecidas por representarem facetas no politopo do conjunto independente, também descrevem facetas para $\mathcal{P}_{\text{IP2}}(G,k,\lambda)$. Para uma pesquisa mais aprofundada nos algoritmos de **B**&**C** para o problema do Conjunto Independente Máximo, como também do estudo poliédrico do chamado politopo do conjunto independente, nos referimos ao trabalho de [Rebennack et al.(2012)Rebennack, Reinelt, and Pardalos].

Teorema 2.1. *$\mathcal{P}_{\text{IP2}}(G,k,\lambda)$ possui dimensão cheia.*

Prova 2.1. *Note que, para todo $u \in V$, temos que $\chi(B_u) \in \mathcal{P}_{\text{IP2}}(G,k,\lambda)$. Além disso, o vetor nulo também pertence a $\mathcal{P}_{\text{IP2}}(G,k,\lambda)$. Estes $|V|+1$ vetores são afim-independentes e, como $\mathcal{P}_{\text{IP2}}(G,k,\lambda) \subset \mathbb{R}^{|V|}$, temos que $\mathcal{P}_{\text{IP2}}(G,k,\lambda)$ possui dimensão cheia.* □

Agora, suponha que (x_1,y_1), (x_1,y_2) e (y_1,y_2) sejam três pares em $\mathcal{F}(k,\lambda)$. Neste exemplo, podemos observar que a desigualdade $x_1+y_1+y_2 \leq 1$ é válida para $\mathcal{P}_{\text{IP2}}(G,k,\lambda)$, visto que nenhum par das três variáveis pode estar simultaneamente com valor um em uma solução viável. Com isso, podemos concluir que a desigualdade (2.17) não define faceta para $\mathcal{P}_{\text{IP2}}(G,k,\lambda)$, visto que ela é dominada pela desigualdade $x_1+y_1+y_2 \leq 1$. Para verificar este último fato, basta ver que somando-se as desigualdades (2.17) associadas aos pares (x_1,y_1), (x_1,y_2) e (y_1,y_2) chega-se à desigualdade válida $x_1+y_1+y_2 \leq \frac{3}{2}$ que, pela Definição 1.9, é dominada por $x_1+y_1+y_2 \leq 1$.

De forma genérica, podemos observar que toda clique $C \in G'$ corresponde a uma desigualdade válida para a envoltória convexa das soluções inteiras de (IP2). Quando a clique C é maximal, a desigualdade 2.20 define faceta para $\mathcal{P}_{\text{IP2}}(G,k,\lambda)$, como será demonstrado no Teorema 2.2:

$$\sum_{u \in C \cap L} y_u + \sum_{v \in C \cap R} x_v \leq 1, \text{para toda clique } C \in G'. \tag{2.20}$$

No exemplo da Figura 2.1(b), tomando-se $k=2$ e $\lambda=1$, temos as seguintes desigualdades de cliques maximais:

$$\begin{gathered} y_1 + x_1 \leq 1 \\ y_1 + y_2 + x_2 \leq 1 \\ y_1 + x_2 + x_3 \leq 1 \\ y_2 + x_4 \leq 1 \\ x_3 + x_4 \leq 1 \\ y_3 + x_3 \leq 1 \end{gathered}$$

Teorema 2.2. *Para toda clique maximal C em G', temos que a desigualdade $\sum_{u \in C \cap L} y_u + \sum_{v \in C \cap R} x_v \leq 1$ define faceta para $\mathcal{P}_{\text{IP2}}(G,k,\lambda)$, sempre que $k \geq 2$.*

Prova 2.2. *Seja C uma clique maximal em G' e $F = \{x \in \mathcal{P}_{\text{IP2}}(G,k,\lambda) \mid \sum_{u \in C \cap L} y_u + \sum_{v \in C \cap R} x_v = 1\}$. Note que, para todo $u \in C$, temos que $\chi(B_u) \in F$. Se $C = V$ então a prova está completa. Suponha então que $C \neq V$. Como C é maximal, dado um vértice $u \in V \setminus C$, existe ao menos um vértice $\phi(u)$ em C, tal que $\{u,\phi(u)\} \notin \mathcal{F}(k,\lambda)$, caso contrário, a clique não seria maximal. Para todo $u \in V \setminus C$, seja $\check{B}_u$ o subgrafo de G induzido por $\{u,\phi(u)\}$ e seja $\check{\mathcal{B}} = \{\chi(\check{B}_u) \in \{0,1\}^{|V|} \mid u \in V \setminus C\}$. Analogamente, para todo $u \in C$, seja $B_u = \{u\}$ e seja $\mathcal{B} = \{\chi(B_u) \in \{0,1\}^{|V|} \mid u \in C\}$. Note que todos*

vetores de $\mathcal{B}$ e $\check{\mathcal{B}}$ estão em F. Com isto, a matriz quadrada M composta pelos vetores $\mathcal{B} \cup \check{\mathcal{B}}$ possui inversa e estas podem ser escritas como:

$$M = \begin{pmatrix} I_{n\times n} & A \\ \mathbf{0} & I_{m\times m} \end{pmatrix} \text{ e } M^{-1} = \begin{pmatrix} I_{n\times n} & -A \\ \mathbf{0} & I_{m\times m} \end{pmatrix},$$

onde $n = |C|$ e $m = |V| - |C|$. Como $posto(M) = |V|$ então temos que os vetores em $\check{\mathcal{B}} \cup \mathcal{B}$ são afim independentes. Logo, $dim(F) = |V| - 1$ e F é uma faceta de $\mathcal{P}_{\text{IP2}}(G, k, \lambda)$. □

Nas discussões que se seguem, ao nos referimos à propriedade de uma desigualdade ser faceta, estaremos nos referindo ao poliedro monótono, a menos que seja explicitado o contrário.

2.4.2 Outras desigualdades que definem facetas

Teorema 2.3. *A desigualdade trivial $x_i \geq 0$ define uma faceta em $\mathcal{P}_{\text{IP2}}(G, k, \lambda)$.*

Prova 2.3. *Seja u um vértice em V e seja $F = \{x \in \mathcal{P}_{\text{IP2}}(G, k, \lambda) | x_u = 0\}$. Note que, para todo $v \in V \setminus \{u\}$, temos que $\chi(B_v) \in F$, exibindo assim $|V| - 1$ pontos afim-independentes. Como o vetor nulo está em F, temos então $|V|$ pontos afim-independentes na face e concluímos que F define faceta em $\mathcal{P}_{\text{IP2}}(G, k, \lambda)$.* □

Teorema 2.4. *A desigualdade trivial $y_i \geq 0$ define uma faceta para o problema monótono.*

Prova 2.4. *Seja u um vértice em L e seja $F = \{x \in \mathcal{P}_{\text{IP2}}(G, k, \lambda) | y_u = 0\}$. Note que, para todo $v \in V \setminus \{u\}$, temos que $\chi(B_v) \in F$, exibindo assim $|V| - 1$ pontos afim-independentes. Como o vetor nulo está em F, temos então $|V|$ pontos afim-independentes na face e concluímos que F define faceta em $\mathcal{P}_{\text{IP2}}(G, k, \lambda)$.* □

2.5 Formulações de PLI com Cliques ($\mathbf{M_{CLIQUE}}$)

Utilizando a desigualdade de clique descrita em (2.20), temos então a seguinte formulação:

$$\begin{aligned}
\text{Função Objetivo} \quad & \max \sum_{i \in R} x_i & & \\
\text{sujeito a} \quad & \sum_{j \in L} y_j = k & & (2.21) \\
& \sum_{u \in C \cap L} y_u + \sum_{v \in C \cap R} x_v \leq 1, \text{ para toda clique } C \in G' & & (2.22) \\
& x_i \in \{0, 1\}, & \forall i \in R & (2.23) \\
& y_j \in \{0, 1\}, & \forall j \in L & (2.24)
\end{aligned}$$

Devido ao número exponencial de restrições representadas pela desigualdade (2.22), uma possível alternativa para se chegar a um modelo compacto para o problema é gerarmos cliques para a formulação inicial de forma que toda aresta do grafo G' esteja contida em ao menos uma clique. Isto é válido pois como toda aresta do grafo G' está contida em uma clique, temos então que toda desigualdade no formato $x_i + y_j \leq 1$, para todo $j \in L$ e $i \in \overline{N(j)}$, é dominada pela restrição $\sum_{u \in C \cap L} y_u + \sum_{v \in C \cap R} x_v \leq 1$, para alguma clique $C \in G'$. Com isto, podemos aplicar um algoritmo de **B**&**C**, como descrito na Seção 2.5.1, para acrescentarmos, em tempo de execução, as novas restrições de clique que vão sendo violadas pela solução ótima das relaxações lineares.

No nosso caso, utilizamos o Algoritmo 1 descrito por [Nemhauser and Sigismondi(1992)] para gerar o conjunto de cliques iniciais da formulação.

Algoritmo 1: Método CLQ

```
1  início
2      Seja CLIQUE um conjunto vazio.
3      para cada cada aresta {i, j} no grafo faça
4          C ← ∅.
5          se a aresta {i, j} ∉ CLIQUE então
6              C ← C ∪ {ij}.
7              para todo vértice v ∈ V \ {ij} faça
8                  se v for adjacente a todo vértice em C então
9                      C ← C ∪ {v}
10             CLIQUE ← CLIQUE ∪ {C}
11     retorna CLIQUE
```

2.5.1 Método de Branch-and-Cut

Dada uma solução fracionária x' da relaxação linear, encontrar uma desigualdade de clique violada por esta solução equivale a encontrar uma clique de peso máximo no grafo, onde o custo de cada nó corresponde ao valor da variável correspondente. Visto que encontrar uma clique de peso máximo é em geral um problema $\mathcal{NP}$-difícil, em [Nemhauser and Sigismondi(1992)] é proposta uma heurística gulosa para esta tarefa. A heurística procura encontrar uma clique cuja solução da relaxação linear do modelo M_{ARESTA} esteja violando alguma das desigualdades de clique, ou seja, a soma das variáveis dos vértices da clique é maior que 1.

A heurística foi utilizada como parte de um algoritmo **B**&**C**, verificando-se em cada nó da árvore de **B**&**B** se existe alguma desigualdade violada. Caso alguma desigualdade seja violada, a desigualdade é adicionada ao modelo e o nó é executado novamente. Dado um grafo $G' = (V, E)$, escolhendo um nó inicial v, o algoritmo remove de G' todos os

vértices que não são vizinhos de v e marca o vértice v como utilizado. Este processo é repetido recursivamente até que todos os vértices restantes em G' tenham sido marcados. A seleção do nó v inicial é realizada através de dois critérios:

1. Escolher a variável x'_v, associada a v, que minimize $\{|x'_v - 1/2| : 0 < x'_v < 1\}$.
2. Escolher a variável x'_v, associada a v, que maximize $\{x'_v : x'_v < 1\}$.

O critério (2) é aplicado apenas quando (1) não consegue encontrar uma desigualdade de clique violada.

2.6 O Problema kMIS Inverso

Nesta seção, estudamos o problema "inverso" ao kMIS, denominado de zMIE, onde o objetivo é encontrar a maior quantidade de subconjuntos cuja a interseção contenha ao menos z elementos.

Isto foi feito porque a resolução do zMIE permite que sejam calculados limitantes duais para o kMIS. O motivo para tal é que, se a maior quantidade de subconjuntos cuja interseção é composta de ao menos z elementos for menor do que k, isto implica que não existe uma solução para o kMIS de valor z. Caso o tamanho da interseção destes subconjuntos seja maior ou igual k, podemos concluir que existe uma solução para o kMIS de valor z.

Com isto em mente, foi desenvolvido um algoritmo iterativo através do cálculo de limitantes duais via zMIE, que, eventualmente, levará a uma solução ótima para o kMIS.

A ideia do algoritmo baseia-se em, através do valor λ obtido por uma solução heurística para o kMIS, fixarmos inicialmente $z = \lambda+1$ e resolvermos o problema zMIE, com o intuito de verificar qual o número máximo de subconjuntos para o qual a interseção tem tamanho não inferior a este valor z. Este valor de z é incrementado em uma unidade até que a maior interseção de z elementos seja menor do que k, de forma que podemos concluir que não existem k subconjuntos cuja interseção seja composta de z elementos, levando assim a uma solução ótima para o kMIS de valor $z - 1$.

A seguir, apresentamos a formulação PLI desenvolvida para o problema zMIE.

$$\begin{aligned}
\text{Maximize} \quad & \sum_{i \in L} y_i \\
\text{sujeito a} \quad & \sum_{j \in R} x_j \geq z && (2.25) \\
& \sum_{u \in C \cap L} y_u + \sum_{v \in C \cap R} x_v \leq 1, \text{ para toda clique } C \in G' && (2.26) \\
& x_i \in \{0, 1\}, \qquad \forall i \in R && (2.27) \\
& y_j \in \{0, 1\}, \qquad \forall j \in L && (2.28)
\end{aligned}$$

As seguintes etapas descrevem a execução do algoritmo, referido como M$_{\text{ITERATIVO}}$:

1. Se durante a execução do modelo PLI para o problema zMIE o melhor limitante dual for menor que k, então podemos encerrar sua execução, visto que é impossível encontrar uma solução para o kMIS que possua k subconjuntos e cuja interseção contenha z elementos. Assim, temos que a solução ótima para o kMIS é de $z - 1$ elementos.

2. Se durante a execução do modelo PLI para o problema zMIE encontramos uma solução primal viável com ao menos k subconjuntos, podemos concluir então que existe uma solução que contém k subconjuntos e no mínimo z elementos para o kMIS. Devido a isso, executamos então novamente o modelo PLI de modo a encontrar a maior quantidade possível de subconjuntos cuja interseção tenha pelo menos $z \leftarrow z + 1$ elementos. Repetir a regra 2 até a regra 1 ser satisfeita.

Capítulo 3

Métodos Heurísticos

Como mencionado na Seção 2.4, é necessário obter o valor de uma solução primal para gerarmos desigualdades de cliques para o modelo de PLI. Além disso, bons limitantes inferiores iniciais podem gerar um maior número de podas na árvore de **B&B**, facilitando assim a resolução de instâncias do problema. Neste sentido, descrevemos nas Seções 3.1 e 3.2 duas heurísticas desenvolvidas para o kMIS.

3.1 Heurística Gulosa

Uma estratégia simples e que obtêm bons resultados para uma variedade de problemas de otimização combinatória são os denominados algoritmos gulosos. Vários problemas clássicos da literatura como o algoritmo de Kruskal [Kruskal(1956)] e de Prim [Prim(1957)] para encontrar a árvore geradora mínima, o algoritmo de Dijkstra [Dijkstra(1959)] para encontrar o caminho mínimo entre dois vértices de um grafo, e o algoritmo de Huffman [Huffman(1952)] que consiste em um método de compressão de símbolos, são resolvidos de maneira exata através de algoritmos gulosos de complexidade polinomial.

O algoritmo guloso é um método iterativo que, a cada iteração, mantém uma lista de candidatos C que podem ser adicionados à solução parcial S' do problema obtida na iteração anterior. A partir desta lista, o algoritmo faz uso de uma função $c(e)$ que mede o custo incremental de adicionarmos o elemento $e \in C$ à solução parcial. Para problemas de maximização, o elemento e' tal que $c^{max} = \max_{e \in C \setminus S'} c(e)$ é inserido na solução parcial do problema. O algoritmo encerra sua execução quando uma solução viável é obtida.

Para o kMIS, seja L' o conjunto composto pelos subconjuntos que compõem a solução parcial do problema. A ideia do algoritmo guloso desenvolvido consiste em, a cada iteração, incluir na solução um subconjunto $s \in L \setminus L'$, tal que $(\bigcap_{e \in L'} e) \cap N(s)$ seja máxima. Este procedimento é realizado até que k subconjuntos estejam na solução, como apresentado no Algoritmo 2.

Algoritmo 2: Heurística Gulosa para o kMIS.

Data: **Grafo G (L,R)-bipartido**, **inteiro** k
Result: Limitante primal para o kMIS

```
1  início
2      Seja E ← R.
3      L' ← ∅.
4      enquanto |L'| ≤ k faça
5          Selecionar um subconjunto de s ∈ L \ L' de maneira que (⋂_{e∈L'} e) ∩ N(s)
           seja máxima.
6          E ← E ∩ N(s).
7          L' ← L' + s.
8      fim
9      return L'
10 fim
```

No Algoritmo 2, temos inicialmente que a solução parcial é composta de todos os R elementos e nenhum subconjunto (linhas 2 e 3). O laço da linha 4 é executado k vezes, de modo que a cada iteração, um subconjunto $s \in L \setminus L'$ tal que $(\bigcap_{e \in L'} e) \cap N(s)$ seja máxima é escolhido para fazer parte da solução (linha 5) e em seguida os elementos e subconjuntos que fazem parte da solução parcial são atualizados devido a inserção de s na solução (linhas 6 e 7). Por fim, a linha 9 retorna os k subconjuntos que fazem parte da solução. Em relação a complexidade computacional, observa-se que no corpo do laço da linha 4 é necessário calcular a interseção da solução parcial com todos os $|L'|$ subconjuntos restantes, o que leva um tempo de $O(|L||R|)$. Dado que o laço itera k vezes, temos que a complexidade assintótica do algoritmo é dada por $O(k|L||R|)$.

3.2 GRASP

Um método heurístico mais sofisticado que a heurística gulosa pode ser obtido por meio de uma meta-heurística *GRASP* [Feo and Resende(1989)]. O *GRASP* também é um algoritmo iterativo no qual cada iteração é composta de duas etapas: construção e busca local. A fase de construção é responsável por encontrar uma solução viável para o problema, enquanto a fase de busca local é composta por uma busca na vizinhança da solução encontrada pela fase da construção, até que um máximo local seja encontrado (considerando um problema de maximização). O Algoritmo 3 descreve o funcionamento do *GRASP* básico, acrescido de uma estratégia de intensificação, chamada de *Path-Relinking*. Nesta última, explora-se o espaço de soluções em busca de um caminho que conecte duas soluções previ-

amente fixadas, com o intuito de encontrar soluções intermediárias boas e que ainda não foram exploradas. No Algoritmo 3, temos que na linha 5 é gerada uma solução viável para o problema. Na linha 6, é feita uma busca na vizinhança da solução gerada até que seu máximo local seja encontrado e na linha 7 o procedimento de *Path-Relinking* é executado. A linha 8 é responsável por verificar qual é a melhor solução (no sentido de custo) entre a solução gerada e a melhor solução e salvá-la. O processo é repetido por $Iteracoes$ vezes e o algoritmo retorna a melhor solução na linha 10.

Algoritmo 3: Meta-heurística GRASP

```
1  início
2      i ← 1
3      MelhorSolucao ← ∅.
4      enquanto i ≤ Iteracoes faça
5          Solucao ← Construcao()
6          Solucao ← Busca_Local(Solucao)
7          Solucao ← Path − Relinking(Solucao)
8          Salvar_Solucao(Solucao, MelhorSolucao)
9          i ← i + 1
10     retorna MelhorSolucao
```

Em cada iteração da fase de construção uma lista restrita de candidatos (LRC), composta pelos elementos $e \in C$ com maior custo incremental $c(e)$ e que não fazem parte da solução parcial do problema, é construída, de modo que o elemento a ser incorporado à solução parcial C' é escolhido de forma aleatória entre os elementos da LRC. O tamanho da LRC é um fator relevante para conseguir um balanço entre a qualidade da solução e o tempo de execução do $GRASP$. O tamanho da lista é associada com um parâmetro $\alpha \in [0, 1]$, que indica o quão aleatório ou guloso o algoritmo será, sendo formada pelos elementos e cujo custo incremental $c(e) \in [c^{min} + \alpha \times (c^{max} - c^{min}), c^{max}]$, em que $c^{max} = \max_{e \in C \setminus C'} c(e)$ e $c^{min} = \min_{e \in C \setminus C'} c(e)$ representam o valor do maior e do menor custo incremental, respectivamente, no caso de problemas de maximização.

Em [Prais and Ribeiro(2000)] é mostrado que utilizar um único parâmetro α muitas vezes dificulta a busca por uma solução de boa qualidade. Um dos métodos mais empregados para a escolha do parâmetro α com o intuito de obter boas soluções é o chamado $GRASP$ Reativo [Prais et al.(1998)Prais, Ribeiro, Celso, and Ribeiro], em que a cada iteração da fase de construção um possível valor α é escolhido a partir de um conjunto de possíveis valores. Seja $X = \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ o conjunto de possíveis valores de α. Inicialmente, a probabilidade de que cada valor de α seja escolhido é de $p_i = \frac{1}{|X|}$. A ideia do $GRASP$ Reativo é atualizar as probabilidades conforme as iterações são realizadas, de maneira a favorecer os valores de α que levarem a melhores soluções. Seja z^* a melhor solução encontrada até o momento e seja A_i a média

dos valores da solução quando x_i é escolhido. As probabilidades de cada valor $x_i \in X$ são atualizadas a cada iteração conforme a equação:

$$p_i = \frac{q_i}{\sum_{j=1}^{m} q_j}, \text{ onde } q_i = \frac{z^*}{A_i}. \tag{3.1}$$

Para o kMIS, a ideia da fase de construção consiste em a cada iteração desta, adicionar à solução o subconjunto s escolhido aleatoriamente a partir da LRC. O processo é repetido até que k subconjuntos façam parte da solução, finalizando assim a fase de construção. O Algoritmo 4 descreve o funcionamento da fase de construção com a utilização do $GRASP$ Reativo para o kMIS. Com relação a complexidade computacional da fase de construção, temos que a cada uma das k iterações do laço da linha 3 (a solução não está completa até que k subconjuntos façam parte dela), para a construção da LRC, é necessário realizar a interseção da solução parcial com todos os subconjuntos restantes, ou seja, os subconjuntos que não fazem parte da solução, a fim de obter os custos incrementais de cada subconjunto a solução parcial, levando assim um tempo de $O(|L||R|)$. Além disso, dado que a lista LRC é composta apenas dos elementos com maior custo incremental, é necessário que todos os custos incrementais sejam ordenados, o que custa $O(|L|log|L|)$. Como o laço é repetido k vezes, temos então que a complexidade do procedimento é $O(k|L|(log|L| + |R|))$.

Algoritmo 4: Fase de Construção

```
1  início
2      Solucao ← ∅.
3      enquanto a solucao nao estiver completa faça
4          Selecionar um valor de α.
5          Construir a LRC com base no valor de α.
6          Selecionar aleatoriamente um subconjunto s da LRC.
7          Solucao ← Solucao + s
8          Atualizar e reavaliar o custo incremental dos candidatos remancescentes.
9          para todo valor de x_i ∈ X faça
10             Atualizar a probabilidade de x_i ser selecionado.
11     retorna Solucao
```

A fase de busca local é composta em uma busca na vizinhança da solução encontrada pela fase de construção, até que um máximo local seja encontrado. Seja S a solução obtida pela fase de construção e $\overline{S}$ o conjunto de todos os subconjuntos que não fazem parte da solução, ou seja, $\overline{S} = L \setminus S$. A vizinhança da solução S é obtida através da remoção de um subconjunto $s \in S$ e da adição de um subconjunto $s' \in \overline{S}$ à solução S. Assim, a cada iteração o procedimento visa encontrar uma solução S' vizinha a S tal que $\bigcap(S') > \bigcap(S)$. O procedimento encerra sua execução quando não existir uma solução S'

vizinha a S que satisfaça essa propriedade. O Algoritmo 5 descreve o funcionamento da fase de busca local para o kMIS.

Existem alguns critérios para a escolha do vizinho S' de S na fase de busca local, como por exemplo a escolha do melhor vizinho (em relação ao custo da solução), de um vizinho escolhido aleatoriamente ou do primeiro vizinho encontrado S' de S tal que $\bigcap(S') > \bigcap(S)$. Devido ao maior tempo gasto para se encontrar o melhor vizinho, optamos por utilizar a abordagem do primeiro vizinho encontrado.

Algoritmo 5: Fase de Busca Local

```
1 início
2     Seja S a solução obtida pela fase de construção.
3     enquanto existir um vizinho S' de S tal que ⋂(S') > ⋂(S) faça
4         S ← S'
5     retorna S
```

Uma estratégia geralmente utilizada em conjunto com a meta-heurística $GRASP$ é o chamado *Path-Relinking*. O *Path-Relinking* foi originalmente proposto em [Glover(1996)] como uma estratégia de intensificação para explorar o trajeto que conecta um par de soluções (x, y), de modo que x é uma solução ótima local obtida após a realização da busca local de toda iteração do $GRASP$ e y é uma solução elite selecionada aleatoriamente de um conjunto E de tamanho Max_Elite de soluções elites, que corresponde a um conjunto composto das melhores soluções (de maior custo) encontradas até o momento pelo algoritmo. Seja x_i a solução inicial e x_a a solução alvo, compostas pela solução obtida pela busca local e uma solução elite, respectivamente. É construído um caminho pelo espaço de soluções que conecta a solução inicial até a solução alvo, gerando assim várias soluções intermediárias a cada movimento executado. Estes movimentos são realizados de forma a inserir atributos na solução atual que estão presentes na solução alvo. No caso do kMIS, este movimento é realizado de forma a trocar um subconjunto que faz parte da solução por um que não faz. A principal ideia deste procedimento é que boas soluções possuem características a serem compartilhadas, e que dentre as soluções intermediárias geradas durante o caminho pode-se encontrar soluções boas ainda não exploradas.

Inicialmente o conjunto de soluções elites está vazio. A ideia é manter no conjunto soluções boas e diversificadas, sendo cada ótimo local obtido na fase de busca local um candidato para ser inserido se este for suficientemente diferente de todas as outras soluções atualmente no conjunto. Se o conjunto E não estiver cheio, então o candidato é simplesmente inserido no conjunto. Se o conjunto E já contiver Max_Elite elementos e o candidato a ser inserido no conjunto for melhor que o pior (no sentido do custo da solução) candidato que estiver no conjunto, então uma possível estratégia corresponde a trocar a solução candidata com a pior solução contida no subconjunto. Outra estratégia que tende

a diversificar mais o conjunto é de substituir a solução contida no conjunto mais similar com a solução candidata, ou seja, a solução com menor diferença simétrica, dentre todas as soluções do conjunto com custo pior que a solução candidata. Esta estratégia foi adotada para os testes computacionais do *GRASP*.

O procedimento de *path-relinking* é realizado após a fase de busca local, definindo-se quem será a solução inicial x_i e a solução alvo x_a entre a solução x obtida pela fase de busca local e uma solução y do conjunto E escolhida aleatoriamente.

Em [Ribeiro et al.(2001)Ribeiro, Uchoa, and Werneck] é observado que o procedimento atinge melhores soluções quando a solução inicial é melhor do que a solução alvo, pois a vizinhança da solução inicial é mais cuidadosamente explorada, o que proporciona uma investigação mais detalhada da vizinhança da região mais promissora. Após isso, é computada a diferença simétrica $\triangle(x_i, x_a)$ entre a solução x_i e x_a, que corresponderá ao conjunto de movimentos necessários para a solução x_i atingir a solução x_a. Em cada etapa, o procedimento verifica todos os movimentos $m \in \triangle(x, x_a)$ a partir da solução x atual e seleciona o movimento com maior benefício em termos de custo, ou seja, o movimento que maximiza a função $f(x \oplus m)$, no qual $x \oplus m$ representa a solução resultante da aplicação do movimento m à solução x. O melhor movimento m^* é realizado, produzindo então a solução $x \oplus m^*$. O procedimento termina quando a solução x_a é atingida, ou seja, quando $\triangle(x, x_a) = 0$. O Algoritmo 6 ilustra o funcionamento do *Path-Relinking* para o kMIS.

Algoritmo 6: Path-Relinking

```
1  início
2      f* ← max{f(x_i), f(x_a)}
3      x* ← argmax{f(x_i), f(x_a)}
4      x ← x_i
5      enquanto △(x, x_a) ≠ 0 faça
6          m* ← argmax{f(x ⊕ m) : m ∈ △(x, x_a)}
7          △(x ⊕ m*, x_a) ← △(x, x_a) \ {m*}
8          x ← x ⊕ m*
9          se f(x) > f* então
10             f* ← f(x)
11             x* ← x
12     retorna x*
```

3.3 Pré-processamento das Instâncias

Nesta seção, descrevemos os pré-processamentos realizados nas instâncias com o intuito de reduzir o tamanho da entrada. Ou seja, o objetivo do pré-processamento é reduzir o

número de vértices do grafo de entrada de maneira a permitir que se resolva a instância mais rapidamente. Assim, foi desenvolvido um pré-processamento visando remover subconjuntos e elementos das instâncias. Dado que $\mathcal{L}(\lambda) = \{\{u, v\} \subset L \ : \ |N(u) \cap N(v)| < \lambda\}$ e $\mathcal{R}(t) = \{\{u, v\} \subset R \ : \ |N(u) \cap N(v)| < t\}$, temos então os seguintes pré-processamentos:

1. Para todo vértice $u \in L$, seja $\mathcal{L}_u(\lambda) = \{v \in L \mid \{u, v\} \in \mathcal{L}(\lambda)\}$ o conjunto de vértices de L que não podem estar simultaneamente com u em qualquer solução viável de $\mathcal{P}_{\text{IP2}}(G, k, \lambda)$. Todo vértice de u em L com $|N(u)| < \lambda$ ou com $|L| - |\mathcal{L}_u(\lambda)| < k$ pode ser removido de G.

2. Para todo vértice v em R, seja $\mathcal{R}_v(k) = \{w \in R \mid \{v, w\} \in \mathcal{R}(k)\}$ o conjunto de vértices de R que não podem estar simultaneamente com v em qualquer solução viável de $\mathcal{P}_{\text{IP2}}(G, k, \lambda)$. Similarmente, todo vértice de v em R com $|N(v)| < k$ ou com $|R| - |\mathcal{R}_v(k)| < \lambda$ pode ser removido de G.

3. Repetir os passos 1 ou 2 enquanto for possível reduzir o tamanho da instância, visto que novos vértices de "grau baixo"podem aparecer no grafo resultante do pré-processamento.

Repare que o procedimento acima pode ser reaplicado toda vez que um novo limitante inferior λ' for encontrado, de modo que $\lambda' > \lambda$.

As Figuras 3.1, 3.2 e 3.3 exibem um exemplo com um grafo antes do pré-processamento, após a realização da etapa 1 e após a realização da etapa 2, respectivamente, para uma instância do problema onde $k = 2$ e a solução primal obtida por uma heurística qualquer possui valor igual a 3.

No exemplo da Figura 3.1, temos que $\mathcal{L}_{S1}(\lambda) = \{S2\}$, $\mathcal{L}_{S2}(\lambda) = \{S1, S3, S4\}$, $\mathcal{L}_{S3}(\lambda) = \{S2\}$ e $\mathcal{L}_{S4}(\lambda) = \{S2\}$. Pelo passo 1, observa-se que o subconjunto $S2$ pode ser removido, visto que $|L| - |\mathcal{L}_{S2}| = 4 - 3 < 2$ ou $|N(S2)| = 2 < 3$, resultando assim no gráfico da Figura 3.2. Para este último, podemos verificar que $\mathcal{R}_{E1}(k) = \{E5\}$, $\mathcal{R}_{E2}(k) = \{E5\}$, $\mathcal{R}_{E3}(k) = \{E5\}$, $\mathcal{R}_{E4}(k) = \{E5\}$ e $\mathcal{R}_{E5}(k) = \{E1, E2, E3, E4\}$. Através do passo 2, nota-se que o elemento $E5$ pode ser retirado, pois $|R| - |\mathcal{R}_{E5}(k)| = 5 - 4 = 1 < 3$ ou $|N(E5)| = 1 < 2$, como mostra o gráfico da Figura 3.3.

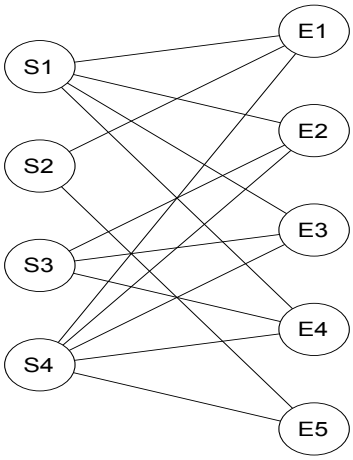


Figura 3.1: Grafo G original.

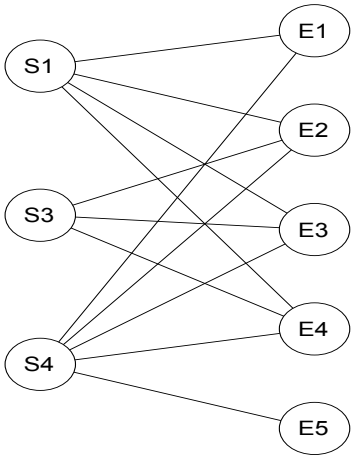


Figura 3.2: Grafo G resultante da poda de subconjuntos (etapa 1).

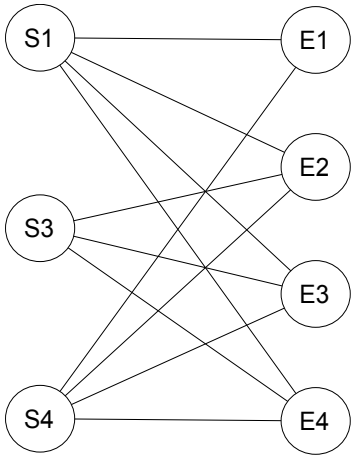


Figura 3.3: Grafo G resultante da poda de elementos (etapa 2).

Capítulo 4

Avaliação Experimental

Neste capítulo apresentamos os resultados computacionais obtidos através dos algoritmos exatos e das heurísticas desenvolvidas. O capítulo encontra-se dividido em três seções, de modo que na Seção 4.1 descrevemos os experimentos preliminares realizados para as heurísticas, na Seção 4.2 os experimentos preliminares para os algoritmos exatos e, por fim, na Seção 4.3 apresentamos os experimentos finais. Para a avaliação experimental foram utilizadas instâncias geradas aleatoriamente como também instâncias obtidas pelo *dataset* disponibilizado em `www.dtic.upf.edu/ocelma/MusicRecommendationDataset/lastfm-1K.html` (último acesso em Dezembro de 2013) por Celma em seu trabalho sobre recomendações de músicas [Celma(2008)]. O *dataset* foi obtido através da LastFM (`www.last.fm`) e é composto de 176.948 bandas, 992 ouvintes e 19.150.868 pares de ouvinte-banda (arestas) indicando o gosto musical de usuários até Maio de 2009.

Analisando o grau de distribuição dos vértices representando as bandas, observamos que este obedece uma *power law* [Newman(2005)]. Como consequência, existem poucas bandas com um número alto de ouvintes e a maioria dos vértices representando as bandas possuem grau próximo de zero.

A partir do *dataset* descrito, foram geradas 6 classes de instâncias contendo 50, 70, 90, 110, 130 e 150 bandas. As bandas fazem o papel de vértices em L, enquanto os ouvintes são associados aos vértices em R. Uma aresta existe entre um vértice representando uma banda u e um ouvinte v se v é fã da banda u. Para gerar uma instância com n bandas, selecionamos as $2n$ bandas mais ouvidas e aleatoriamente escolhemos n bandas dentre elas. O número de ouvintes e arestas para cada classe de bandas são exibidos na Tabela 4.9. Para cada classe foram geradas 10 instâncias e cada instância foi testada com $k = 2, 3, 4, 5, 6, 7$, resultando em um total de 360 instâncias. Estes valores de k foram escolhidos em vista que um show musical geralmente possui um número baixo de bandas. Além disso, como pode ser visto na Figura 4.12, independentemente do modelo, conforme o valor de k aumenta, o tempo de execução também cresce. Porém, após um intervalo, o

Grafo G_b	
L	**R**
50	50
60	60
70	70
80	80
90	90
100	100

Tabela 4.1: Instâncias do grafo G_b.

Grafo G_{de}	
L	**R**
30	60
30	90
30	120
30	150
30	180

Tabela 4.2: Instâncias do grafo G_{de}.

Grafo G_{ds}	
L	**R**
180	30
150	30
120	30
90	30
60	30

Tabela 4.3: Instâncias do grafo G_{ds}.

tempo de execução diminui conforme o valor de k aumenta.

As instâncias aleatórias foram geradas utilizando o modelo clássico de grafo proposto por Erdos e Renyi [Erdos and Renyi(1960)], em que todo grafo composto por V vértices e E arestas tem probabilidade igual de ser gerado, e através do modelo proposto por Gilbert [Gilbert(1959)], em que toda aresta ocorre independentemente, com probabilidade p, $0 < p < 1$. Foram gerados 3 tipos de instâncias, de tal modo que, no primeiro, o número de subconjuntos seja igual ao número de elementos (G_b), no segundo, o número de elementos seja maior que o número de subconjuntos (G_{de}) e, finalmente, no terceiro, o número de subconjuntos seja maior que o número de elementos (G_{ds}). As classes descritas nas Tabelas 4.1, 4.2 e 4.3 foram geradas para os grafos G_b, G_{de} e G_{ds}, respectivamente.

Para cada classe foi gerado um conjunto de 10 instâncias aleatórias, sendo metade utilizando o modelo de grafo descrito por Erdos e Renyi e metade o modelo descrito por Gilbert, totalizando 1440 instâncias, considerando os grafos G_b, G_{de} e G_{ds}. As instâncias geradas foram divididas em grupos de acordo com a densidade e a quantidade k de subconjuntos que devem ser selecionados. Os valores de k foram divididos em faixas de tal forma que ele é considerado baixo quando $0.1|L| \leq k \leq 0.3|L|$, médio quando $0.4|L| \leq k \leq 0.6|L|$ e alto quando $0.7|L| \leq k \leq 0.9|L|$. Analogamente, a densidade do grafo é considerada baixa quando $0.1|E| \leq densidade \leq 0.3|E|$, média quando $0.4|E| \leq densidade \leq 0.6|E|$ e alta quando $0.7|E| \leq densidade \leq 0.9|E|$. No total, realizando a combinação dos grupos de densidade e de valores de k, as seguintes classes de instâncias foram geradas:

A classificação das instâncias em grupos teve por objetivo a possibilidade de identificar de que forma os parâmetros afetam o desempenho dos algoritmos propostos.

Como resolvedor de PLI, utilizamos o *IBM CPLEX 12* em monoprocessamento, com todas as heurísticas e todas as rotinas de plano de corte desligadas (com excessão da rotina de Cliques, utilizada nos modelos M_{CLIQUE} e $\text{M}_{\text{ITERATIVO}}$). Inicialmente, a separação de desigualdades de cliques foi realizada através da heurística descrita na Seção 2.5.1. Contudo, experimentos preliminares mostraram resultados superiores utilizando a rotina de

Classe	Densidade	k
C_{bb}	baixa	baixo
C_{bm}	baixa	médio
C_{ba}	baixa	alto
C_{mb}	média	baixo
C_{mm}	média	médio

Tabela 4.4: Classes de instâncias geradas.

Classe	Densidade	k
C_{ma}	média	alto
C_{ab}	alta	baixo
C_{am}	alta	médio
C_{aa}	alta	alto

Tabela 4.5: Classes de instâncias geradas.

geração de planos de corte de clique disponíveis no *CPLEX*. Os códigos foram desenvolvidos na linguagem $C++$ e executados em uma máquina Intel(R) Xeon(R) @2.40GHz com 8GB de memória RAM.

Testes preliminares foram efetuados com o intuito de estabelecer parâmetros que levassem a um bom desempenho computacional tanto do $GRASP$ quanto dos algoritmos exatos desenvolvidos. Tais experimentos foram conduzidos em um conjunto de 330 instâncias, sendo 270 aleatórias e 60 geradas a partir do *dataset* da LastFM. Para cada instância foi fixado um tempo máximo de execução de 300 segundos.

4.1 Parâmetros do $GRASP$

O primeiro teste consistiu em estabelecer um limite para a quantidade de iterações realizadas pela meta-heurística $GRASP$. Para isto, as instâncias foram executadas arbitrando-se inicialmente um limite de 5000 iterações. Com base nos resultados obtidos, foi observado que, em média, o $GRASP$ atinge a melhor solução para as instâncias em aproximadamente 500 iterações. Logo, decidimos fixar o número de iterações do $GRASP$ em 500.

O segundo teste consistiu em verificar se a etapa de busca local e o *path-relinking* trazem melhorias no valor da solução obtida na fase de construção da meta-heurística. Foi constatado que em aproximadamente 10% das instâncias a busca local e o *path-relinking* encontram uma solução melhor do que aquela produzida pela fase de construção do $GRASP$. Além disso, notamos que na maioria das instâncias a solução obtida pela fase de construção só é melhorada pelas etapas de busca local e *path-relinking* caso seu valor seja de pelo menos 0.8λ, sendo λ o custo da melhor solução gerada até o momento. Logo, com o intuito de diminuir o tempo de execução da meta-heurística $GRASP$ e ainda manter a qualidade das soluções obtidas, optamos por realizar as etapas de busca local e *path-relinking* em uma iteração apenas se o custo da solução obtida pela fase de construção for de pelo menos 80% do valor da melhor solução λ alcançada pelo $GRASP$ até aquela iteração.

4.2 Parâmetros dos modelos de PLI

Como mencionado na Seção 1.3, bons limitantes inferiores iniciais podem reduzir o crescimento da árvore de enumeração, diminuindo assim o espaço de busca das soluções. Com isto em mente, o primeiro teste consistiu em verificar se a inicialização da formulação $\mathrm{M}_{\mathrm{ARESTA}}$ com o valor da solução da meta-heurística *GRASP* melhora o desempenho do modelo em relação a inicialização do modelo $\mathrm{M}_{\mathrm{ARESTA}}$ com o valor da solução heurística do algoritmo guloso. Com base nos resultados, o modelo $\mathrm{M}_{\mathrm{ARESTA}}$ inicializado com o valor da meta-heurística *GRASP* foi possível resolver de maneira exata 262 instâncias, enquanto o modelo resolveu 259 instâncias através da heurística gulosa. Além disso, em 26.36% das instâncias, o *GRASP* levou a limitantes inferiores melhores que a heurística gulosa, enquanto nas instâncias restantes ambos forneceram o mesmo limitante inferior. Com relação ao tempo de execução, não houve diferença significativa entre os dois casos. Com base nestas observações, optamos por inicializar os modelos de PLI com a meta-heurística *GRASP* desenvolvida.

Testes preliminares também foram conduzidos com o objetivo de avaliar o desempenho do *CPLEX* ao receber como entrada os modelos (MEB V), (MEB L) e $\mathrm{M}_{\mathrm{ARESTA}}$. Estes testes mostraram um desempenho superior do modelo $\mathrm{M}_{\mathrm{ARESTA}}$, que foi 34 vezes mais rápido que o modelo (MEB L) e 41 vezes mais rápido que o modelo (MEB V).

Com relação aos modelos $\mathrm{M}_{\mathrm{CLIQUE}}$ e $\mathrm{M}_{\mathrm{ARESTA}}$, foram realizados testes com o intuito de se verificar a força dos mesmos no que diz respeito aos limitantes superiores por eles produzidos. Seja LP o valor da relaxação linear de um modelo M e LB do melhor limitante inferior encontrado entre ambos os modelos. Para verificarmos a força deste modelo M, analisamos o *gap* de integralidade, calculado por $\frac{LP-LB}{LP}$, em cada caso. As Figuras 4.1, 4.2, 4.3 e 4.4 exibem o *gap* de integralidade médio e o desvio padrão das instâncias para todas as classes de instâncias aleatórias e da LastFM.

Como era de se esperar, através das figuras, podemos verificar que a relaxação linear do modelo $\mathrm{M}_{\mathrm{CLIQUE}}$ produz limitantes superiores mais apertados que o modelo $\mathrm{M}_{\mathrm{ARESTA}}$, principalmente quando consideramos as classes C_{bb}, C_{mm} e C_{aa}.

Por último, para os modelos $\mathrm{M}_{\mathrm{CLIQUE}}$ e $\mathrm{M}_{\mathrm{ITERATIVO}}$, foram conduzidos testes para se verificar o modo como a realização do procedimento de *branching* em diferentes variáveis afeta o desempenho do resolvedor. Foram efetuados testes fazendo o *branching* de 3 maneiras distintas: apenas nas variáveis correspondentes aos subconjuntos (Bs), apenas nas variáveis correspondentes aos elementos (Be) e em ambas as variáveis (Ba).

As Figuras 4.5 até 4.10 mostram o comportamento do modelo $\mathrm{M}_{\mathrm{CLIQUE}}$ para as classes de instâncias com a realização dos três tipos de *branching* para o grafo G_b. Os gráficos são exibidos utilizando a abordagem *performance profile* [Dolan and Moré(2002)], de forma que o "Fator de Performance"representa a razão entre o tempo de execução de determinado

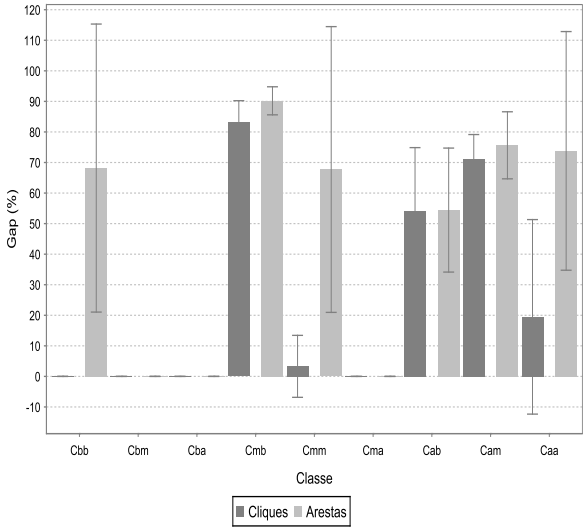


Figura 4.1: *Gap* de integralidade para as instâncias G_b.

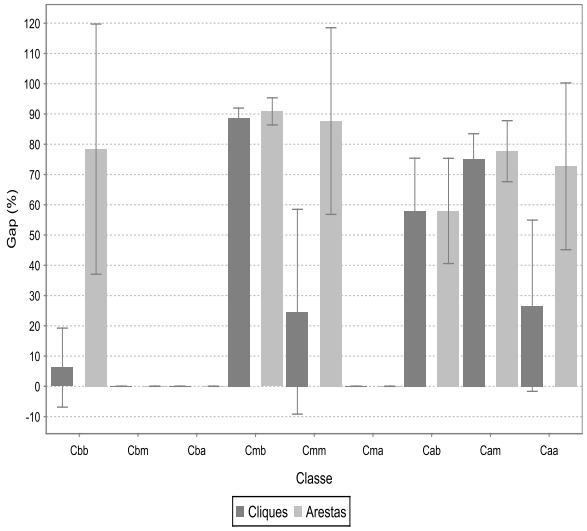


Figura 4.2: *Gap* de integralidade para as instâncias G_{de}.

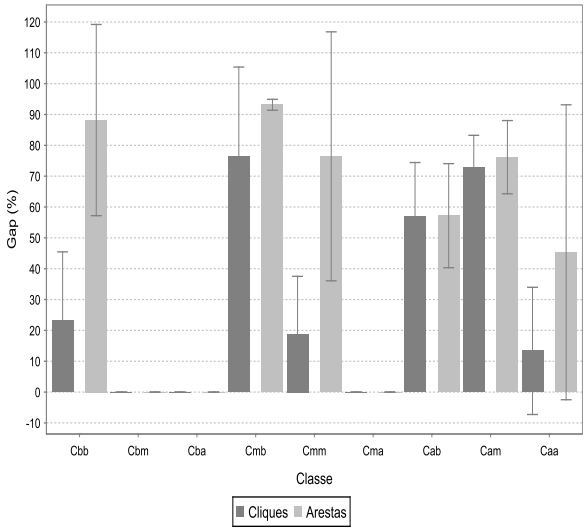


Figura 4.3: *Gap* de integralidade para as instâncias G_{ds}.

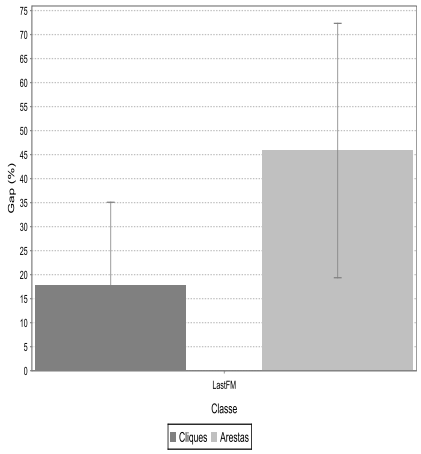


Figura 4.4: *Gap* de integralidade para as instâncias da LastFM.

tipo de *branching* contra o menor tempo de execução entre todos os tipos de *branching*. Cada ponto (x, y) no gráfico representa a porcentagem y de instâncias cujo fator de performance seja menor ou igual a x.

Através da porcentagem de instâncias resolvidas e de seu tempo de execução, foi determinado para cada grupo qual tipo de *branching* será empregado para cada modelo. Nas classes onde não ocorreram diferenças significativas no comportamento dos *branching*, arbitramos aleatoriamente o tipo de *branching* a ser efetuado. Como pode ser observado na Figura 4.5, para a classe C_{bb}, em 50% das instâncias o B_a superou os demais e, além disso, resolveu 90% das instâncias com fator de performance aproximadamente 3, enquanto o B_e superou os demais em 40% das instâncias e resolveu 80% delas com fator de performance próximo a 4, levando assim a escolha do B_a. Com base nestes resultados, nas Tabelas 4.6 e 4.7 exibimos o tipo de *branching* que ficou decidido para cada classe para os modelos M_{CLIQUE} e $\text{M}_{\text{ITERATIVO}}$, respectivamente, para todos os tipos de grafo. Como visto na Tabela 4.6, o tipo de *branching* varia muito dependendo da classe e do tipo de grafo. Denotamos através de um traço (-) as classes em que o pré-processamento foi capaz de remover a maioria dos vértices das instâncias, visto que não houve diferença significativa no comportamento dos *branching* para estas classes.

Não foi possível constatar um padrão na escolha do *branching* para todas as classes, contudo, os resultados fazem sentido de certa forma. Tome como exemplo a classe C_{ab}, em que os três tipos de *branching* ocorrem conforme cada tipo de grafo. Para o grafo G_b, o B_a é adotado pois o número de subconjuntos é igual ao número de elementos e, além disso, o tamanho da interseção é semelhante ao valor de k, o que torna as duas variáveis importantes para a realização do *branching*. Para o grafo G_{de}, devido ao número de subconjuntos ser baixo e o grafo possuir mais elementos do que subconjuntos, o que indica que o tamanho da interseção é grande, o B_s é escolhido. Por último, em virtude do grafo G_{ds} possuir um número maior de subconjuntos do que elementos e valor da interseção ser na maioria dos casos menor que o tamanho de subconjuntos, o B_e foi selecionado.

Para as instâncias da LastFM e para ambos os modelos, o *branching* Bs foi melhor, o que pode ser explicado devido ao baixo número de bandas em comparação ao número de ouvintes e pelos baixos valores de k estabelecidos.

4.3 Avaliação Experimental

Uma vez feito o ajuste de parâmetros descrito na seção anterior, deu-se início aos testes finais visando avaliar a eficiência do pré-processamento desenvolvido e o desempenho do resolvedor PLI com os modelos M_{CLIQUE} e $\text{M}_{\text{ITERATIVO}}$. As Seções 4.3.1 e 4.3.2 exibem os resultados dos modelos para as instâncias da LastFM e aleatórias, respectivamente. Por fim, a Seção 4.3.3 apresenta um resumo dos resultados observados durante este capítulo.

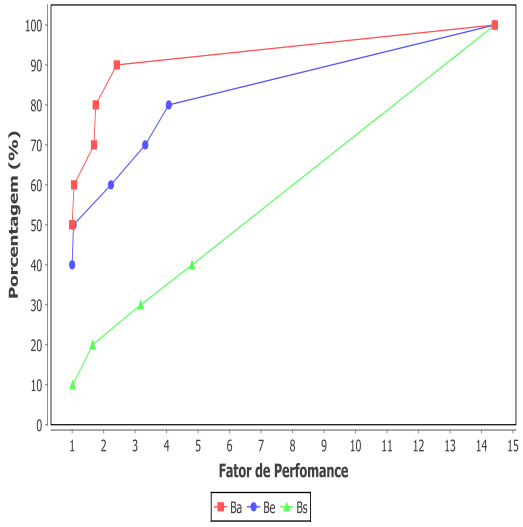


Figura 4.5: Classe C_{bb}.

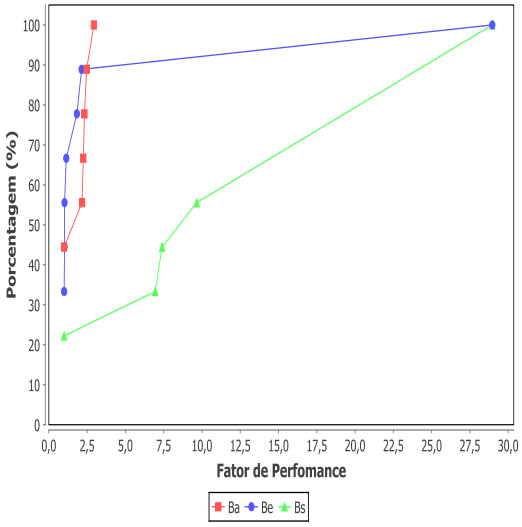


Figura 4.6: Classe C_{mb}.

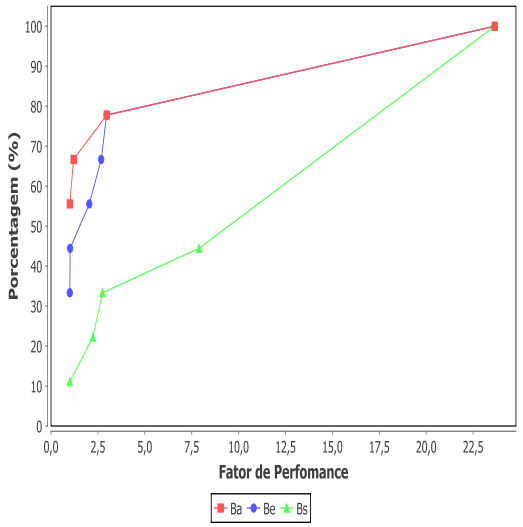


Figura 4.7: Classe C_{mm}.

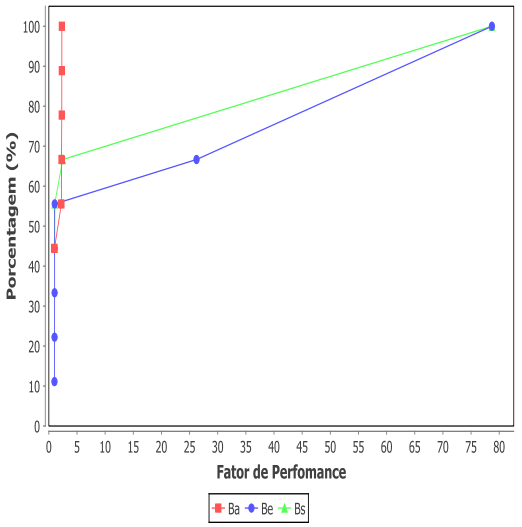


Figura 4.8: Classe C_{ab}.

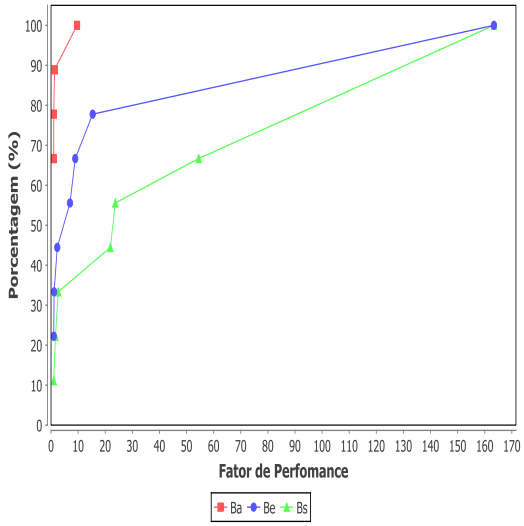


Figura 4.9: Classe C_{am}.

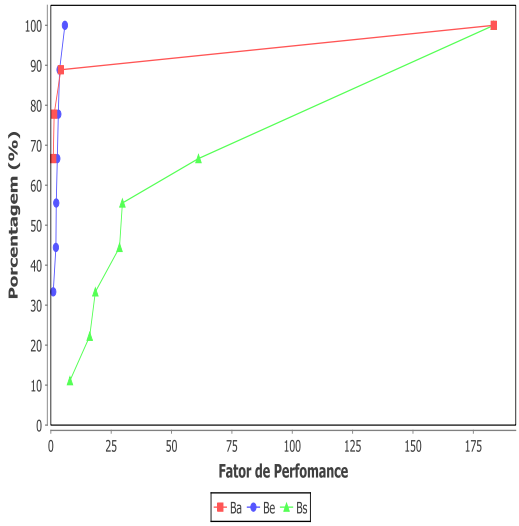


Figura 4.10: Classe C_{aa}.

4.3.1 Instâncias da LastFM

Inicialmente, medimos o impacto do pré-processamento na remoção de vértices das instâncias. Para todo o conjunto de instâncias da LastFM, exibimos na Tabela 4.8 a porcentagem média e o desvio padrão, entre parênteses, de ouvintes, bandas e arestas removidos pelo pré-processamento, separado pelos diferentes valores de k. Na Tabela 4.9, é exibido o número médio e o desvio padrão de ouvintes, bandas e arestas do grafo, antes e após o pré-processamento, separado pelas diferentes classes de instâncias.

Como exibido na Tabela 4.8, para $2 \leq k \leq 4$, o pré-processamento remove a maioria dos ouvintes e arestas. Para todos os valores de k avaliados, a porcentagem de ouvintes removidos foi muito maior do que a porcentagem de bandas removidas. Como pode ser observado, a eficiência do pré-processamento reduz conforme o valor de k aumenta. Isto pode ser explicado pelo fato de que um aumento no valor de k leva a uma redução no valor de λ (o valor da solução encontrada pelo $GRASP$).

Na Tabela 4.9, podemos observar que apesar do aumento do número de bandas, o número de ouvintes, bandas e arestas permanece quase o mesmo após o procedimento do pré-processamento. Este fenômeno é explicado pelo fato que, como mencionado anteriormente, existem poucas bandas com um alto número de ouvintes e a maioria dos vértices que representam as bandas possuem grau próximo de zero. Com isso, mesmo se selecionarmos um alto número de bandas, a maioria delas será removida pelo pré-processamento devido ao seu baixo número de ouvintes. De fato, para grafos gerados com até 4000 bandas, foi visto praticamente o mesmo número de ouvintes, bandas e arestas no respectivo grafo pré-processado, comparando com os valores reportados na Tabela 4.9.

Modelo M_{CLIQUE}			
Classe	G_b	G_{de}	G_{ds}
C_{bb}	Ba	Be	Bs
C_{bm}	-	-	-
C_{ba}	-	Be	Bs
C_{mb}	Ba	Be	Bs
C_{mm}	Be	-	-
C_{ma}	-	Be	Be
C_{ab}	Ba	Bs	Be
C_{am}	Be	Be	Be
C_{aa}	Be	-	Be

Tabela 4.6: Escolha do *branching* realizado em cada grupo para o modelo M_{CLIQUE}.

Modelo $M_{ITERATIVO}$			
Classe	G_b	G_{de}	G_{ds}
C_{bb}	Ba	Bs	Be
C_{bm}	-	-	-
C_{ba}	-	Bs	Be
C_{mb}	Bs	Bs	Be
C_{mm}	Ba	-	-
C_{ma}	-	Be	Bs
C_{ab}	Be	Bs	Be
C_{am}	Be	Be	Be
C_{aa}	Be	-	Ba

Tabela 4.7: Escolha do *branching* realizado em cada grupo para o modelo $M_{ITERATIVO}$.

Uma vez constatada a importância do pré-processamento, retornamos a nossa atenção para a escolha do modelo PLI. Para cada instância foi estabelecido um tempo máximo de 1800 segundos (30 minutos) de execução. Na Tabela 4.10 apresentamos a porcentagem de instâncias resolvidas de maneira ótima para cada modelo e, para aquelas em que isto não ocorre, exibimos os *gaps* obtidos como definido a seguir. Para o modelo M_{CLIQUE}, seja LP o valor da relaxação linear e sejam LB e UB os valores do melhor limitante inferior e superior encontrados durante a execução do *CPLEX*, respectivamente. Os valores na coluna "*Gap* de Integralidade" são dados por $\frac{\text{LP}-\text{LB}}{\text{LP}} \times 100$ enquanto os da coluna "*Gap* Final" são calculados por $\frac{\text{UB}-\text{LB}}{\text{UB}} \times 100$. Devido ao modelo $M_{ITERATIVO}$ apenas comprovar através das iterações se existem soluções de determinado valor λ, limitamo-nos a exibir a porcentagem de instâncias resolvidas de maneira ótima com este modelo. A coluna"% Otimos" mostra a porcentagem de instâncias resolvidas de forma exata.

Conforme se vê na Tabela 4.10, ambos os modelos resolveram o mesmo número de

	Porcentagem Removida					
k	Bandas		Ouvintes		Arestas	
2	48.3	(4.2)	97.7	(0.9)	97.2	(0.8)
3	18.4	(9.3)	84.8	(10.0)	80.5	(11.2)
4	9.8	(2.0)	64.3	(15.7)	57.9	(14.2)
5	8.4	(1.9)	44.4	(22.9)	40.3	(18.9)
6	7.5	(1.2)	24.8	(20.8)	24.6	(16.2)
7	7.7	(1.7)	10.7	(13.2)	14.7	(10.1)

Tabela 4.8: O efeito do pré-processamento para diferentes valores de k.

instâncias levando-se em conta a restrição de tempo imposta. A principal diferença entre os modelos ocorre no tempo de resolução das instâncias, como mostra a Figura 4.11, no qual apresentamos um diagrama de caixa do tempo de execução do *CPLEX* com os modelos M_{CLIQUE} e $\text{M}_{\text{ITERATIVO}}$, considerando unicamente as instâncias resolvidas por ambos os modelos. A parte inferior e superior do diagrama de caixa representam o primeiro e o terceiro quartil, respectivamente, e o risco dentro da caixa corresponde ao segundo quartil (a mediana). Os *whiskers*, as extremidades das linhas que se estendem verticalmente das caixas, são calculados da seguinte maneira: o maior valor menor que $q_3 + 1,5 \times (q_3 - q_1)$ para a extremidade superior e o menor valor maior que $q_1 - 1,5 \times (q_3 - q_1)$ para a extremidade inferior, em que q_1, q_2 e q_3 são o primeiro, segundo e terceiro quartil, respectivamente. Todo *outlier*, ou seja, todo ponto não incluso entre os *whiskers*, é desenhado como um círculo pequeno.

Como visto na Figura 4.11 e na Tabela 4.10, para $k \leq 3$, ambos os modelos resolveram todas as instâncias em menos de 2 minutos. Para todos os valores de k, podemos observar que o modelo M_{CLIQUE} é mais rápido que o modelo $\text{M}_{\text{ITERATIVO}}$. Independentemente do modelo, conforme o valor de k aumenta, o tempo de execução também cresce. Todavia, conforme exibido na Figura 4.12, após um intervalo, o tempo de execução diminui

Grafo Original					**Grafo Pré-Processado**					
Bandas	Ouvintes		Arestas		Bandas		Ouvintes		Arestas	
50	967	(2)	23191	(505)	30	(19)	793	(136)	13889	(8420)
70	974	(2)	30112	(558)	41	(26)	820	(160)	17787	(10351)
90	975	(2)	36616	(510)	44	(31)	820	(151)	18346	(11784)
110	977	(1)	42788	(308)	43	(34)	821	(144)	18178	(12356)
130	982	(1)	48001	(697)	47	(37)	818	(154)	18993	(13133)
150	982	(1)	53495	(768)	43	(36)	806	(151)	17812	(12931)

Tabela 4.9: O efeito do pré-processamento para diferentes números de bandas.

	Gap de Integralidade		Gap Final		% Ótimos Encontrados	
k	M_{CLIQUE}		M_{CLIQUE}		M_{CLIQUE}	$M_{ITERATIVO}$
2	0.0	(0.0)	0.0	(0.0)	100.0	100.0
3	0.0	(0.0)	0.0	(0.0)	100.0	100.0
4	11.4	(9.8)	0.0	(0.0)	100.0	100.0
5	29.3	(8.4)	0.0	(0.0)	100.0	100.0
6	40.3	(6.2)	8.3	(1.2)	96.6	95.0
7	46.9	(5.1)	14.1	(5.3)	70.0	71.6

Tabela 4.10: Porcentagem de instâncias resolvidas de maneira ótima e os *gaps* para as instâncias da LastFM.

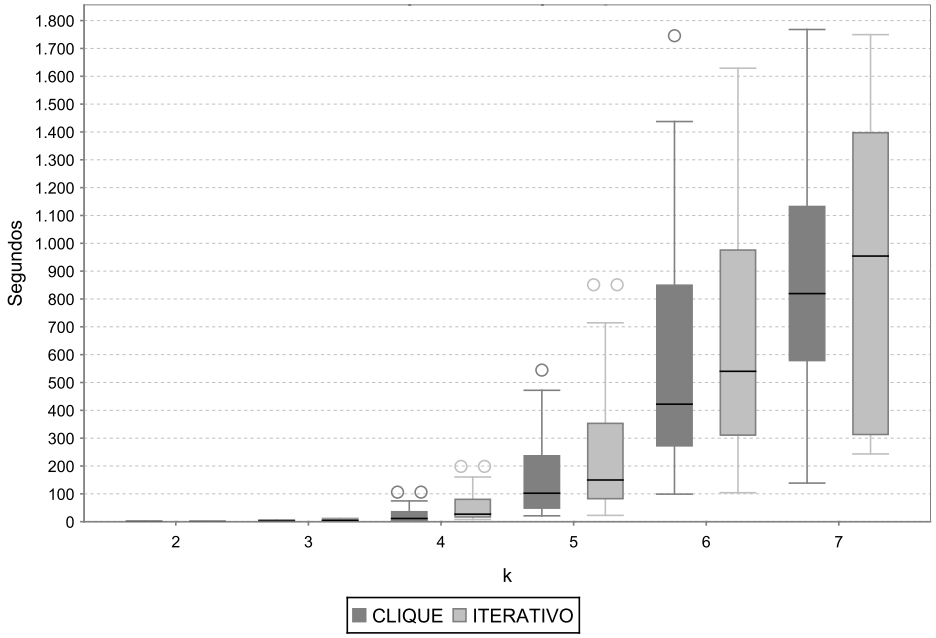


Figura 4.11: Comparação dos tempos de execução dos modelos M_{CLIQUE} e $M_{ITERATIVO}$.

conforme o valor de k aumenta. Isto bate com o esperado para um algoritmo de força bruta que testaria $\binom{|L|}{k}$ combinações de escolha de k subconjuntos, pois para valores de k pequeno e muito grande, o número de combinações é baixo.

4.3.2 Instâncias Aleatórias

Para as instâncias aleatórias descritas nas Tabelas 4.1, 4.2 e 4.3, exibimos na Tabelas 4.11, 4.12 e 4.13 a porcentagem de subconjuntos, elementos e arestas removidos pelo pré-processamento, separado pelas diferentes classes, para os grafos G_b, G_{de} e G_{ds}. Como

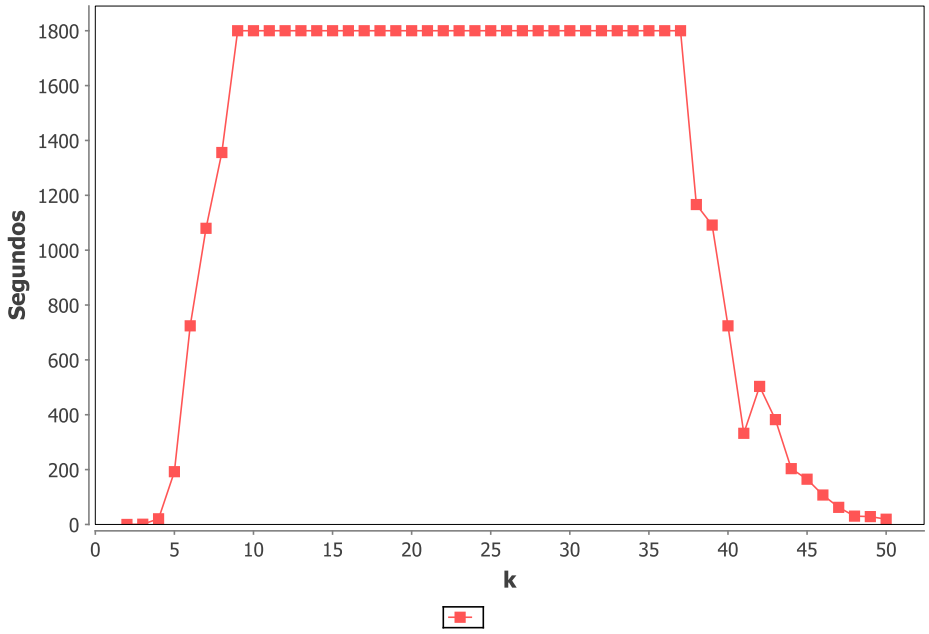


Figura 4.12: Comportamento do tempo de execução em função do valor de k para uma instância que $|L| = 50$.

se vê, o pré-processamento se mostra eficaz na tarefa de remoção de vértices e arestas das instâncias. Em cerca de 45% das classes o pré-processamento remove em média mais de 50% das arestas das instâncias. Além disso, para cerca de 19% das classes o pré-processamento elimina mais de 98% das arestas, chegando em determinados casos a resolver completamente o problema.

Conforme exibido anteriormente para as instâncias da LastFM, a Tabela 4.14 exibe os *gaps* de integralidade, dualidade e a porcentagem de soluções ótimas obtidas por ambos os modelos apenas para o grafo G_b, em virtude do comportamento similar dos modelos para todos os tipos de grafo (G_b, G_{de} e G_{ds}).

De modo geral, para todo o conjunto de instâncias aleatórias (incluindo os grafos G_b, G_{de} e G_{ds}), o modelo $\text{M}_{\text{ITERATIVO}}$ foi superior ao modelo M_{CLIQUE}, resolvendo 1418 instâncias, enquanto o modelo M_{CLIQUE} resolveu 1383, de um total de 1440 instâncias geradas aleatoriamente.

Com relação ao tempo de execução, analisando as instâncias que ambos os modelos conseguiram resolver, o resolvedor com o $\text{M}_{\text{ITERATIVO}}$ foi aproximadamente 1.15 vezes mais rápido do que com o M_{CLIQUE}. Com relação a heurística *GRASP*, um dado interessante de se notar é que, em 93% das instâncias, o valor da solução encontrada pelo *GRASP* foi ótimo, confirmando assim a eficiência da nossa meta-heurística. Para as instâncias em que o *GRASP* não atingiu a solução ótima, o *gap* médio entre a solução

	Porcentagem Removida					
Classe	Subconjutos		Elementos		Arestas	
C_{bb}	15.7	(34.2)	39.8	(39.2)	35.7	(38.2)
C_{bm}	96.9	(14.3)	99.8	(1.3)	99.7	(1.8)
C_{ba}	100.0	(0.0)	100.0	(0.0)	100.0	(0.0)
C_{mb}	0.0	(0.0)	0.0	(0.3)	0.0	(0.2)
C_{mm}	8.8	(26.0)	48.4	(35.0)	45.6	(35.1)
C_{ma}	92.7	(24.6)	99.6	(1.6)	99.5	(1.9)
C_{ab}	0.1	(0.5)	0.0	(0.0)	0.1	(0.4)
C_{am}	0.0	(0.0)	0.0	(0.0)	0.0	(0.0)
C_{aa}	8.4	(27.9)	43.8	(36.9)	42.6	(36.7)

Tabela 4.11: O efeito do pré-processamento para as diferentes classes no grafo G_b.

	Porcentagem Removida					
Classe	Subconjuntos		Elementos		Arestas	
C_{bb}	21.3	(37.4)	49.3	(39.3)	43.4	(39.2)
C_{bm}	83.3	(33.6)	99.3	(1.7)	98.9	(2.9)
C_{ba}	50.3	(50.2)	50.2	(50.2)	50.4	(50.1)
C_{mb}	0.0	(0.0)	25.1	(28.5)	21.4	(26.0)
C_{mm}	59.3	(46.2)	91.5	(20.3)	90.2	(21.9)
C_{ma}	6.1	(16.6)	0.2	(1.3)	6.0	(15.9)
C_{ab}	0.0	(0.0)	16.7	(29.6)	15.8	(28.6)
C_{am}	1.7	(12.9)	68.4	(35.7)	66.6	(35.8)
C_{aa}	0.0	(0.0)	94.2	(0.0)	92.9	(0.0)

Tabela 4.12: O efeito do pré-processamento para as diferentes classes no grafo G_{de}

ótima e a solução heurística foi de aproximadamente 30%, sendo que destas instâncias, aproximadamente 65% possuem densidade alta, o que nos leva a concluir que o *GRASP* produz melhores resultados para grafos pouco densos.

4.3.3 Resumo dos Resultados

Como visto, os modelos baseados no método de *Branch-and-Cut* apresentaram resultados superiores dos modelos M_{ARESTA}, (MEB V) e (MEB L), que são baseados apenas no método de *Branch-and-Bound*. Através da Tabela 4.6, também foi possível verificar que dependendo do tipo de classe de instâncias, a escolha da variável para *branching* é um fator muito relevante para a resolução de uma instância, visto que essa escolha pode afetar significativamente o desempenho do resolvedor. Por fim, através dos testes computacionais

	Porcentagem Removida					
Classe	Subconjuntos		Elementos		Arestas	
C_{bb}	35.9	(44.2)	52.9	(43.2)	50.7	(43.1)
C_{bm}	98.9	(8.5)	99.9	(0.9)	99.8	(1.2)
C_{ba}	50.1	(50.3)	50.0	(50.4)	50.0	(50.4)
C_{mb}	11.8	(28.7)	31.4	(39.8)	30.6	(39.2)
C_{mm}	83.6	(36.9)	88.2	(28.8)	87.9	(29.5)
C_{ma}	1.2	(5.1)	0.0	(0.0)	1.1	(4.8)
C_{ab}	3.9	(18.2)	15.9	(33.6)	15.7	(33.3)
C_{am}	5.2	(22.0)	20.1	(34.7)	19.8	(34.4)
C_{aa}	0.0	(0.0)	0.0	(0.0)	0.0	(0.0)

Tabela 4.13: O efeito do pré-processamento para as diferentes classes no grafo G_{ds}.

	Gap de Integralidade		**Gap Final**		**% Ótimos Encontrados**	
k	M_{CLIQUE}		M_{CLIQUE}		M_{CLIQUE}	$\text{M}_{\text{ITERATIVO}}$
C_{bb}	13.0	(23.8)	0.0	(0.0)	100.0	100.0
C_{mb}	79.8	(12.9)	15.5	(29.1)	78.3	100.0
C_{mm}	3.7	(12.4)	0.0	(0.0)	100.0	100.0
C_{ab}	60.9	(25.2)	30.0	(30.0)	41.7	65.0
C_{am}	72.8	(15.9)	0.0	(0.0)	100.0	98.3
C_{aa}	13.3	(19.6)	0.0	(0.0)	100.0	100.0

Tabela 4.14: Porcentagem de instâncias resolvidas de maneira ótima e os *gaps* para as instâncias do grafo G_b.

realizados com os modelos M_{CLIQUE} e $\text{M}_{\text{ITERATIVO}}$, foi observado que para as instâncias da LastFM, o modelo M_{CLIQUE} foi superior ao modelo $\text{M}_{\text{ITERATIVO}}$, enquanto para as instâncias aleatórias, o modelo $\text{M}_{\text{ITERATIVO}}$ produziu resultados melhores que o modelo M_{CLIQUE}.

Capítulo 5

Conclusão

Neste trabalho, foram propostos cinco modelos de programação linear inteira e duas heurísticas para o problema da Máxima Interseção de k-Subconjuntos, que até então não havia sido tratado de maneira algorítmica na literatura.

Das formulações desenvolvidas, $\mathrm{M}_{\mathrm{CLIQUE}}$ e $\mathrm{M}_{\mathrm{ITERATIVO}}$, baseadas no método *Branch-and-Cut*, atingiram os melhores resultados. Uma vez calculadas pelo CPLEX, elas foram capazes de resolver instâncias de tamanho relativamente grande, como demonstrado pelos experimentos computacionais realizados tanto em instâncias geradas aleatoriamente quanto em instâncias vindas de uma aplicação real.

Com relação às heurísticas propostas, foi observado que a meta-heurística $GRASP$ obteve resultados superiores aos da heurística gulosa, sendo capaz de atingir a solução ótima em cerca de 93% das instâncias e tendo um *gap* médio de cerca de 30% em relação ao valor ótimo das instâncias restantes. Além disso, foi introduzido também um efetivo procedimento de pré-processamento para reduzir o tamanho da entrada de instâncias que, em alguns casos, conseguiu resolver o problema completamente. O bom desempenho obtido pelo pré-processamento foi alcançado em grande parte devido à boa qualidade das soluções produzidas pelo $GRASP$.

Através deste trabalho de mestrado, financiado pela FAPESP (projeto: 2012/08298-6), foram publicados dois artigos, sendo o primeiro em um simpósio nacional e o segundo em um internacional ([Bogue et al.(2013)Bogue, de Souza, Xavier, and Freire] e [Bogue et al.(2014)Bogue, de Souza, Xavier, and Freire], respectivamente).

Uma possibilidade para trabalhos futuros seria estudar o politopo $\mathcal{P}_{\mathrm{IP2}}$ com o intuito de descobrir mais desigualdades que definem facetas. O ponto de partida natural para essa pesquisa seria investigar as desigualdades definidoras de facetas do problema do conjunto independente que, como visto, está intimamente relacionado ao kMIS. Outra possibilidade seria desenvolver algoritmos exatos que não são baseados em técnicas de programação linear inteira, como Programação por Restrições. Por fim, seria interessante

descobrir se existem classes de grafos onde o problema possa ser resolvido de maneira exata em tempo polinomial. Como mencionado na Seção 1.1, se o grafo de entrada é convexo então o kMIS pode ser resolvido em tempo polinomial. Assim, é possível que o mesmo ocorra para outras classes de grafos relevantes.

Referências Bibliográficas

[Acuña et al.(2014)Acuña, Ferreira, Freire, and Moreno] V. Acuña, C.E. Ferreira, A.S. Freire, and E. Moreno. Solving the maximum edge biclique packing problem on unbalanced bipartite graphs. *Discrete Applied Mathematics*, 164, Part 1:2 – 12, 2014. doi: 10.1016/j.dam.2011.09.019. URL `http://www.sciencedirect.com/science/article/pii/S0166218X11003556`.

[Alexe et al.(2004)Alexe, Alexe, Crama, Foldes, Hammer, and Simeone] G. Alexe, S. Alexe, Y. Crama, S. Foldes, P. L. Hammer, and B. Simeone. Consensus algorithms for the generation of all maximal bicliques. *Discrete Applied Mathematics*, 145(1):11 – 21, 2004. ISSN 0166-218X. doi: 10.1016/j.dam.2003.09.004. URL `http://www.sciencedirect.com/science/article/pii/S0166218X04000629`.

[Bogue et al.(2013)Bogue, de Souza, Xavier, and Freire] E. T. Bogue, C. C. de Souza, E. C. Xavier, and A. S. Freire. O problema da máxima interseção de k-subconjuntos. In *Simpósio Brasileiro de Pesquisa Operacional*, 2013.

[Bogue et al.(2014)Bogue, de Souza, Xavier, and Freire] E. T. Bogue, C. C. de Souza, E. C. Xavier, and A. S. Freire. A note on a maximum k-subset intersection problem. In *3rd International Symposium. on Combinatorial Optimization*, 2014.

[Celma(2008)] O. Celma. *Music Recommendation and Discovery in the Long Tail*. PhD thesis, Universitat Pompeu Fabra, 2008.

[Clifford and Popa(2011)] R. Clifford and A. Popa. Maximum subset intersection. *Information Processing Letters*, 111:323 – 325, 2011. doi: 10.1016/j.ipl.2010.12.003. URL `http://www.sciencedirect.com/science/article/pii/S0020019010003959`.

[Dias et al.(2005)Dias, de Figueiredo, and Szwarcfiter] V.M.F. Dias, C.M.H. de Figueiredo, and J. L. Szwarcfiter. Generating bicliques of a graph in lexicographic order. *Theoretical Computer Science*, 337(1-3):240 – 248, 2005. ISSN 0304-3975. doi: 10.1016/j.tcs.2005.01.014. URL `http://www.sciencedirect.com/science/article/pii/S030439750500037X`.

[Dijkstra(1959)] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. ISSN 0029-599X. doi: 10.1007/BF01386390. URL `http://dx.doi.org/10.1007/BF01386390`.

[Dolan and Moré(2002)] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002. ISSN 0025-5610. doi: 10.1007/s101070100263. URL `http://dx.doi.org/10.1007/s101070100263`.

[Erdos and Renyi(1960)] P. Erdos and A. Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–61, 1960.

[Feo and Resende(1989)] Thomas A Feo and Mauricio G. C Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Oper. Res. Lett.*, 8 (2):67–71, April 1989. ISSN 0167-6377. doi: 10.1016/0167-6377(89)90002-3. URL `http://dx.doi.org/10.1016/0167-6377(89)90002-3`.

[Gilbert(1959)] E. N. Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 12 1959. doi: 10.1214/aoms/1177706098. URL `http://dx.doi.org/10.1214/aoms/1177706098`.

[Glover(1996)] F. Glover. *Tabu search and adaptive memory programming – advances, applications, and challenges*. Kluwer Academic Publishers, 1996.

[Huffman(1952)] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, September 1952.

[Karmarkar(1984)] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 302–311, New York, NY, USA, 1984. ACM. ISBN 0-89791-133-4. doi: 10.1145/800057.808695. URL `http://doi.acm.org/10.1145/800057.808695`.

[Kruskal(1956)] Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7 (1):48–50, February 1956. URL `http://www.jstor.org/stable/2033241`.

[Nemhauser and Sigismondi(1992)] G. L. Nemhauser and G. Sigismondi. A strong cutting plane/branch-and-bound algorithm for node packing. *The Journal of the Operational Research Society*, 43:443–457, 1992.

[Newman(2005)] MEJ Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics*, 46(5):323–351, 2005.

[Nussbaum et al.(2010)Nussbaum, Pu, Sack, Uno, and Takeaki] D. Nussbaum, S. Pu, J. Sack, T. Uno, and H. Takeaki. Finding maximum edge bicliques in convex bipartite graphs. In My Thai and Sartaj Sahni, editors, *Computing and Combinatorics*, volume 6196 of *Lecture Notes in Computer Science*, pages 140–149. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-14030-3. URL `http://dx.doi.org/10.1007/978-3-642-14031-0_17`.

[Peeters(2003)] R. Peeters. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, 131:651 – 654, 2003. doi: 10.1016/S0166-218X(03)00333-0. URL `http://www.sciencedirect.com/science/article/pii/S0166218X03003330`.

[Prais and Ribeiro(2000)] M. Prais and C.C. Ribeiro. Parameter variation in GRASP procedures. *Investigación Operativa*, 9:1–20, 2000. URL `http://yahoo.com`.

[Prais et al.(1998)Prais, Ribeiro, Celso, and Ribeiro] Marcelo Prais, Celso C. Ribeiro, Celso, and C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in tdma traffic assignment. *Informs Journal on Computing*, 12:164–176, 1998.

[Prim(1957)] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technology Journal*, 36:1389–1401, 1957.

[Rebennack et al.(2012)Rebennack, Reinelt, and Pardalos] S. Rebennack, G. Reinelt, and P. M. Pardalos. A tutorial on branch and cut algorithms for the maximum stable set problem. *International Transactions in Operational Research*, 19:161–199, 2012.

[Ribeiro et al.(2001)Ribeiro, Uchoa, and Werneck] Celso C. Ribeiro, Eduardo Uchoa, and Renato F. Werneck. A hybrid grasp with perturbations for the steiner problem in graphs. *Informs Journal on Computing*, 14:200–2, 2001.

[Vinterbo(2002)] S.A. Vinterbo. A note on the hardness of the k-ambiguity problem. Technical report, Harvard Medical School, Boston, MA, USA, 2002.

[Xavier(2012)] E. C. Xavier. A note on a maximum k-subset intersection problem. *Information Processing Letters*, 112:471 – 472, 2012. doi: 10.1016/j.ipl.2012.03.007. URL `http://www.sciencedirect.com/science/article/pii/S0020019012000750`.