Algoritmos para Resolução do Problema de Empacotamento de Conjuntos Utilizando Poliedros Quase Inteiros

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Claudia Akemi Furushima Porto e aprovada pela Banca Examinadora.

Campinas, 10 de Dezembro de 2010.

Prof. Dr. Cid Carvalho de Souza
Instituto de Computação, Unicamp
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Fabiana Bezerra Müller – CRB8 / 6162

Porto, Claudia Akemi Furushima
P838a Algoritmos para resolução do problema de empacotamento de conjuntos utilizando poliedros quase inteiros/Claudia Akemi Furushima Porto-- Campinas, [S.P. : s.n.], 2010.

Orientador : Cid Carvalho de Souza.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1.Otimização combinatória. 2.Combinatória poliédrica. 3.Algoritmo Branch and Cut. 4.Problema de empacotamento. I. Souza, Cid Carvalho de . II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Algorithms for the set packing problem using quasi integer polyhedra

Palavras-chave em inglês (Keywords): 1. Combinatorial optimization. 2. Polyhedral combinatorics. 3. Branch-and-Cut algorithms. 4. Set packing.

Área de concentração: Teoria da Computação

Titulação: Mestre em Ciência da Computação

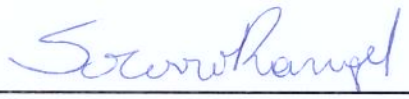
Banca examinadora: Prof. Dr. Cid Carvalho de Souza (IC-UNICAMP)
Profª. Drª. Maria do Socorro Nogueira Rangel (IBILCE-UNESP)
Prof. Dr. Flávio Keidi Miyazawa (IC-UNICAMP)

Data da defesa: 10/12/2010

Programa de Pós-Graduação: Mestrado em Ciência da Computação

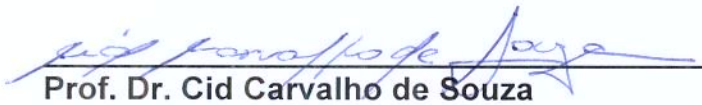
TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 10 de dezembro de 2010, pela Banca examinadora composta pelos Professores Doutores:



Profª. Drª. Maria do Socorro Nogueira Rangel
Ibilce / UNESP

Prof. Dr. Flávio Keidi Miyazawa
IC / UNICAMP



Prof. Dr. Cid Carvalho de Souza
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Algoritmos para Resolução do Problema de Empacotamento de Conjuntos Utilizando Poliedros Quase Inteiros

Claudia Akemi Furushima Porto[1]

Dezembro de 2010

Banca Examinadora:

- Prof. Dr. Cid Carvalho de Souza
  Instituto de Computação, Unicamp (Orientador)
- Profa. Dra. Maria do Socorro Nogueira Rangel
  Instituto de Biociências, Letras e Ciências Exatas de São José do Rio Preto, Unesp
- Prof. Dr. Flávio Keidi Miyazawa
  Instituto de Computação, Unicamp
- Prof. Dr. Vinicius Amaral Armentano
  Faculdade de Engenharia Elétrica e de Computação, Unicamp (Suplente)
- Prof. Dr. Orlando Lee
  Instituto de Computação, Unicamp (Suplente)

[1]Suporte financeiro de: Bolsa da FAPESP (processo 07/53616-8) 2007–2009.

Resumo

Seja P um poliedro, onde $P = \{x \in \mathbb{R}^n : Ax \leq \mathbb{1}^m\}$, e $S = P \cap \mathbb{B}^n$, sendo $\mathbb{B}^n$ o conjunto de vetores binários de dimensão n, A uma matriz de tamanho $m \times n$ e apenas coeficientes com valores em $\{0, 1\}$ e $\mathbb{1}^m$ um vetor de tamanho m com elementos de valor 1.

Neste trabalho visamos resolver problemas de maximização de uma função de domínio S, conhecidos também pelo nome *Set Packing*, através de um algoritmo de *branch-and-bound* com adição de cortes. A idéia aqui consiste em desmembrá-los em vários subproblemas da forma $P_i = A'_i x \leq \mathbb{1}^{m'_i}$, sendo $\bigcup P_i = P$ e A'_i uma submatriz de A de tamanho $m'_i \times n$, com $m'_i - 1$ linhas formando uma matriz Totalmente Unimodular, no nosso caso uma matriz de rede pura refletida, e A' uma matriz Quase Totalmente Unimodular .

Sabemos que estes problemas pertencem à classe de problemas NP-difíceis, e portanto tentar resolvê-los em sua otimalidade é uma tarefa árdua, exceto quando as restrições do problemas formam sua própria envoltória convexa. Portanto, com a intenção de encolher o poliedro P e assim tentarmos agilizar a convergência do algoritmo, inserimos alguns planos de corte à instância.

Através do estudo da envoltória convexa definida por uma matriz Quase Totalmente Unimodular, encontramos uma família de desigualdades válidas fortes para P_i, e as inserimos. O intuito deste trabalho foi utilizarmos tal estudo para adicionarmos cortes fáceis de serem encontrados e fortes o suficiente para diminuirmos o tempo necessário para obter a solução ótima das instâncias tratados.

Abstract

Let P be a polihedra, where $P = \{x \in \mathbb{R}^n : Ax \leq \mathbb{1}^m\}$, and $S = P \cap \mathbb{R}^n$. $\mathbb{R}^n$ is the subset of binary vectors of dimension n , A is a $m \times n$ sized matrix which coefficients are in $\{0, 1\}$ and $\mathbb{1}^m$ is a m-element array of ones.

In this work we strived to solve maximization problems of a function whose domain is S, also known as Set Packing, with a branch-and-cut algorithm with cut addition. The idea consists of breaking them into several smaller $P_i = \{A'_i x \leq \mathbb{1}^{m'_i}\}$ subproblems, with $\bigcup P_i = P$ and A'_i being a $m'_i \times n$ submatrix of A with $m'_1 - 1$ lines forming a Totally Unimodular matrix (a pure, reflected network matrix in our particular case), and A' being a quasi-totally unimodular matrix.

These problems are known to belong to the NP-hard set of problems, therefore trying to solve them optimally is an arduous task, except when the problem's constraints define its own convex hull. Thus, we add a few cut planes to the instance being solved to shrink P and therefore speedup the algorithm's convergence.

Through the study of the convex hull defined by a quasi-totally unimodular matrix, a family of strong inequalities valid for P_i are found and added to the instance. The goal of this work was to utilize such study to add cuts that are both easy to find and strong enough to decrease the time necessary to optimally solve Set Packing instances.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

Nesta dissertação é estudado o *problema do empacotamento* (*SPack*) sob a óptica da Programação Inteira e da Combinatória Poliédrica. O objetivo é a obtenção de desigualdades válidas a partir de subestruturas da matriz de restrições cuja identificação não seja muito custosa e que, ao mesmo tempo, possibilitem o desenvolvimento de algoritmos *branch-and-cut* (*B&C*) capazes de resolver o problema de forma exata em um baixo tempo de computação. Formalmente, o SPack pode ser definido como segue.

Seja um conjunto finito $M = \{1, \ldots, m\}$ e $C = \{C_1, \ldots, C_n\}$ um conjunto de subconjuntos de M. Denote-se por $N = \{1, \ldots, n\}$ o conjunto de índices de C. Um *empacotamento de* M consiste em um subconjunto F de N de modo que todo elemento de M pertence a no máximo um dos subconjuntos C_j, para algum j em F. Em outras palavras, diz-se que $F \subseteq N$ é um empacotamento de M se e somente se $C_i \cap C_j = \emptyset,\ \forall\ i, j \in F$.

Agora, suponha que a cada $j \in N$, esteja associado um custo c_j. Define-se o custo de qualquer subconjunto F de N, denotado por $c(F)$, como sendo a soma dos custos dos elementos de F, ou seja, $c(F) = \sum_{j \in F} c_j x_j$. No *problema do empacotamento*, o objetivo é encontrar um empacotamento F de custo máximo.

Para formular o SPack como um problema de Programação Linear Inteira, considere a matriz de restrições A como sendo a matriz de incidência dos elementos de M nos conjuntos C_j, ou seja, para todo $i \in M$ e todo $j \in N$, temos:

$$a_{ij} = \begin{cases} 1, \text{ se } i \in C_j, \\ 0, \text{ caso contrário.} \end{cases}$$

As variáveis do modelo são binárias e estão associadas biunivocamente aos subconjuntos de C. Supondo que F seja o empacotamento que corresponde à solução do modelo, essas variáveis identificam quais são os subconjuntos de C que estão presentes em F. Deste modo, as variáveis são definidas por:

$$x_j = \begin{cases} 1, \text{ se } j \in F \text{ (ou seja, se o subconjunto } j \text{ é selecionado)}, \\ 0, \text{ caso contrário.} \end{cases}$$

É fácil perceber que, com as variáveis x definidas acima, o custo de F que corresponde à função objetivo (FO) do problema é dado por $\sum_{j \in N} c_j x_j = \sum_{j \in F} c_j$.

Logo, a partir das definições anteriores, o SPack pode ser modelado por:

$$\begin{array}{lll} \max & \sum_{j=1}^{n} c_j x_j & \\ \text{s.a} & \sum_{j=1}^{n} a_{ij} x_j \leq 1 & \forall\ i \in M, \\ & x_j \in \{0,1\}, & \forall\ j \in N. \end{array} \tag{1.1}$$

Este problema é bem conhecido na literatura possuindo inúmeras aplicações práticas. Ele é difícil de ser resolvido fazendo parte dos 21 problemas NP-completos originalmente apresentados por Karp em [20]. Existem ainda duas outras famílias de problemas intimamente relacionadas com o empacotamento. Esses problemas possuem os mesmo dados de entrada e podem ser formulados usando as mesmas variáveis binárias apresentadas anteriormente. São eles:

1. **Problema da cobertura (SCov).**

   Diz-se que $F \subseteq N$ é uma cobertura de M se e somente se $\bigcup_{j \in F} C_j = M$.

   Sua formulação é dada por:

$$\begin{array}{lll} \min & \sum_{j=1}^{n} c_j x_j & \\ \text{s.a.} & \sum_{j=1}^{n} a_{ij} x_j \geq 1 & \forall\ i \in M, \\ & x_j \in \{0,1\}, & \forall\ j \in N. \end{array} \tag{1.2}$$

2. **Problema da partição (SPar).**

   Diz-se que $F \subseteq N$ é uma partição de M se e somente se for um empacotamento e uma cobertura ao mesmo tempo, cuja formulação é:

$$\begin{array}{lll} \min & \sum_{j=1}^{n} c_j x_j & \\ \text{s.a.} & \sum_{j=1}^{n} a_{ij} x_j = 1 & \forall\ i \in M, \\ & x_j \in \{0,1\}, & \forall\ j \in N. \end{array} \tag{1.3}$$

Os principais métodos de resolução exata dos três problemas baseiam-se em Programação Linear Inteira e no uso de algoritmos de B&B, B&C ou relaxação Lagrangeana, vistos mais detalhadamente na seção 2.1. Mas, apesar dos avanços algorítmicos dos últimos anos, ainda existem dificuldades em resolver instâncias do porte daquelas encontradas em aplicações práticas. Nesta dissertação investigamos alternativas que pudessem melhorar esse quadro.

1.1 Motivação

O uso de algoritmos de planos de corte para a resolução de problemas de Programação Linear Inteira (*PLI*) tornou-se uma das principais ferramentas disponíveis para a resolução exata de problemas combinatórios difíceis, notadamente a partir da década de 80. Isto ocorreu principalmente graças ao estudo da estrutura facial dos poliedros associados às envoltórias convexas das soluções desses problemas. A imersão dos algoritmos de planos de corte faciais em métodos enumerativos do tipo *branch-and-bound* (*B&B*) deu origem aos chamados algoritmos de *branch-and-cut* (*B&C*) e *cut-and-branch* (*C&B*) que, em muitos casos, levaram à resolução exata de problemas antes considerados intratáveis.

Diversos tipos de PLIs possuem todas as suas variáveis binárias, e iremos denotá-los por *PLI 0-1*. Além do SPack, um outro exemplo que se encaixa nessa descrição é o *problema da mochila* (binária), definido abaixo.

Suponha uma mochila que agüenta um peso máximo W, e um conjunto de objetos N tal que $|N| = n$. Cada objeto j em N tem um valor c_j e um peso w_j associado a ele. As soluções viáveis da mochila são os subconjuntos de N formados por objetos cuja soma dos pesos não ultrapassa o peso máximo que a mochila comporta. Entre todas as possíveis combinações, a solução desejada é um conjunto $M \subseteq N$ de objetos que podem ser colocados dentro da mochila simultaneamente tal que a soma de seus valores seja máximo. Assim, podemos definir sua formulação como:

$$\begin{array}{ll} \max & c^T x \\ \text{s.a.} & w^t x \leq W, \\ & x \in \mathbb{N}^n, \end{array}$$

sendo que, para todo $j \in N$, a variável x_j é definida da seguinte maneira:

$$x_j = \begin{cases} 1, & \text{se } j \in M, \\ 0, & \text{caso contrário.} \end{cases}$$

O problema da mochila tem muitas aplicações práticas e é muito importante para a área de Otimização Combinatória. Por este motivo, foi amplamente estudado na literatura. Destaca-se o fato de que diversas desigualdades válidas fortes e até definidoras de facetas são conhecidas para o poliedro correspondente à sua envoltória convexa. Note que para um PLI 0-1 geral, esse conhecimento pode levar ao desenvolvimento de formulações mais fortes já que, a princípio, podemos considerar cada restrição do modelo como sendo uma mochila binária e, assim, substituí-la por uma descrição, ainda que parcial, da sua envoltória convexa. Para melhor explicar esta idéia, considere um PLI 0-1 geral dado por $\max\{cx \ : \ Ax \leq b, x \in \mathbb{N}^n\}$, sendo A uma matriz $m \times n$ e b um vetor $m \times 1$. Seja P a envoltória convexa do conjunto de soluções desse problema. Ademais, para $q = 1, \ldots, m$,

defina P_i como sendo a envoltória convexa dos vetores $x \in \mathbb{N}^n$ que satisfazem à i-ésima restrição definida pela matriz A, isto é, $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$. Com estas definições, é fácil ver que

$$P \subseteq \bigcap_{i=1}^{m} P_i.$$

Desta maneira, qualquer desigualdade que define uma faceta de P_i descreve uma desigualdade válida para P. Esta estratégia pode ser utilizada na obtenção de desigualdades válidas fortes para o PLI 0-1, melhorando os limitantes duais da formulação e, conseqüentemente, possibilitando resolvê-lo mais rápido.

Esta técnica foi testada com sucesso em um trabalho desenvolvido por Crowder, Johnson e Padberg [10]. Nele os autores apresentam resultados extremamente encorajadores de um algoritmo de C&B usado para resolver PLIs difíceis definidos sobre variáveis binárias sujeitas a várias restrições do tipo mochila. Durante a execução do algoritmo, sempre que um ponto fracionário precisava ser separado, cada desigualdade da formulação era tratada independentemente como se fosse uma única mochila binária para a qual fosse necessário resolver o problema de separação. Nesse caso, as classes de desigualdades procuradas eram formadas por desigualdades válidas fortes para o problema da mochila.

A seguir mostra-se como este trabalho pioneiro desenvolvido por Crowder, Johnson e Padberg serviu de fonte de inspiração para a pesquisa desenvolvida nesta dissertação. A grosso modo, como será visto, nossa estratégia pautou-se no estudo da envoltória convexa de um problema combinatório simples, porém não trivial, satisfazendo a propriedade de que a envoltória convexa das soluções do SPack está contida na intersecção de várias destas envoltórias. Fazendo uma analogia com o trabalho desenvolvido em [10], em relação ao SPack, esse problema mais simples teria um papel semelhante àquele desempenhado pelo problema da mochila em relação ao PLI 0-1 geral.

1.2 Pesquisa desenvolvida

A idéia descrita na seção 1.1 não pode ser aplicada diretamente ao SPack pois, como visto, todos os coeficientes da matriz de restrições desse problema têm valor 1 ou 0. Portanto, como veremos no capítulo 2, uma restrição qualquer do SPack juntamente com as desigualdades triviais ($0 \leq x_i \leq 1$) já descrevem sua própria envoltória convexa. Ou seja, nehuma nova desigualdade referente à formulação original do problema é adicionada. Na verdade, para todo PLI que é formulado com uma matriz de restrições que é *totalmente unimodular (TU)*, a envoltória convexa das soluções é dada pela própria relaxação linear do modelo. Esse é o caso de qualquer problema de empacotamento descrito por uma ou duas desigualdades.

Para explicar o estudo realizado nesta dissertação, introduz-se a seguinte notação. Se M uma matriz $p \times q$, o poliedro $P(M)$ é definido por: $P(M) = \{x \in \mathbb{R}^q : Mx \leq \mathbb{1}_m \text{ e, } 0 \leq x_j \leq 1, \forall\ j = 1, \ldots, q\}$, sendo $\mathbb{1}_p$ um vetor de p elementos, todos de valor um. A notação $conv(P(M))$ será usada para designar a envoltória convexa das soluções inteiras de $P(M)$. Finalmente, daqui até o final do texto, denotaremos por A a matriz de restrições do SPack e usaremos P para nos referirmos a $P(A)$.

Uma vez introduzidas as definições acima e supondo A com dimensão $m \times n$, podemos também supor que sempre existe um subconjunto de linhas de A que formam uma submatriz TU B de dimensão $m' \times n$ tal que $m' \geq 2$. Vamos supor que B é uma submatriz TU maximal de A.

Como nosso intuito é resolver SPacks difíceis, vamos considerar que o poliedro P possua no mínimo um ponto extremo fracionário. Sendo assim, existe uma linha π que está em $A \setminus B$ tal que o poliedro $\{x \in \mathbb{R}^n : \begin{pmatrix} B \\ \pi \end{pmatrix} x \leq \mathbb{1}_{m'+1},\ \text{e, } 0 \leq x_j \leq 1, \forall\ j = 1, \ldots, q\}$ não é inteiro, ou seja, tem um ponto extremo fracionário. Nesse caso, dizemos que a matriz $\begin{pmatrix} B \\ \pi \end{pmatrix}$ é *quase totalmente unimodular (QTU).*

Note que, apesar de existir na literatura outra definição para matrizes quase totalmente unimodulares, como pode ser visto em [13], a definição acima é a que será utilizada nesta dissertação.

Sabemos que, em várias instâncias oriundas de aplicações práticas do SPack, é comum que a matriz A possua densidade extremamente baixa (menos de 5%). Isso nos leva a acreditar que temos boas chances de encontrar grandes submatrizes de A que sejam TUs maximais. Sendo B uma tal submatriz, ao uní-la com qualquer outra linha de $A \setminus B$, teremos uma matriz QTU. Então, conforme mencionado no final da subseção anterior, podemos pensar em usar os algoritmos de planos de corte baseados em desigualdades válidas fortes para essas QTUs para desenvolver um B&C genérico para o SPack. Essa foi justamente a estratégia concebida neste trabalho.

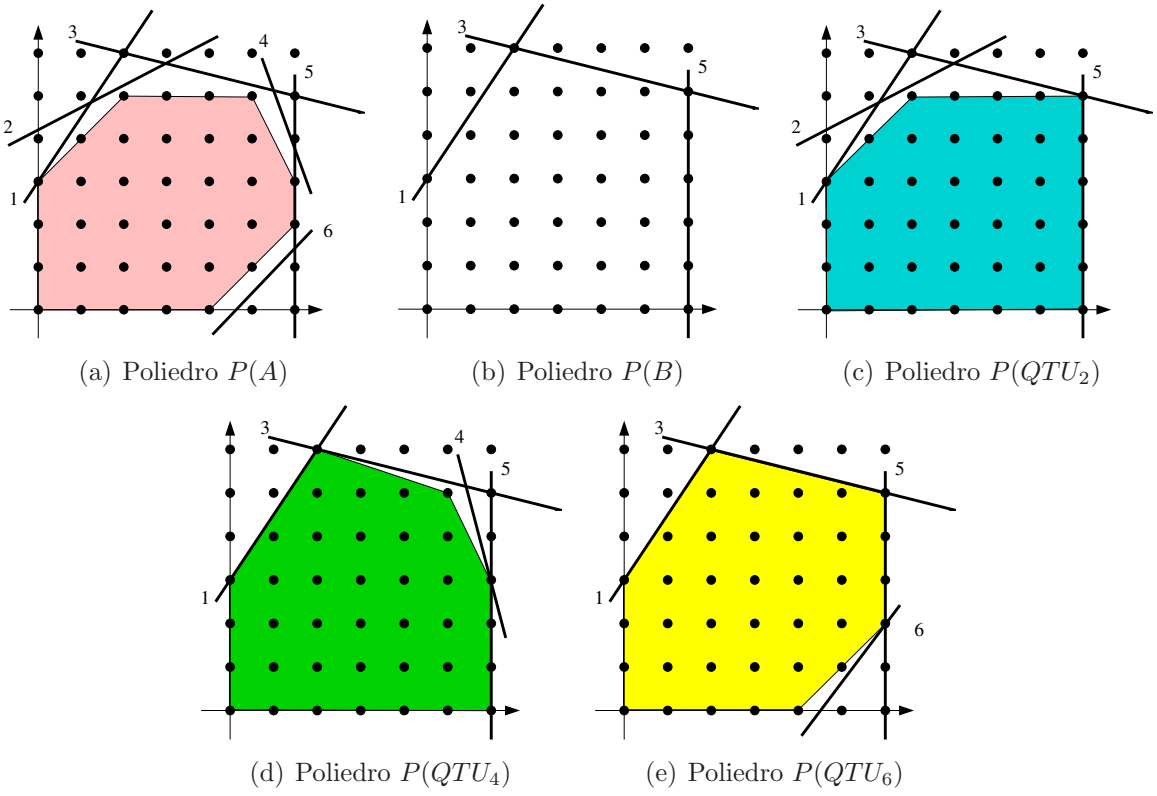
Para ilustrar melhor a idéia do trabalho que foi feito, usaremos o exemplo das figuras 1.1 e 1.2. Note que, nesse exemplo restringimo-nos a um PLI em apenas duas variáveis onde, diferentemente do SPack, as restrições não são necessariamente do tipo empacotamento e as variáveis não são binárias. Contudo, não é difícil perceber que a idéia geral explicada a seguir também pode ser aplicada ao SPack.

O PLI do exemplo das figuras 1.1 e 1.2 é formado por seis desigualdades da forma

${r_i}^T x \leq k_i, i = 1, \ldots, 6$. Considere então as seguintes matrizes:

$$A = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{pmatrix}, B = \begin{pmatrix} r_1 \\ r_3 \\ r_5 \end{pmatrix}, QTU_2 = \begin{pmatrix} r_1 \\ r_3 \\ r_5 \\ r_2 \end{pmatrix}, QTU_4 = \begin{pmatrix} r_1 \\ r_3 \\ r_5 \\ r_4 \end{pmatrix}, QTU_6 = \begin{pmatrix} r_1 \\ r_3 \\ r_5 \\ r_6 \end{pmatrix},$$

sendo B uma matriz TU maximal. Perceba que, para facilitar a analogia, denotamos a matriz de restrição da formulação desse PLI por A. Do mesmo modo, denotaremos por $P(M)$ o poliedro formado por $\{x \in \mathbb{R}^2 \ : \ Mx \leq k, x \geq 0\}$. Essa formulação, assim como $conv(P(A))$, estão representados na figura 1.1(a), sendo que cada face associamos um número que representa o índice da restrição que a suporta.



(a) Poliedro $P(A)$ (b) Poliedro $P(B)$ (c) Poliedro $P(QTU_2)$

(d) Poliedro $P(QTU_4)$ (e) Poliedro $P(QTU_6)$

Figura 1.1: Poliedros e envoltórias convexas.

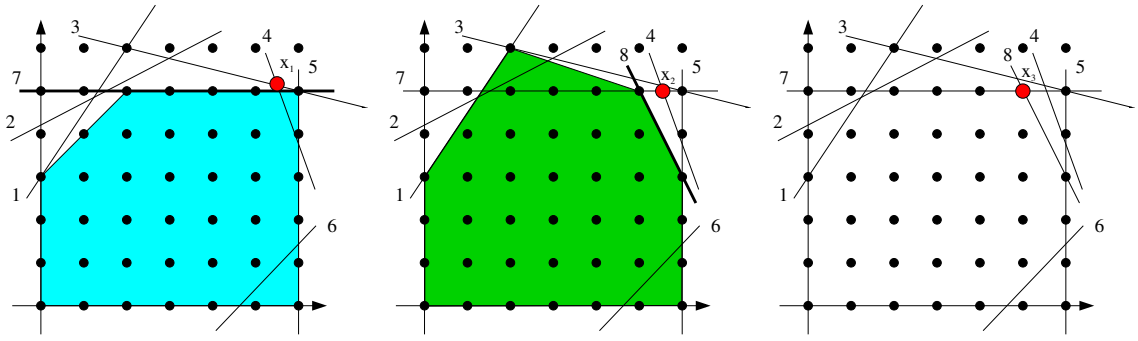
A partir da submatriz TU maximal B, cujo poliedro inteiro associado está representado na figura 1.1(b), obtemos três submatrizes QTU: QTU_2, QTU_4 e QTU_6, cujos poliedros e envoltórias convexas correspondentes estão mostrados nas figuras 1.1(c), 1.1(d) e 1.1(e).

Suponha que o ótimo da relaxação linear de $P(A)$ seja dado pelo ponto x_1 na intersecção entre as faces 3 e 4. Suponha que, ao analisarmos $conv(P(QTU_2))$, descobrimos que existe uma desigualdade válida forte ${r_7}^T \leq k_7$ que separa x_1 desse poliedro, conforme mostrado na figura 1.2(a) e representado pela face de número 7.

Adicionando essa nova desigualdade à formulação corrente e resolvendo a relaxação linear resultante, chega-se a um novo ponto ótimo x_2. Suponha que ao se analisar $conv(P(QTU_4))$ seja descoberta a desigualdade 8, $({r_8}^T x \leq k_8)$, mostrada na figura 1.2(b), que separa x_2 desse poliedro.

Repetindo-se o processo, encontramos o ponto x_3, que é a solução inteira procurada. O poliedro resultante da aplicação desse procedimento chamaremos de $P(A+cortes)$, isto é:

$$P(A + cortes) = P \begin{pmatrix} A \\ r_7 \\ r_8 \end{pmatrix}.$$



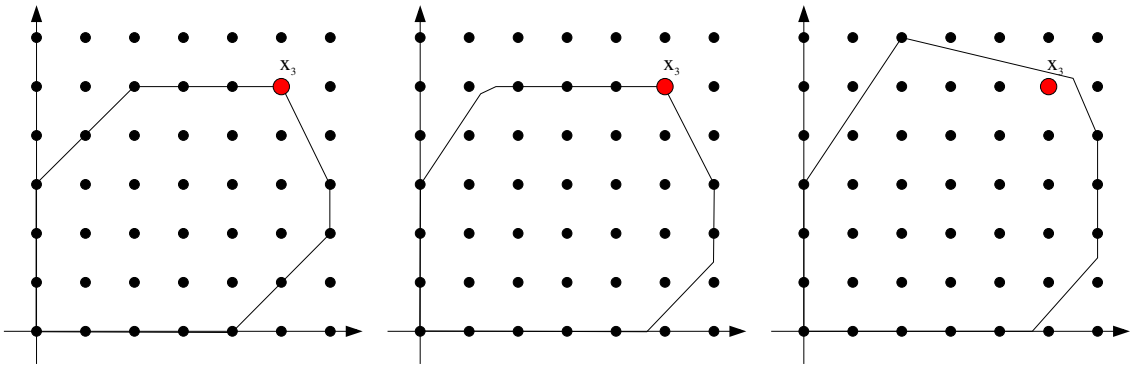
(a) Separando x_1 (b) Separando x_2 (c) Encontrando o ótimo

Figura 1.2: Planos de corte.

Note na figura 1.3 a diferença entre $conv(P(A)), P(A + cortes)$ e $P(A)$. Percebe-se que, apesar de $P(A + cortes)$ ainda não ser igual a $conv(P(A))$, a inserção dos cortes associados às QTUS foi bastante útil para fortalecer a formulação na região próxima ao ponto ótimo. Idealmente, esse é o comportamento que gostaríamos de observar ao aplicarmos um algoritmo de planos de corte para o SPAck onde fossem aplicadas desigualdades válidas fortes para submatrizes QTU.

1.3 Objetivos do Trabalho

Primeiramente, o objetivo deste trabalho foi realizar uma investigação teórica sob a óptica da Combinatória poliédrica visando a obtenção de desigualdades válidas fortes que permi-



(a) $conv(P(A))$ (b) $P(A + cortes)$ (c) $P(A)$

Figura 1.3: Comparação entre $conv(P(A)), P(A + cortes)$ e $P(A)$.

tam uma melhor formulação PLI do SPack quando a matriz de restrições é QTU. Nosso intuito era verificar quais dessas desigualdades válidas para o QTU podem ser utilizadas com freqüência como planos de corte para os SPacks gerais no processo de separação, e, através destes resultados, desenvolver um algoritmo de B&C inserindo tais inequações como cortes. Por último, queríamos avaliar o impacto dos resultados teóricos obtidos por nós no desempenho computacional do algoritmo citado acima, projetado para resolver o problema do empacotamento.

1.4 Organização do Texto

O presente documento está organizado da seguinte forma. No capítulo 2 são dados mais detalhes sobre as bases teóricas do algoritmo discutido anteriormente. No capítulo 3 é feito um estudo da estrutura facial da envoltória convexa das soluções inteiras do problema SPack quando a matriz de restrições é QTU. A partir daí, é justificada a ausência em nossas implementações de algoritmos de separação para algumas desigualdades definidoras de facetas do SPack conhecidas na literatura. Nos casos de outras desigualdades do SPack cuja separação foi implementada, os algoritmos de separação são discutidos em maiores detalhes. O capítulo 4 é dedicado à apresentação de outros algoritmos utilizados, incluindo o algoritmo B&C para o SPack baseado na separação de desigualdades fortes para SPacks definidos sobre matrizes QTUs menores, como já discutido anteriormente. Ainda naquele capítulo, discute-se os resultados computacionais alcançados com nossas implementações. Por fim, no capítulo 5 apresentamos as nossas conclusões e as possíveis direções de pesquisa que podem ser exploradas a partir desta dissertação.

Capítulo 2

Fundamentos Teóricos

Neste capítulo serão apresentados alguns fundamentos teóricos utilizados neste trabalho.

Primeiramente serão mostrados conceitos gerais de Programação Linear e Programação Linear Inteira, e alguns métodos de resolução de PLI utilizando Programação Linear. Em particular, tendo em vista que PLI é NP-difícil, será destacada a importância de termos "boas" formulações para os problemas modelados como programas inteiros. Para tanto, definiremos o que se entende por uma "boa" formulação, o que nos levará a introduzir alguns conceitos básicos de Combinatória Poliédrica relevantes para este trabalho como os de *envoltória convexa* e *lifting*. Nesta contexto teremos a oportunidade de definir o que são as *matrizes de rede puras refletidas* que desempenham um papel central na pesquisa que foi conduzida nesta dissertação.

Também apresentaremos neste capítulo alguns conceitos fundamentais de Teoria dos Grafos. Muitos destes conceitos serão empregados no capítulo seguinte para auxiliar na descrição de desigualdades que podem ser acrescentadas ao modelo PLI do problema do empacotamento – o SPack, que é alvo deste estudo – contribuindo para melhorar a formulação.

2.1 PL e PLI

Para que as idéias presentes neste capítulo possam ser melhor entendidas, serão dadas ao longo do mesmo algumas definições, corolários e teoremas. Tais conceitos, e outros mais, podem ser encontrados em [24].

Definição 2.1.1 *Um poliedro $P \subseteq \mathbb{R}^n$ é o conjunto de pontos que satisfazem um número finito de inequações lineares, ou seja, $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ é um poliedro, com $A \in \mathbb{R}^m \times \mathbb{R}^n$ e $b \in \mathbb{R}^m$. Um poliedro é dito ser racional se A e b possuem coeficientes racionais.*

Definição 2.1.2 *Um poliedro $P \subseteq \mathbb{R}^n$ é limitado se existe um $\omega \in \mathbb{R}_+$, tal que $P \subseteq \{x \in \mathbb{R}^n : -\omega \leq x_j \leq \omega \text{ para } j = 1, \ldots, n\}$. Um poliedro limitado é chamado de politopo.*

Note que sempre podemos supor, sem perda de generalidade, que o poliedro P é dado por um sistema linear como aquele descrito na definição 2.1.1. Isso ocorre porque sempre é possível reescrever $A'x \geq b'$ como $Ax \leq b$, fazendo-se $A = -A'$ e $b = -b'$. Também podemos transformar um sistema linear com igualdades neste mesmo formato, pois $A'x = b' \Rightarrow A'x \leq b'$ e $A'x \geq b'$, ou seja, pode ser escrito como $Ax \leq b$ com $A = \begin{pmatrix} A' \\ -A' \end{pmatrix}$ e $b = \begin{pmatrix} b' \\ -b' \end{pmatrix}$.

O Problema de Programação Linear (PL) lida com a maximização (*max*) ou minimização (*min*) de uma função linear $z(x)$, chamada de função objetivo (FO), tal que sua solução x^* deve estar contida em um poliedro P.

Como max $z(x)$ é equivalente a min $-z(x)$, para simplificar, todas as vezes que nos referirmos aos PLs, estaremos considerando max $z(x)$, com $x \in P$, a menos que outro formato seja explicitado.

Resumidamente, um PL é apresentado da seguinte forma:

$$\begin{array}{ll} \max & z(x) = c^T x \\ \text{s.a.} & Ax = b \\ & x \geq 0 \end{array} \quad , \tag{2.1}$$

sendo que a sigla "s.a." significa *"sujeito a"*, e o vetor $c \in \mathbb{R}^n$ é chamado de vetor de custos.

Esta é chamada de representação padrão de um PL. As restrições são apresentadas na forma de igualdades, acrescentando-se variáveis de folgas quando necessário, e considera-se as variáveis não negativas, fazendo substituição de variáveis quando for preciso. Alguns exemplos onde tais técnicas devem ser usadas são dados abaixo.

Seja $a \in \mathbb{R}^m$

$$a^T x \leq b \quad \rightsquigarrow \quad \begin{array}{l} a^T x + x' = b \\ x' \geq 0 \end{array} \tag{2.2}$$

$$a^T x \geq b \quad \rightsquigarrow \quad \begin{array}{l} a^T x - x' = b \\ x' \geq 0 \end{array} \tag{2.3}$$

Seja $k_i \in \mathbb{R}, i = 1, 2, 3, 4$, faremos a substituição de x' por $y_1 - y_2$

$$\begin{array}{lll} \max & c^T x + k_1 x' & \\ \text{s.a.} & a^T x + k_2 x' \leq b & \\ & x \geq 0 & \rightsquigarrow \\ & k_3 \leq x' \leq k_4 & \end{array} \qquad \begin{array}{lll} \max & c^T x + k_1 y_1 - k_1 y_2 & \\ \text{s.a.} & a^T x + k_2 y_1 - k_2 y_2 \leq b & \\ & x, y_1, y_2 \geq 0 & \\ & y_1 - y_2 \geq k_3 & \\ & y_1 - y_2 \leq k_4 & \end{array} \tag{2.4}$$

A cada formulação associamos um poliedro P formado pelas restrições nela presentes. Os coeficientes das variáveis compõem a matriz A denominada matriz de restrições.

Chamamos de solução viável os pontos $x \in S$, tal que o conjunto S é chamado de região viável sendo que, para um PL, $S = P$.

O valor de uma solução viável x é dada por $z(x)$. Uma solução viável cujo valor seja o maior possível é chamada de solução ótima (x^*) e seu valor de valor ótimo, denotado por $z^* = z(x^*)$.

Um problema de PL é dito ser bem definido quando ele é viável ($P \neq \emptyset$), e seu valor ótimo não é ilimitado, em outras palavras, que ele possui uma solução ótima.

Problemas deste tipo podem ser resolvidos em tempo polinomial, através de algoritmos como os de *pontos interiores.* No entanto, um dos algoritmos mais utilizados é o `SIMPLEX`, que apesar de ter complexidade exponencial, mostra-se bastante rápido na prática. Algumas idéias fundamentais do `SIMPLEX` são comentadas a seguir, o que nos leva a apresentar conceitos de Álgebra Linear que serão úteis nas discussões dos capitulos seguintes.

Definição 2.1.3 *Um conjunto de pontos $x^1, \ldots, x^k \in \mathbb{R}^n$ é linearmente independente (l.i.) se a única solução de $\sum_{i=1}^k \lambda_i x^i = 0$ é $\lambda_i = 0, i = 1, \ldots, k$.*

Definição 2.1.4 *Um conjunto de pontos $x^1, \ldots, x^k \in \mathbb{R}^n$ é afim independente (a.i.) se a única solução de $\sum_{i=1}^k a_i x^i = 0, \sum_{i=1}^k \alpha_i = 0$ é $\alpha_i = 0$ para $i = 1, \ldots, k$.*

Definição 2.1.5 *Seja A uma matriz de dimensão $m \times n$. O posto de A, denotado por posto(A), é o número máximo de linhas ou colunas linearmente independentes de A. Dizemos que A tem posto completo (ou cheio) se posto(A) $= m$ ou posto(A) $= n$.*

Os conceitos acima serão usados para mostrar como podemos encontrar soluções de sistemas lineares onde o número de equações é menor que o número de incógnitas. Uma vez que as restrições de um PL na forma padrão, excluída a não-negatividade das variáveis, nada mais é do que um sistema linear, este método pode ser aplicado para encontrar soluções viáveis de um PL.

Seja $A = (a_1, a_2, \ldots, a_n)$ uma matriz $m \times n$, onde a_j é a j-ésima coluna de A. Suponha que $m \leq n$ e posto(A) $= m$. Como o posto de A é completo, existe uma submatriz de A de dimensão $m \times m$ da forma $A_B = (A_{B_1}, \ldots, A_{B_n})$, com $B_i \in \{1, \ldots, n\}$, que é não-singular.

Seja agora $B = (B_1, \ldots, B_m)$ e seja $N = (1, \ldots, n) \setminus B$. Permutemos as colunas de A de modo que $A = (A_B, A_N)$. Assim, podemos escrever $Ax = b$ como $A_B x_B + A_N x_N = b$, onde $x = (x_B, x_N)$. Deste modo, uma solução para $Ax = b$ é dada por $x_B = A_B^{-1} b$ e $x_N = 0$.

Definição 2.1.6 *Considere as matrizes A, A_B e A_N definida como acima. Então temos:*

- *A matriz $m \times m$ não-singular A_B é chamada de base.*
- *A solução $x_B = A_B^{-1} b, x_N = 0$ é chamada de solução básica de $Ax = b$ associada a A_B.*
- *x_B é o vetor de variáveis básicas, e x_N é o vetor de variáveis não-básicas.*
- *Se $A_B^{-1} b \geq 0$, então (x_B, x_N) é chamada de solução primal básica viável e A_B é chamada de base primal viável.*

O `SIMPLEX` se baseia na idéia de que o conjunto de todas as soluções viáveis de um PL é um poliedro e que, se o valor ótimo da função objetivo for finito, necessariamente haverá um ponto extremo (vértice) deste poliedro que será uma solução ótima. Assim, resumidamente, o que o algoritmo faz é percorrer os pontos extremos da região viável (não necessariamente todos) até encontrar o ótimo. Computacionalmente, a exploração de pontos extremos só é possível pois estes podem ser representados algebricamente. É neste momento que necessitamos da definição de solução básica vista acima.

Definição 2.1.7 *x é um ponto extremo de um conjunto Q se não existem $x^1, x^2 \in Q, x^1 \neq x^2$, tal que $x = \frac{1}{2}x^1 + \frac{1}{2}x^2$.*

Teorema 2.1.1 *Dado um poliedro P, um ponto $x \in P$ é um ponto extremo se e somente se ele é uma solução básica viável do PL.*

O corolário a seguir garante que o algoritmo `SIMPLEX` termina em um número finito de passos, desde que seja assegurado que ele não irá visitar mais de uma vez um mesmo ponto extremo. A validade do resultado decorre do fato de que o número de soluções básicas de um sistema linear com m equações e n variáveis, $m \leq n$, é dado por $\binom{m+n}{n}$.

Corolario 2.1.1 *Um poliedro possui um número finito de pontos extremos.*

Embora muitos outros detalhes fossem necessários para descrevermos o algoritmo `SIMPLEX` em sua plenitude, vamos nos limitar aos resultados fundamentais dados acima. Esta escolha em não prosseguir com esta discussão deveu-se a dois fatores. Primeiramente, o algoritmo `SIMPLEX` é tema de vários excelentes livros-texto sobre PL, sendo a grande

maioria de fácil acesso. Além disso, os resultados anteriores são aqueles mais relacionados ao trabalho desta dissertação.

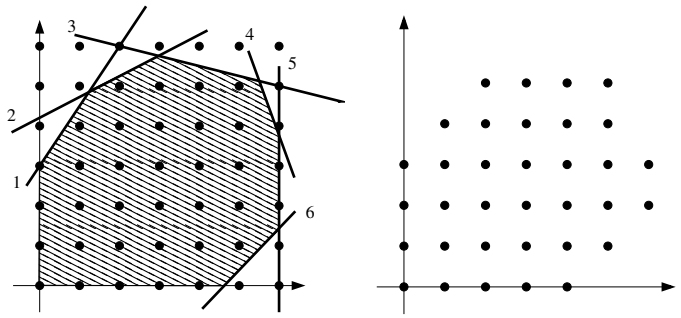
A seguir, passamos a uma breve discussão sobre Programação Linear Inteira. A grosso modo, podemos pensar em um PLI como sendo um problema de PL ao qual se adiciona a restrição $x \in \mathbb{Z}^n$ e, portanto, sua região viável S é dada por $S = P \cap \mathbb{Z}^n$. Note-se que, a rigor, em um PLI podemos ter todas ou algumas variáveis assumindo valores inteiros. No contexto desta dissertação, vamos supor que a integralidade será imposta a todo o conjunto de variáveis. Ao longo do texto, a menos que outro formato seja explicitado, todas as vezes que nos referirmos aos PLIs, estaremos considerando um problema da forma $\max z(x)$, com $x \in P \cap \mathbb{Z}^n$ e $P = \{x \in \mathbb{R}^n : Ax \leq B\}$.

Problemas de PLI são encontrados em diversas aplicações práticas e, em geral, são difíceis de serem resolvidos, pois pertencem à classe *NP-difícil*. Os algoritmos utilizados para a resolução dos PLIs têm sua eficiência fortemente dependente de dois limitantes chamados de *primal* e *dual* e que são atualizados ao longo das iterações. O limitante primal, cujo valor é sempre menor ou igual ao valor ótimo z^* do PLI, é dado, em geral, pelo custo de uma solução viável do problema, não necessariamente ótima. Já o limitante dual é sempre maior ou igual ao valor de z^* e, normalmente, é computado resolvendo-se alguma *relaxação* do problema original. Antes de passarmos à discussão sobre possíveis relaxações a serem empregadas no caso de PLI, observamos que, como dito acima, quando queremos maximizar a função objetivo, o limitante primal é um limitante inferior e o dual um limitante superior para o valor ótimo. Convém salientar entretanto que, de modo análogo, se tivermos um problema de minimização, os limitantes primal e o dual são os limitantes superior e inferior do valor ótimo, respectivamente. Dito isto, passemos às relaxações.

2.1.1 Relaxações e formulações PLIs alternativas

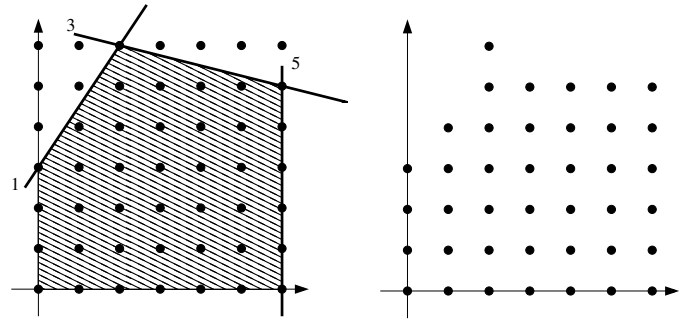
Uma *relaxação de um problema* é obtida sempre que removemos algumas de suas restrições. Usualmente, isto deve tornar sua resolução mais simples. Porém, é claro que, a remoção de restrições de um problema tende a aumentar o conjunto de soluções viáveis do mesmo. Assim, para ilustrar a idéia do uso de relaxações na obtenção de limitantes duais, suponha que temos um PLI (problema original) com soluções dadas por um conjunto $S \in \mathbb{Z}^n$ e seja P um poliedro do $\mathbb{R}^N$ tal que $S = P \cap \mathbb{Z}^n$. Considere um novo PLI, cujas soluções compõem o conjunto S' de $\mathbb{Z}^n$ de modo que $S' = P' \cap \mathbb{Z}^n$ e P' é obtido de P removendo-se algumas das desigualdades lineares que fazem parte da descrição de P. Um exemplo deste procedimento é mostrado na figura 2.1, onde P' é obtido de P retirando-se as restrições identificadas pelas retas 2, 4 e 6. Note pelas figuras 2.1(b) e 2.1(d) que S' contém S, resultando em toda solução do PLI original ser solução do novo PLI (o relaxado). Isso

faz com que o valor ótimo do PLI relaxado seja no mínimo tão bom quanto àquele do PLI original. Se, por hipótese, o PLI relaxado for fácil de ser resolvido, seremos capazes de gerar rapidamente um limitante dual para o PLI original. É justamente a capacidade de gerarmos bons limitantes duais e primais em um baixo tempo computacional que irá determinar o sucesso de um algoritmo exato para PLI. Daí a importância de escolhermos boas relaxações.



(a) Exemplo de P obtido através das restrições de um PLI.

(b) $S = P \cap \mathbb{Z}$.



(c) P' obtido através da relaxação de P.

(d) $S' = P' \cap \mathbb{Z}$.

Figura 2.1: Exemplos de relaxações.

Uma das relaxações mais comuns para PLI é a chamada *relaxação linear*. Esta relaxação corresponde ao PL obtido ao removermos do PLI original as restrições de integralidade das variáveis. É fácil verificarmos que se o conjunto de soluções de um PLI (com n variáveis) é dado por $P \in \mathbb{Z}^n$, então a relaxação linear deste PLI corresponde ao PL com mesma FO e com a região de viabilidade dada pelo poliedro P.

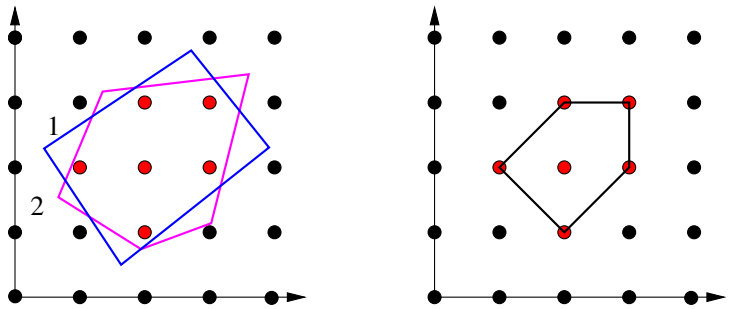
Como o conceito de relaxação linear será bastante utilizado mais adiante, denotaremos um problema inteiro como IP e sua relaxação linear por LP. Esta nomenclatura será empregada ao nos referenciarmos sobre valores e/ou propriedades inerentes a estes problemas. Por exemplo, denotaremos por z^*_{IP} e z^*_{LP} os valores ótimos dos problemas inteiro

e de sua relaxação linear, respectivamente.

A principal vantagem da relaxação linear é que ela é resolvível em tempo polinomial e pode ser aplicada a qualquer PLI sem necessidade de nenhuma informação específica sobre o problema sendo tratado, ou seja, pode ser feita facilmente de maneira automática.

Via de regra, um algoritmo para PLI irá gerar limitantes primais e duais ao longo de suas iterações. Vimos acima que estes últimos podem ser calculados através da resolução de relaxações lineares e, como mencionamos, os primeiros podem ser computados a partir de quaisquer soluções conhecidas para o PLI original. A eficiência do algoritmo será tanto maior quanto mais rápido estes limitantes convergirem para o valor ótimo do PLI. Tipicamente, limitantes duais de qualidade são mais difíceis de serem obtidos que limitantes primais. Para gerá-los via relaxações lineares precisamos encontrar "boas" formualções para o PLI em questão.

Para entendermos melhor a discussão iniciada no parágrafo anterior, considere um PLI definido sobre n variáveis cujo conjunto de soluções é dado por $S \subseteq \mathbb{Z}^n$. Um poliedro (ou seja, um conjunto de soluções de um sistema de desigualdades lineares) $P \in \mathbb{R}^N$ é uma *formulação* para este PLI se e somente se $S = P \cap \mathbb{Z}^n$. Note que, por esta definição, existem infinitas formulações para um PLI. Tal situação encontra-se exemplificada na figura 2.2. Percebe-se que alguns poliedros correspondentes à formulações do PLI apresentam relações de continência entre si, porém, isso nem sempre é verdade (ver figura 2.2(a)). Mesmo assim, o que é certo é que qualquer poliedro P associado a uma formulação de um PLI cujo conjunto de soluções é S deve ser tal que $conv(S) \subseteq P$, onde $conv(S)$ pode ser entendido como sendo o menor poliedro contendo os pontos de S e é chamado de *envoltória convexa* de S. Na figura 2.2(b) é mostrada a envoltória convexa das soluções do PLI formulado pelos poliedros da figura 2.2(a). As formulações de tais poliedros são dadas na figura 2.3.



(a) Exemplo de Poliedros

(b) Exemplo de Envoltória Convexa

Figura 2.2: Exemplos de poliedros diferentes para o mesmo problema.

Formalmente, a envoltória convexa de um conjunto S pode ser definida da seguinte maneira:

$$\begin{array}{rcrcl} 4x_1 & + & 5x_2 & \leq & 29 \\ 3x_1 & - & 2x_2 & \leq & 6 \\ 2x_1 & + & 3x_2 & \geq & 6 \\ -5x_1 & + & 4x_2 & \leq & 4 \end{array}$$

(a) Exemplo de formulação para o poliedro 1 da figura 2.2(a)

$$\begin{array}{rcrcl} 3x_1 & - & 7x_2 & \leq & 0 \\ 8x_1 & - & x_2 & \leq & 24 \\ -x_1 & + & 4x_2 & \leq & 11 \\ 8x_1 & - & 3x_2 & \geq & 0 \\ 3x_1 & + & 2x_2 & \geq & 6 \end{array}$$

(b) Exemplo de formulação para o poliedro 2 da figura 2.2(a)

$$\begin{array}{rcrcl} x_1 & & & \leq & 3 \\ & & x_2 & \leq & 3 \\ -x_1 & + & x_2 & \leq & 1 \\ x_1 & + & x_2 & \geq & 3 \\ x_1 & - & x_2 & \leq & 1 \end{array}$$

(c) Exemplo de formulação para o poliedro da figura 2.2(b)

Figura 2.3: Formulação para os poliedros da figura 2.2

Definição 2.1.8 *Dado um conjunto $S \subseteq \mathbb{R}^n$, um ponto $x \in \mathbb{R}^n$ é uma combinação convexa dos pontos de S se existe um conjunto finito de pontos $\{x_i\}_{i=1}^t$ em S e um $\lambda \in \mathbb{R}_+^t$ com $\sum_{i=1}^t \lambda_i = 1$ e $x = \sum_{i=1}^t \lambda_i x_i$. A envoltória convexa de S, denotada por $conv(S)$, é o conjunto de todos os pontos que são combinações convexas dos pontos em S.*

Em geral existem várias escolhas de A e b para um poliedro $P = \{x \in \mathbb{R}^n : Ax \leq b\}$, tal que $P \cap \mathbb{Z}^n = S$. Pode ser bem simples encontrar algum par (A, b) que leve a um poliedro P com a propriedade desejada. Porém, tal escolha tem grande influência no tempo de convergência do algoritmo para a resolução de um PLI pois, essencialmente, o valor ótimo do PL definido sobre P é que será usado como limitante dual pelo algoritmo. Assim, nem sempre a escolha mais óbvia é a melhor.

As definições que são apresentadas a seguir serão úteis para definirmos, pelo menos do ponto de vista matemático, o que se entende por uma "boa" formulação. Nelas, iremos supor que P é um poliedro no $\mathbb{R}_+^n$.

Definição 2.1.9 *A inequação $\pi x \leq \pi_0$ ou (π, π_0) é uma* desigualdade válida *para P se ela é satisfeita por todos os pontos de P.*

Na literatura especializada, ao se estudar um determinado poliedro, é comum nomear as desigualdades válidas observando-se quais variáveis têm coeficientes não-nulos na inequação linear. Assim, vamos aproveitar este ponto para definirmos abaixo o que vem a ser o *suporte* de uma desigualdade.

Definição 2.1.10 *Para $x \in \mathbb{R}^n$, o suporte de uma desigualdade $\pi x \leq \pi_0$ é dado pelo conjunto de índices $\{j \in \{1, \ldots, n\} \ : \ \pi_j \neq 0\}$.*

As desigualdades válidas de um poliedro P estão associadas a subconjuntos do poliedro que, como será visto adiante, são de grande relevância para o PLI.

Definição 2.1.11 *Se (π, π_0) é uma desigualdade válida para P e $F = \{x \in P : \pi x = \pi_0\}$, diz-se que F é uma* face *de P, e que (π, π_0)* representa *F. Uma face é dita ser* própria *para P se $F \neq \emptyset$ e $F \neq P$.*

Definição 2.1.12 *A face F de P representada por (π, π_0) é dita ser não vazia se e somente se* $\max\{\pi x : x \in P\} = \pi_0$. *Quando F é não vazia, dizemos que* (π, π_0) suporta P.

Definição 2.1.13 *Dado P, dizemos que as desigualdades válidas $\pi x \leq \pi_0$ e $\gamma x \leq \gamma_0$ são equivalentes se $(\gamma, \gamma_0) = \lambda(\pi, \pi_0)$ para algum $\lambda > 0$. Se elas não são equivalentes e existe $\mu > 0$ tal que $\gamma \geq \mu\pi$ e $\gamma_0 \leq \mu\pi_0$, então $\{x \in \mathbb{R}^n_+ : \gamma x \leq \gamma_0\} \subset \{x \in \mathbb{R}^n_+ : \pi x \leq \pi_0\}$. Neste caso dizemos que $\gamma x \leq \gamma_0$ domina, ou é mais forte que $\pi x \leq \pi_0$ ou que $\pi x \leq \pi_0$ é dominada por ou é mais fraca que $\gamma x \leq \gamma_0$.*

Definição 2.1.14 *Dizemos que uma desigualdade válida é* maximal *se ela não é dominada por nenhuma outra desigualdade válida. Qualquer desigualdade válida maximal de S define uma face não-vazia de $conv(S)$.*

Neste sentido, entende-se que uma formulação de um PLI com soluções dadas pelo conjunto S é melhor quanto mais fortes forem suas desigualdades. Por outro lado, quanto mais fortes forem as desigualdades de uma formulação, mais próximo o seu poliedro correspondente estará da envoltória convexa de S. Como $conv(S)$ é o menor poliedro que contém S, ao aplicarmos a relaxação linear, quanto melhor a formulação, melhor será o limitante dual e, conseqüentemente, mais rápido o algoritmo tenderá a convergir.

Deve-se observar que se a solução da relaxação linear for um ponto inteiro, então, teremos encontrado uma solução ótima para o problema inteiro. Isso ocorre porque este ponto inteiro certamente está em S (caso contrário P não seria uma formulação para S) e, portanto, o seu custo fornece um limitante primal cujo valor é o próprio limitante dual dado pelo valor ótimo da relaxação. Além disso, como a relaxação linear é um PL, ela pode ser resolvida em tempo polinomial. Deste modo, numa situação ideal, teríamos resolvido o PLI em tempo polinomial. Um exemplo onde tal situação acontece é quando a relaxação linear corresponde à própria envoltória convexa de S.

A situação descrita ao final do parágrafo anterior é de interesse especial para esta dissertação. Mais especificamente, queremos tirar proveito de formulações que sabidamente correspondem a poliedros tais que todos os seus pontos extremos (vértices) têm coordenadas inteiras. Este é precisamente o tema da seção que se segue.

2.1.2 Poliedros inteiros

Inicialmente definimos o que vem a ser um *poliedro inteiro*.

Definição 2.1.15 *Um poliedro $P \subseteq \mathbb{R}^n$ não vazio é dito ser inteiro se e somente se todos seus pontos extremos são inteiros.*

A proposição a seguir provê uma caracterização útil para os poliedros inteiros.

Proposição 2.1.1 *Seja P um poliedro e c um vetor arbitrário no $\mathbb{R}^n$. Considere o PL dado por $z = \max\{cx : x \in P\}$. As seguintes afirmações são equivalentes:*

1. *P é inteiro.*
2. *o PL possui uma solução ótima inteira para qualquer $c \in \mathbb{R}^n$, para o qual ele tenha uma solução ótima.*
3. *o valor ótimo do PL, ou seja z, é inteiro para todo $c \in \mathbb{Z}^n$ para o qual ele tenha uma solução ótima.*

Note que pelas definições 2.1.8 e 2.1.15, temos que a envoltória convexa de um PLI é um poliedro inteiro. Portanto, se este for o caso, ao resolvermos a relaxação linear deste problema, o valor encontrado já será o valor do ótimo do problema original. Se utilizarmos algoritmos como o `SIMPLEX`, cujas soluções encontradas são sempre pontos extremos, já teremos inclusive o ponto ótimo procurado. Este é o motivo de tantos esforços serem empregados no intuito de se encontrar formulações cada vez melhores, ou seja, que mais se aproximam da envoltória convexa.

A questão que se coloca agora é se existe alguma característica de uma formulação PLI que permita dizer que o poliedro correspondente à sua relaxação linear é inteiro. Uma forma possível de se fazer isso é examinar a matriz de restrições do problema. Mais especificamente, se $P = \{x \in \mathbb{R}^n : Ax \leq b\}$, queremos identificar propriedades da matriz A que, em sendo verificadas, garantam que P seja inteiro. De fato, existem características estruturais que fazem com que algumas famílias de matrizes estejam associadas a poliedros inteiros. Dentre estas famílias, a mais conhecida é aquela composta pelas *matrizes totalmente unimodulares (TU)*.

A seguir introduzimos alguns conceitos importantes sobre matrizes TU que serão empregados neste trabalho. Novamente, nossa intenção não é esgotar o assunto e o leitor interessado em se aprofundar mais no tema deve recorrer, por exemplo, a [24].

Definição 2.1.16 *Uma matriz $m \times n$ inteira A é totalmente unimodular se o determinante de qualquer submatriz quadrada de A pertence a $\{-1, 0, 1\}$.*

Obviamente, esta definição implica que em qualquer matriz TU todos elementos devem estar no conjunto $\{-1, 0, 1\}$. A seguinte proposição também segue diretamente da definição 2.1.16.

Proposição 2.1.2 *As seguintes afirmações são equivalentes.*

1. *A é TU.*
2. *A^T é TU.*
3. *$(A|I)$ é TU.*
4. *A matriz obtida retirando-se uma linha ou coluna de A é TU.*
5. *A matriz obtida multiplicando-se uma linha ou coluna de A por -1 é TU.*
6. *A matriz obtida permutando-se duas linhas ou duas colunas de A é TU.*
7. *A matriz obtida duplicando-se uma linha ou coluna de A é TU.*

Outros três resultados interessantes sobre matrizes TU são dados logo a seguir.

Proposição 2.1.3 *Se A é TU, então $P(b) = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$ é inteiro para todo $b \in \mathbb{Z}^m$ para o qual ele não é vazio.*

Um argumento similar nos leva a generalização da proposição 2.1.3.

Proposição 2.1.4 *Se A é TU, b, b', d, d' são inteiros e $P(b, b', d, d') = \{x \in \mathbb{R}^n : b' \leq Ax \leq b, d' \leq x \leq d\}$ é não vazio, então $P(b, b', d, d')$ é um poliedro inteiro.*

Teorema 2.1.2 *Se $P(b) = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$ é inteiro para todo $b \in \mathbb{Z}^m$ para o qual ele não é vazio, então A é TU.*

Note que o teorema 2.1.2 é falso para $P(b) = \{x \in \mathbb{R}^n_+ : Ax = b\}$.

Um subconjunto das matrizes TU de importância capital para esta dissertação é formado pelas chamadas *matrizes de rede pura refletida* (MRPR).

Definição 2.1.17 *Uma matriz A é uma matriz de Rede Pura (MRP) se todos os elementos de A pertencem ao conjunto $\{-1, 0, 1\}$, e cada coluna de A possui no máximo um elemento igual a 1 e um elemento igual a -1.*

Definição 2.1.18 *A reflexão de uma linha da matriz A muda o sinal de todos os elementos desta linha. A matriz A é dita ser uma Matriz de Rede Pura Refletida (*MRPR*) se existe uma seqüência de reflexões de linhas que transformem A em uma MRP.*

Proposição 2.1.5 *Se a matriz A com elementos em $\{-1, 0, 1\}$ não possui mais do que dois elementos não-nulos em cada coluna e $\sum_i a_{ij} = 0$ para toda coluna j com dois coeficientes não-nulos, então A é TU.*

Pela proposição 2.1.5, vemos que as MRPRs são matrizes TUs.

Veremos neste trabalho que as matrizes TU e as MRPRs podem ser empregadas na solução de PLIs mesmo quando ocorrem como submatrizes próprias da matriz de restrições. Identificar grandes submatrizes da matriz de restrições nestas famílias pode ser de enorme valia em métodos de decomposição como é o caso ao usarmos *relaxações lagrangeanas* (cf., [24]). De uma maneira simplificada, também podemos pensar em gerar uma relaxação de um PLI removendo-se do seu conjunto de restrições todas as desigualdades que não correspondam a linhas de uma submatriz TU ou MRPR de sua matriz de restrições, desde que esta última contenha todas as colunas da matriz original. Repare que se isso for feito, um limitante dual pode ser computado resolvendo-se o PL restante, portanto, em tempo polinomial. Voltaremos a esta discussão nos próximos capítulos e apenas vamos terminar aqui já antecipando que o problema de encontrar a maior submatriz da matriz de restrições de um PLI que forme um poliedro inteiro também pertence à classe de problemas NP-difíceis.

Na próxima seção prosseguimos com o caso dos PLIs cuja formulação não corresponde a um poliedro inteiro.

2.1.3 Teoria Poliedral

Até agora vimos que a formulação de um PLI com o conjunto de soluções dado por S será considerada "boa"'(ou forte) quanto mais próximo da envoltória convexa de S estiver o conjunto de soluções da relaxação linear da formulação. Idealmente, este conjunto corresponde à própria envoltória convexa, como é o caso dos poliedros inteiros. Porém, na maior parte dos casos, esta situação ideal não se verifica. Mesmo quando isso ocorre, precisamos ser capazes de identificar quais desigualdades se aproximam da descrição desta envoltória convexa. É neste contexto que empregamos os conceitos da Teoria Poliedral que são sumarizados a seguir e que podem ser encontrados em [24].

Inicialmente precisamos definir o que entendemos por *dimensão* de um conjunto.

Definição 2.1.19 *Dado um conjunto $Q \in \mathbb{R}^n$, a dimensão de Q será igual a k, denotado por $dim(Q) = k$, se o número máximo de pontos afim independentes em Q é $k + 1$.*

Agora considere o seguinte poliedro no $\mathbb{R}^n$: $P = \{x \in \mathbb{R}^n : A^{=}x = b^{=}, A^{\leq}x \leq b^{\leq}\}$. Vamos supor que, no total, existam m restrições no sistema linear que descreve P e que elas sejam indexadas pelo conjunto $M = \{1, 2, \ldots, m\}$, sendo $M^{=}$ o subconjunto dos índices de M associados a restrições de igualdade e $M^{\leq} = M \setminus M^{=}$, ou seja, $M^{\leq}$ são os índices das restrições correspondentes às desigualdades. O seguinte resultado permite obter a dimensão de P a partir do sistema linear que o descreve.

Proposição 2.1.6 *Seja $P \subseteq \mathbb{R}^n$ descrito pelo sistema linear como acima. Então $dim(P)+$ $posto(A^=, b^=) = n$, onde $(A^=, b^=)$ denota a matriz obtida ao acrescentar o vetor coluna $b^=$ à matriz $A^=$.*

Veremos mais adiante que o conhecimento da dimensão de um poliedro P é fundamental para podermos estabelecer a força de uma desigualdade relativamente a P. Também será de grande valia para esta dissertação a definição a seguir.

Definição 2.1.20 *Um poliedro $P \subseteq \mathbb{R}^n$ possui dimensão cheia se $dim(P) = n$.*

Neste documento iremos trabalhar sempre com P racional e supor que a matriz (A, b) possui posto completo, ou seja, que todas as restrições redundantes tenham sido eliminadas previamente. Além disso, como os problemas que estamos tratando vieram de situações práticas, iremos supor que o poliedro possua alguma solução ótima (i.e., não seja inviável e seja limitado na direção do gradiente da FO).

Considere novamente o poliedro dado por $P = \{x \in \mathbb{R}^n : A^=x = b^=, A^{\leq}x \leq b^{\leq}\}$ como anteriormente. Como estamos sob a hipótese de que não existem restrições redundantes no sistema linear que descreve P, queremos ser capazes de caracterizar quais desigualdes são necessárias e suficientes para descrever P (note que, por hipótese, as equações são linearmente independentes entre si e, por isso, podemos nos restringir a analisar as inequações). Neste momento revela-se a importância do conceito de dimensão.

Para P dado no parágrafo anterior, seja $\pi x \leq \pi_0$ uma desigualdade válida para P. A face definida por esta desigualdade em P, a qual denotaremos por $F(\pi, \pi_0)$, é dada pelo conjunto de pontos em $\{x \in P : \pi x = \pi_0\}$.

Toda face de P é um poliedro e é representada por alguma desigualdade válida para este poliedro. Na verdade, pode-se mostrar que, para uma face qualquer F de P, o sistema linear que a descreve é obtido do mesmo sistema que descreve P apenas transformando-se em igualdades um subconjunto das restrições indexadas por $M^{\leq}$. Assim, se aplicarmos a proposição 2.1.6 ao poliedro associado a uma face, vemos que quanto maior a quantidade de desigualdades do sistema linear original que são transformadas em igualdades, menor será a dimensão da face. Abaixo caracterizamos as faces próprias de maior dimensão que podem ser obtidas para o poliedro P.

Definição 2.1.21 *Uma face F de P é dita ser uma faceta de P se $dim(F) = dim(P)-1$.*

Conforme discussão anterior, o seguinte resultado vale para facetas.

Proposição 2.1.7 *Se F é uma faceta de P, então existe alguma inequação $a^k x \leq b_k$ para $k \in M^{\leq}$, representando F.*

O resultado abaixo mostra que, dada uma faceta F de P, qualquer sistema linear descrevendo P deve necessariamente conter uma desigualdade que represente F.

Proposição 2.1.8 *Para cada faceta F de P, alguma inequação representando F é necessária para a descrição de P.*

A importância das facetas vai além do que foi mostrado até aqui. O próximo resultado mostra que elas não são apenas necessárias para a descrição de P mas também são suficientes.

Proposição 2.1.9 *Toda inequação $a^r x \leq b_r$ para $r \in M^{\leq}$ que representa a face de P de dimensão menor que $dim(P) - 1$ é irrelevante para a descrição de P.*

Vejamos agora qual a importância dos resultados acima para a Programação Linear Inteira. Novamente consideramos um PLI com conjunto de soluções dado por S e envoltória convexa dada por $P = conv(S)$. Não é difícil perceber que podemos computar a dimensão de P mesmo sem conhecer uma descrição linear para este poliedro. Com isso, se conhecermos uma desigualdade válida para S (e, consequentemente, para $P = conv(S)$), também podemos computar a dimensão da face definida por essa desigualdade em P e, eventualmente, concluir que ela representa (ou define) uma faceta de P ou, pelo menos, uma face de dimensão bastante alta neste poliedro. Com isso, temos um método para determinar quão "boa" ou "forte" é esta desigualdade e a nossa formulação como um todo.

Neste contexto, a prova de que uma determinada desigualdade define uma faceta de P é, em geral, tanto mais difícil quanto menor a dimensão do poliedro. Isso ocorre porque quando o poliedro não tem dimensão cheia, uma mesma faceta pode ser representada por uma infinidade de desigualdades lineares diferentes. Contudo, conforme o resultado expresso no teorema a seguir, se P tem dimensão cheia, esta representação é, de certa forma, única.

Teorema 2.1.3 *Um poliedro P de dimensão cheia possui uma única (a menos de multiplicações por escalares positivos) representação minimal por um conjunto finito de inequações lineares. Particularmente, para cada faceta F_i de P existe uma inequação $a^i x \leq b_i$ (a menos de multiplicações escalares positivas) representando F_i e $P = \{x \in \mathbb{R}^n : a^i x \leq b_i$ para $i = 1, \ldots, t\}$.*

Existem dois métodos usados comumente para caracterizar facetas de um poliedro. Dado uma inequação válida $\pi x \leq \pi_0$ de um poliedro $P \subseteq \mathbb{R}^n$, o primeiro método (direto), consiste em exibir $\dim(P)$ pontos afim independentes no conjunto $\{x \in P \ : \ \pi x = \pi_0\}$. Uma segunda alternativa (método indireto) é obtido através do seguinte teorema:

Teorema 2.1.4 *Seja $(A^=, b^=)$ o conjunto de igualdades de $P \subseteq \mathbb{R}^n$ e seja $F = \{x \in P : \pi x = \pi_0\}$ uma face própria de P. As seguintes afirmações são equivalentes:*

i) F é uma faceta de P.

ii) se $\lambda x = \lambda_0$ para todo $x \in F$, então

$$(\lambda, \lambda_0) = (\alpha\pi + uA^=, \alpha\pi_0 + ub^=)$$

para algum $\alpha \in \mathbb{R}_+$ e algum $u \in \mathbb{R}^{|A^=|}$.

Pelo que foi exposto até aqui, fica claro que, ao tentarmos obter uma boa formulação para um PLI, iremos nos concentrar nas desigualdades definidoras de facetas da envoltória convexa do conjunto S de soluções do problema. Contudo, nem sempre é simples se chegar a esta situação ideal. Uma técnica comumente empregada para tentar "reforçar" uma desigualdade válida que se conhece para $conv(S)$ recebe o nome de *lifting*.

Muito embora não haja uma definição universalmente aceita sobre o que venha a ser exatamente o *lifting* de uma desigualdade, vamos adotar aquela encontrada em [14], reproduzida abaixo:

> "Dada uma inequação válida $\pi x \leq \pi_0$ para um poliedro $P \subseteq \mathbb{R}^n$, seja F_π uma face de P dado por $\{x \in P : \pi x = \pi_0\}$. Suponha que exista outra inequação válida $\mu x \leq \mu_0$ para P, que define a face F_μ em P. Então, a inequação $\mu x \leq \mu_0$ é dita ser um *lifting* da inequação $\pi x \leq \pi_0$ se ela obedecer às seguintes afirmações:
>
> (a) $F_\pi \subset F_\mu$.
> (b) $dim(F_\pi) < dim(F_\mu) \leq dim(P) - 1$."

Note que, por esta definição, se uma inequação é definidora de faceta, então ela não pode sofrer *lifting*. Ademais, se P está no $\mathbb{R}^n_+$ e os vetores π e μ diferem em apenas uma componente, digamos i, então $\pi_i < \mu_i$. Ou seja, o coeficiente da i-ésima variável aumentou estritamente, enquanto os demais permaneceram iguais. Este tipo de *lifting* é conhecido como *lifting sequencial*, que tem como caso particular o *zero lifting* quando $\pi_i = 0$ e $\mu_i > 0$.

Em alguns trechos desta dissertação são aplicados o último tipo de *lifting*, cuja aplicação é exemplificada pelas proposições que se seguem.

Proposição 2.1.10 *Suponha $S \subseteq \mathbb{B}^n, S^\delta = S \cap \{x \in \mathbb{B}^n : x_1 = \delta\}$ para $\delta \in \{0, 1\}$ e*

$$\sum_{j=2}^{n} \pi_j x_j \leq \pi_0 \tag{2.5}$$

é válida para S^0. Se $S^1 = \emptyset$, então $x_1 \leq 0$ é válida para S. Se $S^1 \neq \emptyset$, então

$$\alpha_1 x_1 + \sum_{j=2}^{n} \pi_j x_j \leq \pi_0 \tag{2.6}$$

é válida para S para qualquer $\alpha_1 \leq \pi_0 - \zeta$, onde $\zeta = \max\{\sum_{j=2}^{n} \pi_j x_j : x \in S^1\}$. Além disso, se $\alpha_1 = \pi_0 - \zeta$ e a equação (2.5) nos dá uma face de dimensão k de $conv(S^0)$, então a equação (2.6) nos dá uma face de dimensão $k+1$ de $conv(S^0)$, nos fornecendo uma faceta de $conv(S)$.

Proposição 2.1.11 *Suponha que 2.5 é válida para S^1. Se $S^0 = \emptyset$, então $x_1 \geq 1$ é válido para S. Se $S^0 \neq \emptyset$, então*

$$\gamma_1 x_1 + \sum_{j=2}^{n} \pi_j x_j \leq \pi_0 + \gamma_1 \tag{2.7}$$

é válida para S para qualquer $\gamma_1 \geq \zeta - \pi_0$, onde $\gamma_1 = \max\{\sum_{j=2}^{n} \pi_j x_j : x \in S^0\}$. Além disso, se $\gamma_1 = \zeta - \pi_0$ e 2.5 nos dá uma face de dimensão k de $conv(S^1)$, então 2.7 nos dá uma face de dimensão $k+1$ de $conv(S)$.

Quando $\alpha_1 = \pi_0 - \zeta$ na proposição 2.1.10 ou quando $\gamma_1 = \zeta - \pi_0$ na proposição 2.1.11, dizemos que o *lifting* é máximo.

2.1.4 Resolvendo PLIs usando PL

Existem duas abordagens clássicas para resolver PLIs usando aproximações lineares: algoritmos de planos de corte (APC) e o B&B ou algoritmo de enumeração implícita. Estes algoritmos são brevemente descritos a seguir.

O algoritmo APC é baseado no reforço sistemático da relaxação do PL mediante a adição de inequações válidas à formulação. Lembrando que uma inequação é dita ser válida em relação a S se ela é satisfeita por todos os pontos de S.

A cada iteração i de APC, uma relaxação linear PL^i do problema PLI é resolvida. Seja x^i uma solução ótima obtida pela relaxação linear PL^i. Se x^i está em S, o algoritmo pàra, retornando x^i como solução ótima do PLI. Caso contrário, a relaxação deve ser fortalecida. Para isto, uma inequação válida $\pi x \leq \pi_0$ para S deve ser encontrada tal que é violada por x^i. Uma nova iteração deve ser executada onde a relaxação PL^{i+1} é obtida através de PL^i adicionando-se a inequação $\pi x \leq \pi_0$ no atual conjunto de restrições lineares. Seja z^i e z^{i+1} os valores das soluções ótimas de PL^i e PL^{i+1} respectivamente, ou seja, $z^i = cx^i$ e $z^{i+1} = cx^{i+1}$. Assumindo que o PLI é um problema de maximização, então $z^{i+1} \leq z^i$ ou, em outras palavras, o limitante superior da solução ótima do PLI decresce monotonicamente a cada iteração.

A idéia do APC é ilustrado na figura 2.1.4 para um PLI (maximização) com 2 variáveis. A região limitada pelas linhas pretas representa o conjunto de soluções viáveis para a relaxação linear. Os pontos indicam as soluções inteiras. As soluções em S são denotados pelos pontos pretos. A flecha indica a direção de maximização e sua cauda está localizada sobre a solução (fracionária) ótima do PL. A linha tracejada representa uma inequação válida que remove tal solução fracionária.

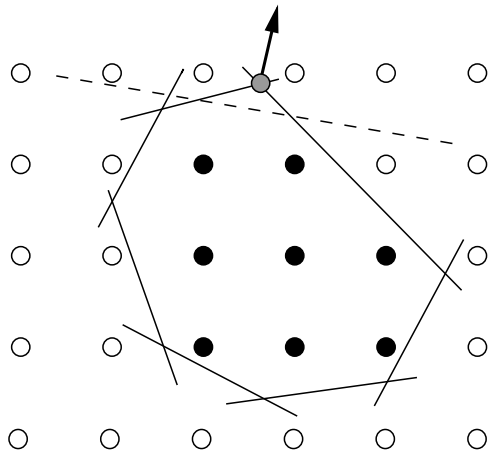


Figura 2.4: Ilustração de um APC.

O estudo inicial sobre a adição de inequações válidas para um problema genérico de PLI foi conduzido por Gomory no fim dos anos 50. Na época as inequações propostas por ele para serem adicionadas na formulação não se mostraram muito eficientes na prática pois a convergência do algoritmo tornava-se demasiadamente lenta. Porém, atualmente, graças a estudos como em [1], tal técnica é muito utilizada, estando presente na maioria dos solvers comerciais.

A abordagem por B&B é baseado no princípio da divisão e conquista, i.e., se for muito difícil otimizar sobre um conjunto S, tal conjunto é particionado em subconjuntos menores na esperança de que estes possam ser otimizados, e através dos resultados obter a solução ótima de S.

Frequentemente, a partição é contruída recursivamente. Isto permite uma representação gráfica de todo o processo na forma de uma árvore: a *árvore de enumeração*. Nesta representação, os filhos de um determinado nó formam uma partição da região viável de seu pai.

Em geral, para problemas de PLI, a árvore de enumeração é uma árvore binária. Cada nó i da árvore corresponde a uma relaxação linear PL^i do problema de PLI definido em

um subconjunto S^i de S. Seja x^i uma solução ótima encontrada para PL^i e $z^i = cx^i$. Dependendo do valor de z^i, o nó i pode gerar dois outros nós (seus filhos) ou pode ser podado, i.e., o subconjunto de soluções viáveis do nó i ou é particionado em dois novos subconjuntos ou não será mais particionado durante os passos restantes do algoritmo.

Suponha que seja decidido pela partição do nó i. Então, uma variável ξ_j com um valor fracionário ψ de x_i é escolhido. A esta variável chamamos de variável de ramificação. A relaxação linear do primeiro (segundo) filho do nó i é obtida pela adição da restrição $\xi_j \leq \lfloor \psi \rfloor$ ($\xi_j \geq \lceil \psi \rceil$) ao PL^i. Portanto, dado um nó da árvore de enumeração, o conjunto de solução viáveis é um subconjunto de S com algumas variáveis possuindo limitantes superiores menores e outras possuindo limitantes inferiores maiores aos limitantes originais. Facilmente podemos verifcar que se nenhum nó é podado, teremos uma árvore de enumeração completa.

Alguns critérios são utilizados para evitar a ramificação de um nó da árvore (fase limitante). Claramente, em um problema de minimização, o valor ótimo de um nó nunca será maior que o valor ótimo de seus filhos. Assim, se soubermos um limitante superior para o valor ótimo do problema PLI original, poderemos podar todos os nós cujas relaxações lineares resultem em um valor ótimo maior que este limitante. Além disto, os nós cujas soluções ótimas são inteiras também podem ser podados. Os nós das árvores que não são podados são chamados de *nós ativos*. O algoritmo para quando não houver mais nós ativos.

Outro método para resolver problemas de PLI agrega uma fase de planos de corte ao algoritmo de *branch-and-bound* formando um algoritmo de *branch-and-cut*. Para descrever tal algoritmo, considere a envoltória convexa de um conjunto viável S denotado por $conv(S)$.

A envoltória convexa de S é um poliedro e, consequentemente, pode ser descrito como um sistema linear de equações e inequações. Se este sistema linear for conhecido, o problema de PLI pode, a princípio, ser resolvido por programação linear, já que todos os seus pontos extremos são soluções viáveis inteiras de S. Infelizmente, o número de inequações no sistema é exponencial para problemas NP-difíceis e usualmente apenas algumas inequações da descrição de $conv(S)$ são conhecidas. Entretanto, suponha que uma certa classe $\mathcal{F}$ de desigualdades válidas fortes para $conv(S)$ seja conhecida. Mais do que isto, dado um ponto $x \in \mathbb{R}^n_+$, suponha que existe um algoritmo que procura por desigualdades em $\mathcal{F}$ violadas pelo ponto x. Tal algoritmo é chamado de rotina de separação (esta nomenclatura ficará clara na seção 2.1.4). Agora podemos descrever o B&C.

Em uma iteração típica do algoritmo, estamos em um nó i da árvore de enumeração e $P^i = \{x \in \mathbb{R}^n \ : \ A^i x \leq b \ , \ 0 \leq x\}$ é o politopo correspondente à relaxação linear PL^i. Se x^i é a solução ótima desta relaxação linear e ela possui componentes (variáveis) fra-

cionários, a rotina de separação é chamada para procurar por uma desigualdade violada em $\mathcal{F}$. Se a rotina de separação retorna uma desigualdade $\pi x \leq \pi_0$, tal desigualdade é adicionada ao sistema de inequações definindo P^i e PL^i é resolvido novamente. Continuamos repetindo tal proesso até que: ou x^i é inteiro, ou z^i é maior do que o atual limitante superior disponível, ou a rotina de separação falha em produzir uma nova restrição que separa o ponto x^i. Neste caso uma variável é escolhida para ser ramificada.

A implementação de um B&C requer um grande esforço. A eficiência do código depende de vários fatores, desde o gerenciamento de dados até os aspectos do algoritmo. Alguns desses aspectos são discutidos abaixo.

Primeiramente, a fase de cortes só é atrativa se uma família $\mathcal{F}$ de desigualdades válidas "fortes" é conhecida. Encontrar tal família implica em um estudo teórico do poliedro que descreve a envoltória convexa das soluções viáveis. Por outro lado, isto também requer a elaboração de um algoritmo para gerar desigualdades violadas em $\mathcal{F}$. Este algoritmo, chamado de *rotina de separação* para $\mathcal{F}$, deve ser rápido e ter uma probabilidade alta de retornar a desigualdade violada em $\mathcal{F}$ quando esta existir.

Mas encontrar uma forte desigualdade válida violada e rotinas de separação não é o suficiente para termos um bom código de B&C. Existem outras partes sensíveis do algoritmo que são intrínsecas para o algoritmo de B&C. Assim, é necessário termos heurísticas para gerar soluções válidas inteiras a partir da solução ótima fracionária dos problemas relaxados (o que pode produzir limitantes superiores que irão podar alguns nós da árvore de enumeração) e um bom critério para ramificação. Melhoramentos adicionais incluem uma fase de preprocessamento que pode, por exemplo, eliminar variáveis *a priori*. Para maiores informações e diferentes estratégias possíveis que podem ser usadas nas implementações dos algoritmos de B&B e B&C ver [24].

A seguir destacamos a importância do conhecimento de facetas do poliedro correspondente a envoltória convexa das soluções de um problema inteiro no desenvolvimento de um APC.

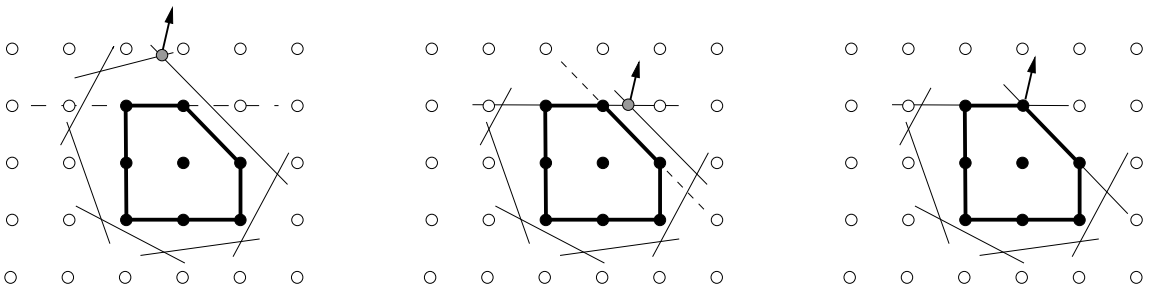
Considere o problema de PLI $\min\{cx : x \in S\}$ onde S é um conjunto finito de pontos inteiros em $\mathbb{R}^n$ (i.e., $S \subset \mathbb{Z}^n$). Este problema pode ser resolvido por qualquer algoritmo de PL, desde que conheçamos um sistema linear de desigualdades $Ax \leq b$ que descreve $conv(S)$ (já que os pontos extremos de $conv(S)$ estão em S). Se toda inequação em $Ax \leq b$ é definidora de faceta de $conv(S)$ e, inversamente, toda faceta F de $conv(S)$ é definida por exatamente uma inequação em $Ax \leq b$, então $Ax \leq b$ é um sistema minimal para $conv(S)$.

Normalmente, na prática não é possível o uso de um sistema minimal descrevendo $conv(S)$. Existem dois principais motivos para isto. O primeiro é que frequentemente o número de desigualdades no sistema minimal é exponencial. Esta afirmação é verdadeira não somente para problemas NP-difíceis, mas também para alguns problemas polinomiais.

O segundo é que, em geral, não temos todas as desigualdades que descrevem $conv(S)$, i.e., existem famílias de desigualdades definidoras de facetas que não são conhecidas. Frequentemente este é o caso dos problemas NP-difíceis.

Entretanto, para uma dada função objetivo, o problema de PLI pode ser resolvido por um PL se o sistema de desigualdade linear correspondente à relaxação de $conv(S)$ contém as desigualdades que definem a solução ótima. Isto sugere o uso de um algoritmo para a solução do problema de PLI que recursivamente melhora a relaxação linear de $conv(S)$ pela adição de novas desigualdades (preferencialmente facetas) na esperança de que, em algum ponto, a solução ótima da relaxação esteja em S. Isto é como o algoritmo de planos de corte (APC) clássico funciona. Porém, numa tentativa de deixar o algoritmo mais rápido, adiciona-se aqui preferencialmente desigualdades que são definidoras de faceta para $conv(S)$.

A figura 2.5 ilustra a idéia de um APC baseado no uso de desigualdades definidoras de faceta. O exemplo dado nesta figura é o mesmo da figura 2.1.4. A região limitada pelas linhas em negrito representam a envoltória convexa de S. Na figura 2.5(a) a solução ótima (fracionária) do PL é separado pela desigualdade definidora de faceta representada pela linha tracejada. A solução ótima da nova formulação (fortalecida) é novamente fracionária mas pode ser separada pela desigualdade definidora de faceta indicada na figura 2.5(b). Embora a próxima formulação PL (figura 2.5(c)) não descreve $conv(S)$, a solução ótima do PL está em S e portanto é a solução ótima do problema PLI.



(a) Passo 1 do APC. (b) Passo 2 do APC. (c) Passo 3 do APC.

Figura 2.5: Ilustração de um APC cujas desigualdades adicionadas definem facetas.

Como dito acima, é crucial que um APC seja capaz de gerar desigualdades violadas. Na próxima subseção o problema da identificação de desigualdades violadas é tratado. Este problema é conhecido como o problema de separação e veremos que, pelo menos na teoria, é tão difícil de ser resolvido quanto o problema de otimização.

Problemas de Separação e Otimização

Considere os seguintes problemas:

Problema de Separação para uma Família de Poliedros:
Dado um ponto $y \in \mathbb{R}^n$ e um poliedro P da família, decidir quando y pertence ou não a P e, se não, encontrar uma desigualdade $\pi x \leq \pi_0$ que é válida para P e tal que $\pi y > \pi_0$.

Problema de Otimização para uma Família de Poliedros:
Dado um vetor $c \in \mathbb{R}^n$ e um poliedro P da família, suponha que $P \neq \emptyset$ e que cx é limitado para todo $x \in P$. Encontre uma solução $x^* \in P$ tal que $cx^* \geq cx$ para todo $x \in P$.

Problema de Separação para uma Família $\mathcal{F}$ de desigualdades
Dado um ponto $y \in \mathbb{R}^n$, mostre que y satisfaz todas as desigualdades em $\mathcal{F}$, ou encontre uma ou mais desigualdades $\pi x \leq \pi_0$ em $\mathcal{F}$ tal que $\pi y > \pi_0$.

Podemos facilmente perceber que, se considerarmos a família de poliedros que são completamente descritos por uma família $\mathcal{F}$ de desigualdades, o primeiro e o terceiro problemas são equivalentes.

O próximo teorema, de Grötschel, Lovasz e Schrijver [17] mostra que as complexidades dos problemas de separação e otimização são equivalentes, ou, em outras palavras, estes dois problemas são igualmente difíceis de se resolver.

Teorema 2.1.5 *Para uma família de poliedros, existe um algoritmo polinomial para o problema de separação se e somente se existe um algoritmo de tempo polinomial para o problema de otimização.*

Seja $conv(S)$ a envoltória convexa de soluções viáveis de um problema de PLI que queremos resolver usando um APC. A cada iteração do algoritmo $conv(S)$ é representado por uma relaxação linear PL($\mathcal{F}$), onde $\mathcal{F}$ é uma família de desigualdades válidas para $conv(S)$. O número de desigualdades em $\mathcal{F}$ é tipicamente exponencial e, portanto, PL($\mathcal{F}$) somente contém algumas delas.

Se $conv(S)$ pode ser completamente descrito usando somente desigualdades em $\mathcal{F}$ e o problema PLI é polinomialmente resolvível, então o teorema 2.1.5 implica que o problema de separação é resolvível polinomialmente. Isto significa que, durante o APC, sempre podemos encontrar em um tempo polinomial uma desigualdade em $\mathcal{F}$ que separa a solução fracionária.

Por outro lado, se o problema de PLI é NP-difícil e $\mathcal{F}$ representa uma ou mais famílias de desigualdades válidas fortes para $conv(S)$, pode haver alguma outra família para a qual o problema de separação é NP-difícil. Consequentemente, haverá alguma instância do problema de PLI para o qual o APC terminará com uma solução fracionária.

Finalmente, um outro algoritmo que foi amplamente utilizado nessa dissertação é conhecido na literatura como *cut-and-branch*. Nesse algoritmo, no primeiro nó da árvore de enumeração aplica-se um APC baseado em alguma família de desigualdades válidas fortes conhecidas para o problema. Ou seja, esse primeiro nó é processado exatamente como no algoritmo *branch-and-cut*. A partir daí, o algoritmo passa a operar exatamente como o algoritmo *branch-and-bound*, isto é, não são mais gerados novos cortes para adicioná-los à relaxação linear que resulta da aplicação do APC no primeiro nó.

2.2 Teoria dos Grafos

Para podermos apresentar os resultados conhecidos sobre a formulação do SPack como um PLI e também para explicar os desenvolvimentos que fizemos ao longo deste trabalho, precisamos de alguns conceitos relacionados à Teoria dos Grafos.

O propósito desta seção é justamente o de introduzir tais conceitos. Contudo, deixamos claro que não temos a pretensão de exaurir o assunto, limitando-nos apenas àqueles tópicos necessários para um melhor entendimento das idéias contidas neste texto. Caso o leitor queira saber mais a respeito deste tópico, pode encontrar outras informações em livros sobre Teoria dos Grafos, como o livro [2], de onde foi retirado o conteúdo abaixo e cujo material pode ser obtido gratuitamente na página do autor, para uso pessoal, em `http://www.ecp6.jussieu.fr/pageperso/bondy/bondy.html`.

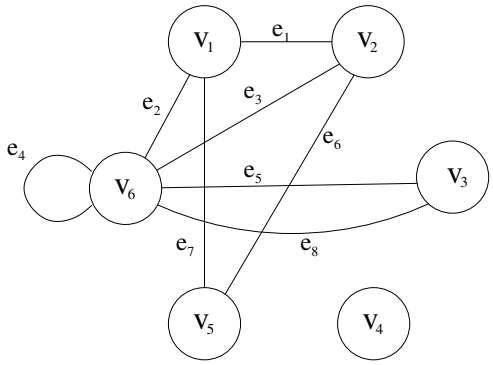
Definição 2.2.1 *Um grafo G é uma tripla ordenada $(V(G), E(G), \psi_G)$ sendo $V(G)$ um conjunto não-vazio de vértices, $E(G)$ um conjunto disjunto de $V(G)$ de arestas e uma função de incidência ψ_G que associa a cada aresta de G um par não-ordenado (não necessariamente distintos) de vértices de G. Se e é uma aresta e u e v são vértices tal que $\psi_G(e) = uv$, então dizemos que e conecta u e v e os vértices u e v são chamados de extremos de e.*

Grafos também podem ser representados facilmente de forma gráfica. Neste caso, cada vértice é indicado por um ponto, ou um círculo, e cada aresta por uma linha conectando os pontos que representam seus extremos.

Apesar de por definição o grafo ser uma tripla, é usual explicitarmos apenas seus conjuntos de vértices e arestas, ou seja, $G = (V(G), E(G))$. Quando apenas um grafo está sendo utilizado, podemos denotá-lo como $G = (V, E)$ ficando subentendido que $V = V(G)$ e $E = E(G)$. Também é usual representar uma aresta pelo par ordenado de seus extremos, ou seja, a aresta e_i com $\psi_G(e_i) = uv$ é a aresta (u, v).

Um exemplo de grafo pode ser visto na figura 2.6. Tal figura servirá também para exemplificar algumas denominações dadas abaixo.

Dizemos que:



(a) G graficamente

$G = (V(G), E(G), \psi_G)$, sendo:
$V(G) = \{v_1, v_2, v_3, v_4, v_5, v_6\}$
$E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$
$\psi(e_1) = v_1v_2 \quad \psi(e_2) = v_1v_6$
$\psi(e_3) = v_2v_6 \quad \psi(e_4) = v_6v_6$
$\psi(e_5) = v_3v_6 \quad \psi(e_6) = v_2v_5$
$\psi(e_7) = v_1v_5 \quad \psi(e_8) = v_3v_6$

(b) G como uma tripla ordenada

Figura 2.6: Exemplo de um grafo G.

- Os extremos de uma arestas são incidentes nela e as arestas são incidentes em seus extremos;
- Dois vértices são adjacentes se forem os extremos de uma mesma aresta, assim como também são adjacentes duas arestas incidentes em um mesmo vértice. Então os vértices v_1 e v_2 são adjacentes pois são os extremos da aresta e_1, e as arestas e_1 e e_3 também são adjacentes pois são ambas incidentes em v_2;
- Para todo $v \in V(G)$, $N_G(v) = \{u \in V(G) : (u, v) \in E(G)\}$ é o conjunto dos vizinhos de v em G;
- Um laço é uma aresta cujos extremos são o mesmo vértice. Em G a aresta e_4 é um laço;
- Duas arestas são paralelas se seus extremos forem idênticos. Assim, as arestas e_5 e e_8 são paralelas;
- Um vértice é isolado quando ele não é adjacente a nenhum outro vértice. Por exemplo, o vértice v_4 é um vértice isolado;
- Um grafo é simples quando ele não possui laços ou arestas paralelas. Portanto, G não é um grafo simples.
- O grau de um vértice v denotado por $\delta(v)$ é o número de arestas de G incidentes em v, sendo que laços contam como duas arestas incidentes no vértice. Logo $\delta(v_6) = 6$, $\delta(v_4) = 0$ e $\delta(v_2) = 3$.

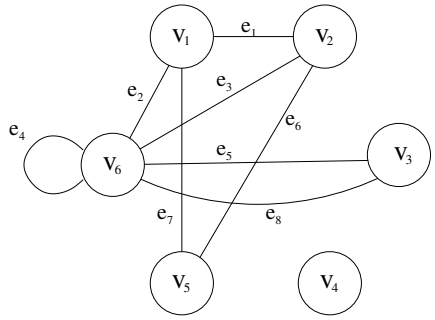
Definição 2.2.2 *Um grafo $G = (V, E)$ é dito k-regular se $\delta(v) = k,\ \forall\ k \in V$*

Definição 2.2.3 *Um grafo H é um subgrafo de G (denotado por $H \subseteq G$) se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$. Quando $H \subseteq G$ mas $H \neq G$, dizemos que $H \subset G$ e chamamos H de subgrafo próprio de G. Se H é um subgrafo de G então G é um supergrafo de H.*

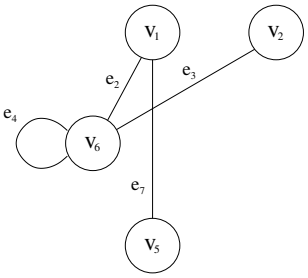
Definição 2.2.4 *Suponha que V' é um subconjunto não-vazio de $V(G)$. O subgrafo de G cujo conjunto de vértices é V' e cujo conjunto de arestas é o conjunto das arestas de G que possuem ambos os extremos em V' é chamado de subgrafo induzido de G e denotado por $G[V']$.*

Definição 2.2.5 *Um subgrafo H de G é dito ser um grafo gerador de G se $V(H) = V(G)$.*

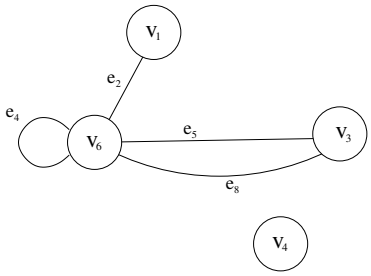
Na figura 2.7 estão alguns exemplos de subgrafos de G (figura 2.6(a)).



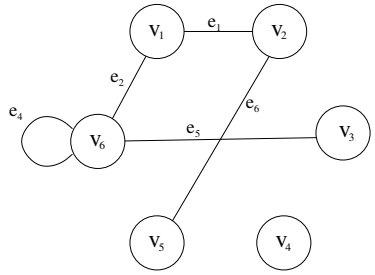
(a) $H_1 = G$



(b) subgrafo próprio H_2



(c) $H_3 = G[\{v_1, v_3, v_4, v_6\}]$



(d) Grafo G_4 gerador de G.

Figura 2.7: Exemplos de Subgrafos de G.

Definição 2.2.6 *Um passeio em um grafo G é uma seqüência finita e não-vazia $W = v_0e_1v_1e_2v_2 \ldots e_kv_k$, cujos termos são vértices e arestas alternados, sendo que para $1 \leq i \leq k$ a aresta e_i é incidente aos vértices v_{i-1} e v_i. Dizemos que W é um passeio de v_0 a v_k, os vértices v_0 e v_k são chamados de origem e fim de W, respectivamente, $v_1, v_2, \ldots, v_{k-1}$ são os vértices internos e o comprimento de W é k.*

Definição 2.2.7 *Se as arestas $e_1, e_2, \ldots, e_k$ de um passeio W são distintas, dizemos que W é uma trilha. Se os vértices $v_0, v_1, \ldots, v_k$ também são distintas, dizemos que W é um caminho.*

Definição 2.2.8 *Um passeio ou trilha é fechado se ele possui um comprimento positivo e sua origem e fim são os mesmos. Uma trilha fechada cuja origem e vértices internos são diferentes formam um ciclo. Quando um grafo não possui ciclos, dizemos que ele é acíclico e quando um ciclo tem comprimento ímpar, ele é chamado de ciclo ímpar.*

Seja $W = v_1(1,2)v_2(2,3)v_3 \ldots (n-1,n)v_n(n,1)v_1$ um ciclo em um grafo G. Dizemos que uma aresta (v_i, v_j) é uma corda de W se $j \neq i+1$. Um ciclo sem cordas é denomidado *buraco*, e um ciclo sem cordas de comprimento ímpar é um *buraco ímpar* (do inglês *odd hole*).

Na figura 2.8 vemos alguns exemplos de passeios descritos acima. Para indicar os passeios utilizamos setas que vão de um vértice a outro sempre acompanhando alguma aresta. Tais setas foram numeradas para indicar a ordem. Assim, se a seta vai de um vértice v_i para o vértice v_j ao lado da aresta $e_{(i,j)}$, ela representa a seguinte seqüência no passeio: $v_i e_{(i,j)} v_j$.

Para introduzirmos alguns outros conceitos sobre grafos, lembremos uma definição da Teoria dos Conjuntos.

Definição 2.2.9 *Dado um conjunto S, dizemos que $(S^i : i = 1, \ldots, k)$ é uma divisão de S se $\cup_{i=1}^{k} S^i = S$. Uma divisão de S é dita ser uma partição de S se $S^i \cap S^j = \emptyset$ para $i, j = 1, \ldots, k$, $i \neq j$.*

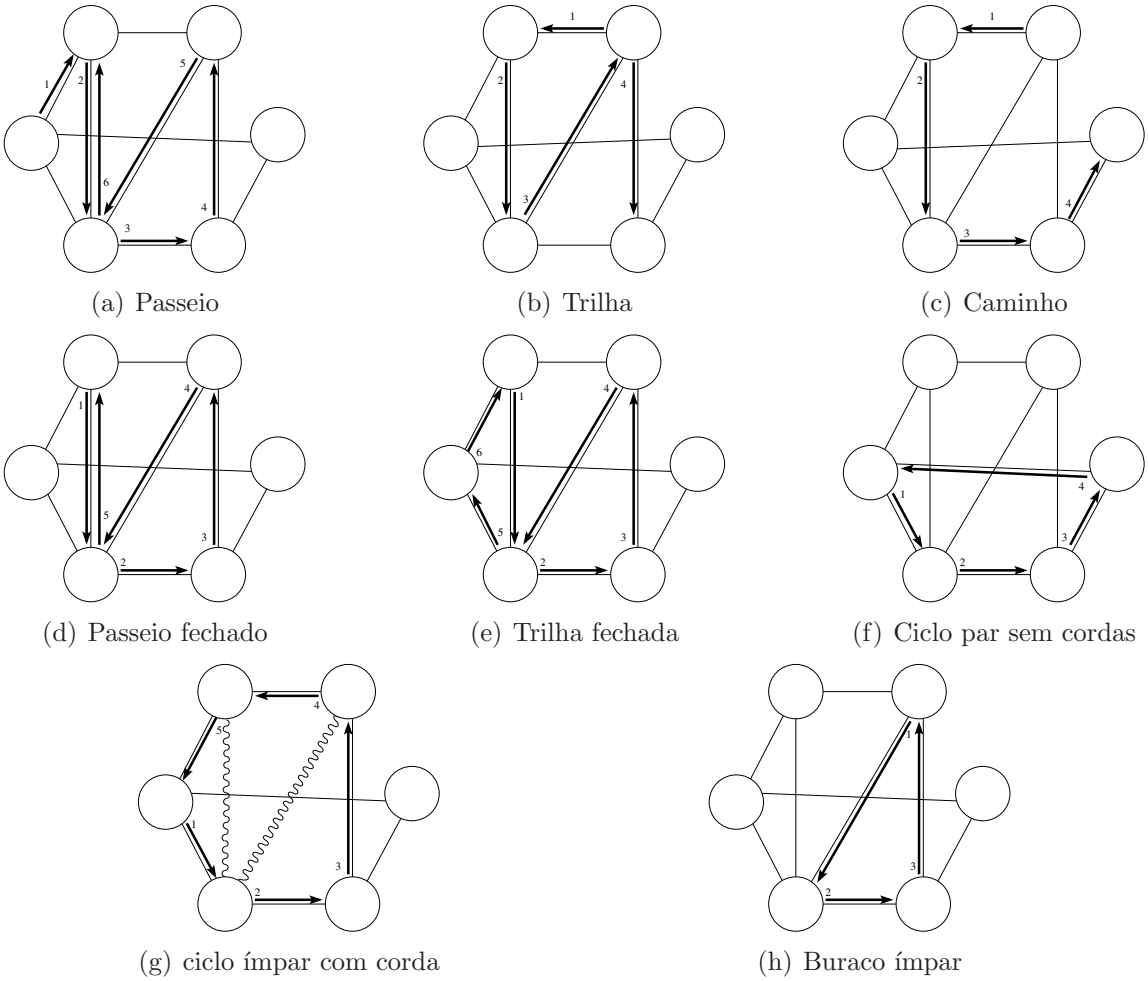
Definição 2.2.10 *Dois vértices u e v de $G = (V, E)$ estão conectados se existe um caminho de u a v em G. Existe uma partição de V em subconjuntos não-vazios $V_1, V_2, \ldots, V_\omega$ tal que dois vértices u e v estão conectados se e somente se ambos pertencem ao mesmo conjunto V_i. Os subgrafos induzidos $G[V_1], G[V_2], \ldots, G[V_n]$ são chamados de componentes de G. Se G possui exatamente uma componente, dizemos que G é conexo, caso contrário G é desconexo.*

Na figura 2.9 estão exemplos de grafos conexos e desconexos.

Definição 2.2.11 *Uma árvore é um grafo acíclico conexo. Uma subárvore de uma árvore T é um subgrafo induzido de T que possui as propriedades de árvore.*

Definição 2.2.12 *Uma subárvore de um grafo G é uma árvore T que seja um subgrafo de G. Uma árvore geradora de G é uma árvore T que seja um subgrafo gerador de G.*

Definição 2.2.13 *Uma união disjunta de árvores é chamado de floresta.*



(a) Passeio (b) Trilha (c) Caminho

(d) Passeio fechado (e) Trilha fechada (f) Ciclo par sem cordas

(g) ciclo ímpar com corda (h) Buraco ímpar

Figura 2.8: Exemplos de passeios.

Analogamente, podemos chamar de floresta geradora de G a união disjunta de várias árvores que juntas formam um subgrafo gerador de G.

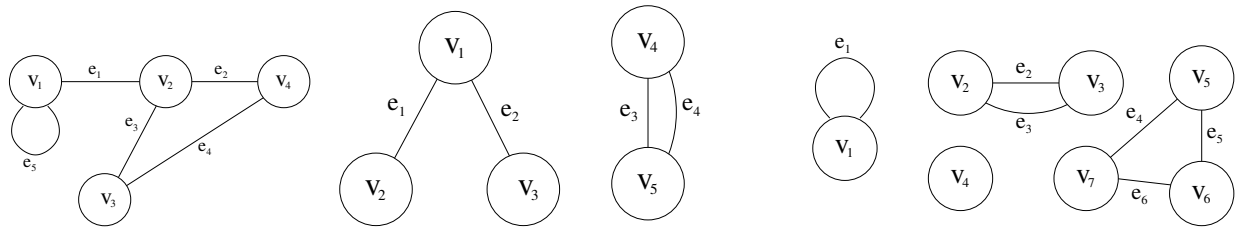
Exemplo de árvores e florestas geradoras são dados na figura 2.10.

Definição 2.2.14 *Um grafo simples, no qual todo par de vértices distintos estão conectados por uma aresta é chamado de grafo completo.*

Um grafo completo com seis vértices pode ser visto na Figura 2.11.

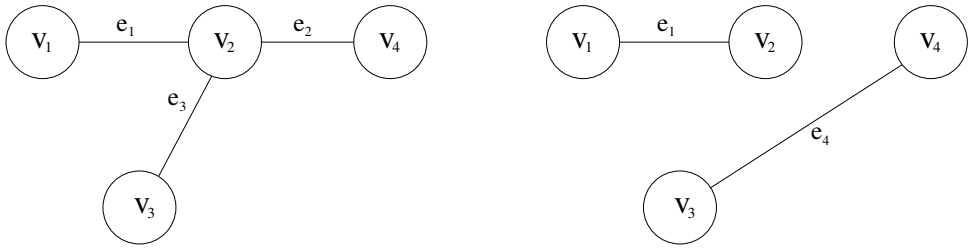
Definição 2.2.15 *O grafo complementar $\overline{G}$ de um grafo simples G, ou o complemento de G, é o grafo simples cujo conjunto de vértices é V, sendo que dois vértices são adjacentes em $\overline{G}$ se e somente se eles não são adjacentes em G.*

Um exemplo de grafo complementar pode ser visto em 2.12.



(a) Grafo conexo (b) Grafo com 2 componentes (c) Grafo com 4 componentes

Figura 2.9: Exemplo de grafos conexos e desconexos.



(a) Exemplo de uma árvore geradora do grafo da figura 2.9(a).

(b) Exemplo de uma floresta geradora do grafo da figura 2.9(a).

Figura 2.10: Exemplo de árvore e floresta.

Definição 2.2.16 *Um grafo direcionado D é uma tripla ordenada $(V(D), A(D), \psi_D)$, tal que $V(D)$ é um conjunto não-vazio de vértices, $A(D)$ um conjunto disjunto de $V(D)$ de arcos, e ψ_D uma função de incidência que associa a cada arco de D um par ordenado (não necessariamente distintos) de vértices de D. Se a é um arco e u e v são vértices tal que $\psi_D(a) = (u, v)$, então dizemos que a conecta u e v, sendo u e v a cauda e a cabeça de a respectivamente. Um grafo direcionado é chamado também de digrafo.*

Definição 2.2.17 *Um digrafo D' é um subdigrafo de D se $V(D') \subseteq V(D), A(D') \subseteq A(D)$ e $\psi_{D'}$ é a restrição de ψ_D com relação a $A(D')$.*

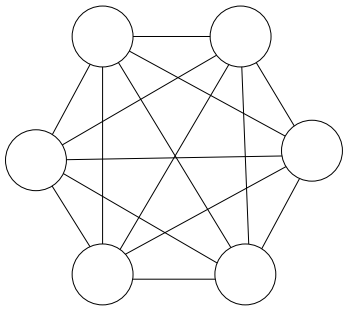
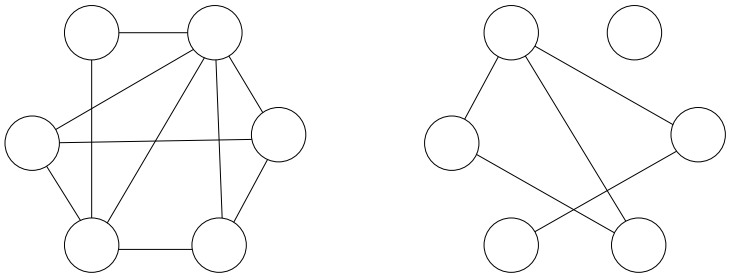


Figura 2.11: Exemplo de um grafo completo.



(a) Exemplo de um grafo C (b) Grafo $\overline{C}$

Figura 2.12: Exemplo de grafo complementar.

As terminologias e notações de digrafos e subdigrafos são análogas aos de grafos e subgrafos.

Definição 2.2.18 *Um passeio direcionado em D é uma seqüência finita e não-vazia $W = (v_0, a_1, v_1, \ldots, a_k, v_k)$ cujos termos são vértices e arcos alternados, tal que para $i = 1, \ldots, k$ o arco a_i tem cabeça v_i e cauda v_{i-1}. Uma trilha direcionada é um passeio direcionado cujos arcos são distintos.*

Caminho direcionado e ciclo direcionado são definidos de maneira similar.
A seguir algumas famílias de grafos bem conhecidas:

Definição 2.2.19 *Um subconjunto M de E é chamado de emparelhamento em G se seus elementos não são laços e não são adjacentes entre eles.*

Um exemplo de emparelhamento são as arestas destacadas na figura 2.13.

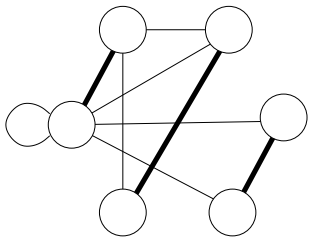
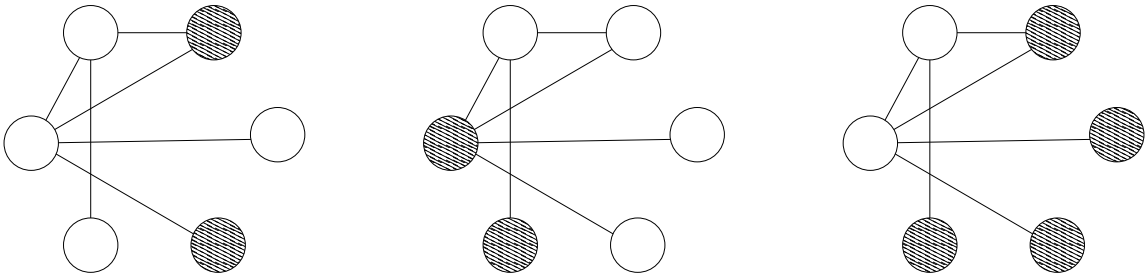


Figura 2.13: Exemplo de um emparelhamento em G.

Definição 2.2.20 *Um subconjunto S de V é chamado de conjunto independente (IS) de G se não existem dois vértices em S que são adjacentes em G. Um conjunto independente S é maximal se não existe nenhum conjunto independente S' tal que $S \subset S'$ e um conjunto independente é máximo se não existe nenhum conjunto independente S', com $|S'| > |S|$.*



(a) IS (b) IS maximal (c) IS máximo

Figura 2.14: Exemplos de conjuntos independentes.

Exemplos de conjuntos independentes podem ser vistos na figura 2.14.

Definição 2.2.21 *Uma clique de um grafo simples G é um subconjunto S de V tal que $G[S]$ é completo.*

Conseqüentemente, S é uma clique do grafo G se e somente se S é um conjunto independente de $\overline{G}$.

Capítulo 3

Desigualdades válidas para o problema do empacotamento

Neste capítulo descrevemos as principais desigualdades válidas presentes na literatura para o problema do empacotamento (SPack). Inicialmente relembramos a formulação PLI do SPack dada no primeiro capítulo desta dissertação. Em seguida, mostraremos a relação existente entre o SPack e o problema do Conjunto independente máximo de um grafo, aqui denotado por maxIS. Como veremos, esta relação será conveniente para descrevermos as classes de desigualdades válidas, muitas delas definidoras de facetas, que são conhecidas para o SPack. Mais que isto, a redução do SPack ao maxIS facilita o desenvolvimento de algoritmos, exatos ou heurísticos, para efetuar a separação destas desigualdes no contexto de um algoritmo de planos de corte, conforme iremos mostrar. Finalmente, de acordo com o que foi exposto na seção 1.2, como estamos interessados em SPacks cujas matrizes de restrições definem um poliedro inteiro a menos de uma restrição, as chamadas matrizes *quase totalmente unimodulares (QTUs)*, terminaremos este capítulo estudando quais dentre as desigualdades válidas conhecidas na literatura são relevantes para este caso particular. É este estudo que irá justificar a não utilização de várias desigualdades conhecidas para o SPack nas implementações que serão discutidas no próximo capítulo.

3.1 Desigualdades válidas e facetas para o SPack

Como vimos no capítulo anterior, para fazer um estudo poliedral para um determinado problema combinatório objetivando o desenvolvimento de um algoritmo baseado em planos de corte, precisamos inicialmente formular este problema usando PLI. Sendo assim, vamos relembrar a formulação do SPack dada no capítulo 1.

Novamente, para efeitos da apresentação, consideramos que a instância do SPack é

composta dos conjuntos finito $M = \{1, \ldots, m\}$ e $C = \{C_1, \ldots, C_n\}$, sendo C um conjunto de subconjuntos de M. Denotaremos por $N = \{1, \ldots, n\}$ o conjunto de índices de C e vamos supor que, para cada $j \in N$, tenhamos um custo c_j associado ao subconjunto C_j de C.

Considere agora a matriz A cujos elementos a_{ij} são definidos abaixo:

$$a_{ij} = \begin{cases} 1, \text{ se } i \in C_j \\ 0, \text{ caso contrário} \end{cases} \quad \forall\ i \in M,\ \forall\ j \in N.$$

A formulação do SPack como um PLI é dada, então, por:

$$\begin{array}{ll} \max & z = c^T x \\ \text{s.a} & Ax \leq \mathbb{1}^m \\ & x \in \mathbb{B}^n \end{array}$$

sendo $\mathbb{1}^k$ o vetor de tamanho k com todos os elementos unitários, e $x \in \mathbb{B}^n$ significa que as variáveis só podem assumir valores binários.

3.1.1 Relação entre o SPack o maxIS

Como visto na seção 2.2, um conjunto independente, ou IS, em um grafo não-direcionado $G = (V, E)$ é um subconjunto dos vértices do grafo tal que não existem arestas entre dois de seus elementos. Se associarmos a cada vértice i de V um peso c_i, definimos o peso de um subconjnuto de vértices de G como sendo a soma dos pesos individuais dos seus vértices. Deste modo, o *problema do conjunto independente máximo de um grafo*, aqui denotado por maxIS, pede que seja encontrado um conjunto independente de G cujo peso seja máximo. O maxIS pode ser modelado pelo PLI abaixo

$$\begin{array}{ll} \max & z = c^T u \\ \text{s.a} & x_i + x_j \leq 1, \forall (i,j) \in E, \\ & x \in \mathbb{B}^n, \end{array}$$

onde, para todo vértice i de V, a variável binária x_i assume valor 1 se e somente se o vértice i faz parte da solução ótima (IS de peso máximo). As restrições da formulação são facilmente entendidas: elas impedem que as duas extremidades de qualquer aresta de G estejam simultaneamente na solução. Isto garante que a solução retornada é, de fato, um IS.

Para a redução inversa, dado um grafo $G = (V, E)$, podemos fazer $M = \{(u, v) : (u, v) \in E\}$, e $C_i = \{(i, v) : (i, v) \in E\},\ \forall\ i \in V$. Assim, um conjunto independente em G poderia formar um empacotamento, pois se não existe aresta entre quaisquer dois vértices

i e j de G, não existe aresta em comum entre quaisquer dois subconjuntos C_i e C_j de C. Logo, o IS pode ser formulado como um SPack.

É possível se chegar a uma formulação mais forte, no sentido discutido no capítulo anterior, para o maxIS. Isto é conseguido substituindo-se as restrições do modelo anterior por restrições do tipo *clique*. Em cada uma destas restrições, todas as variáveis tem coeficientes nulos ou um. Os vértices cujas variáveis tem coeficiente um formam uma *clique* no grafo original e o lado direito da desigualde também tem valor um. Com isto, uma desigualde destas diz que, em uma clique de G, apenas um dos vértices pode pertencer ao IS e, portanto, sua validade decorre diretamente das definições de cliques e conjuntos independentes em grafos.

Em [23] são dados alguns detalhes importantes sobre a formulação forte para o maxIS discutida no parágrafo acima. Dentre elas destacamos que: *(i)* apenas as desigualdades associadas a cliques maximais precisam ser adicionadas ao modelo (veremos o porquê disto mais adiante) e, *(ii)* muito embora, em geral, o número de cliques maximais de um grafo seja exponencial no número de vértices do grafo, uma formulação PLI correta já é obtida se as cliques maximais associadas às restrições do modelo formarem uma *cobertura* para as arestas do grafo de entrada. Ou seja, o maxIS admite uma formulação PLI que, na forma matricial, pode ser expressa por:

$$\begin{array}{ll} \max & z = c^T x \\ \text{s.a} & Ax \leq \mathbb{1}^k, \\ & x \in \mathbb{B}^n, \end{array}$$

sendo que x é o mesmo vetor de variáveis da formulação dada anteriormente para o maxIS, enquanto que, na matriz A com dimensão $k \times n$ e coeficientes binários, o conjunto de vértices associados às variáveis com coeficientes não-nulos em cada restrição formam uma clique maximal em G e as k cliques maximais representadas pela matriz formam uma cobertura das arestas de G. Este último fato faz com que, para cada aresta (i, j) em E, existe pelo menos uma linha de A tal que i e j possuem coeficientes um. Novamente, fica evidente que o maxIS é um caso particular do SPack.

Passemos agora à argumentação de que o SPack pode ser reduzido ao problema do maxIS. Ou seja, precisamos mostrar que, dada uma instância qualquer do SPack caracterizada pela matriz A e pelo vetor c, podemos transformá-la em uma instância do maxIS dada pelo grafo $G(A) = (V, E)$ e custos nos vértices c' tal que, ao obtermos o valor ótimo do $maxIS$ podemos calcular o resultado ótimo do SPack. Para tanto, faz-se o seguinte:

- Para cada variável do SPack, associamos um vértice no grafo.
- Para $i \neq j$, a aresta (i, j) é incluída no conjunto E se e somente se existe alguma restrição do SPack onde os coeficientes das variáveis x_i e x_j são simultaneamente não-nulos.

- Para cada variável x_i do SPack, iguala-se o custo do vértice correspondente em V ao custo da variável x_i, isto é, faz-se $c'_i = c_i$.

É fácil ver que, com as transformações descritas acima, cada restrição do SPack corresponde a uma desigualdade clique na formulação do maxIS resultante da redução entre os dois problemas. Além disto, é imediato perceber que as soluções das instâncias do maxIS e do Spack estão em uma relação um-para-um e que os seus custos são os mesmos nos dois problemas. Assim, resolver a instância do SPack é equivalente a resolver a instância do maxIS .

Uma última observação, mas que é importante para esta dissertação, é que diferentes formulações para a mesma instância do SPack podem levar, por meio da redução descrita anteriormente, a um mesmo grafo de entrada para o maxIS. Para exemplificar esta observação, suponha que no problema SPack desejamos que as variáveis binárias x_i, x_j, x_k e x_l não possam assumir simultaneamente o valor 1. Tal situação poderia ser modelada, por exemplo, pelos quatro conjuntos de restrições mostrados a seguir:

1.
$$\left\{ x_i + x_j + x_k + x_l \leq 1 \right.$$

2.
$$\left\{ \begin{array}{rcrcrcrcl} x_i & + & x_j & & & & & \leq & 1 \\ x_i & & & + & x_k & & & \leq & 1 \\ x_i & & & & & + & x_l & \leq & 1 \\ & & x_j & + & x_k & + & x_l & \leq & 1 \end{array} \right.$$

3.
$$\left\{ \begin{array}{rcrcrcrcl} x_i & + & x_j & & & & & \leq & 1 \\ x_i & & & + & x_k & & & \leq & 1 \\ x_i & & & & & + & x_l & \leq & 1 \\ & & x_j & + & x_k & & & \leq & 1 \\ & & x_j & & & + & x_l & \leq & 1 \\ & & & & x_k & + & x_l & \leq & 1 \end{array} \right.$$

4.
$$\left\{ \begin{array}{rcrcrcrcl} x_i & + & x_j & + & x_k & & & \leq & 1 \\ & & x_j & + & x_k & + & x_l & \leq & 1 \\ x_i & & & & & + & x_l & \leq & 1 \end{array} \right.$$

Embora seja fácil perceber que todas estas formulações represente corretamente a situação desejada, algumas levam a formulações mais fortes do que as outras. Para ilustrar esta afirmativa, note que o vetor $\{0.5, 0.5, 0.5, 0.5\}$ é viável para a formulação 3, mas não

para as demais. Por outro lado, qualquer solução viável para a formulação 1 será viável para as outras formulações também.

Fazendo-se a redução do SPack ao maxIS como proposto nesta seção, podemos facilmente associar resultados da teoria poliedral aplicados a um dos problemas ao outro. Esta observação é bastante útil para a nossa pesquisa, tendo em vista que podemos usar as desigualdades válidas fortes, especialmente as definidoras de facetas, que são conhecidas para o maxIS no desnvolvimento de algoritmos de planos de corte para o SPack. Nos artigos [7], [5], [6], [8], [11], [27], [21], [22], [26], [25], [28], entre outros, são introduzidas várias classes de desigualdades definidoras de facetas para o maxIS e alguns algoritmos de separação para as mesmas. Também em [3] são apresentados diversos aspectos relativos ao estudo poliedral dos problemas de empacotamento, cobertura e partição de conjuntos. Na próxima seção vamos apresentar alguns dos resultados da literatura listados neste último trabalho.

3.1.2 Facetas conhecidas do maxIS

Nesta seção apresentamos algumas desigualdades válidas e definidoras de facetas para o problema do maxIS, ou seja, para o conjunto de soluções inteiras do problema, que nada mais são do que os vetores característicos de conjuntos independentes do grafo de entrada. Denotaremos este conjunto de soluções viáveis por S e, como de praxe, $conv(S)$ denotará a envoltória convexa de S. Como é comum em estudos como o que faremos aqui, as desigualdades válidas serão apresentadas considerando-se o seu suporte (ver definição 2.1.10). Este último fato simplifica o entendimento pois o suporte das desigualdades corresponderão à subgrafos do grafo da instância do maxIS, doravante denotada por $G = (V, E)$. Tais subgrafos são denominados de *grafos suporte* das respectivas desigualdades.

Nem todas as desigualdades válidas elencadas a seguir são definidoras de facetas para $conv(S)$. Contudo, se o grafo que representa a instância do problema for o próprio *grafo suporte* da desigualdade, o resultado se verifica em todos os casos que apresentaremos. Esta informação é relevante pois, em alguns casos, ao trazer novos vértices de G que não estão no grafo suporte da desigualdade, podemos realizar um *lifting* da mesma que, mesmo que não assegure a propriedade de definir uma faceta, ao menos garantirá que obteremos uma desigualdade válida "forte" para o caso do grafo estendido por este novo vértice.

Antes de passarmos à apresentação das desigualdades, é preciso que falemos sobre a dimensão de $conv(S)$, afinal, como vimos no capítulo anterior, este é um passo fundamental para podermos avaliar a "força" de qualquer desigualdade válida para este poliedro.

Proposição 3.1.1 *A dimensão de $conv(S)$ é $n = |V|$, ou seja, o poliedro tem dimensão cheia.*

Prova: Note que os conjuntos vazios e unitários (formados por um único vértice) são sempre conjuntos independentes de G. Os vetores característicos destes conjuntos satisfazem trivialmente as restrições do maxIS logo, eles estão em S. É imediato verificar que estes vetores são afim independentes. Portanto, $\dim(conv(S)) = n + 1 - 1 = n$. □

Cliques. A primeira desigualdade que discutimos aqui é aquela que faz parte da formulação forte do maxIS discutida na seção 3.1.1. Lembremos que todo subgrafo induzido completo de G define uma clique neste grafo. O *tamanho da clique* é dado pela quantidade de vértices que ela contém. Assim, na figura 3.1 vê-se um exemplo de uma clique de tamanho 4.

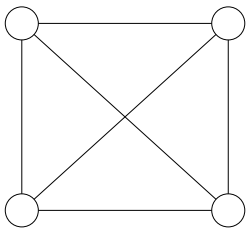


Figura 3.1: Exemplo de uma clique de tamanho 4.

Como vimos, se K é uma clique de G, a *desigualdade clique* associada à K é dada por:

$$\sum_{j \in K} x_j \leq 1. \tag{3.1}$$

Um conceito importante para analisarmos a "força" de uma desigualdade clique refere-se à *maximalidade* do seu grafo suporte. Diz-se que uma clique K em G é *maximal* se o grafo induzido pelos vértices do conjunto $V(K) \cup \{u\}$ em G não é completo, para todo vértice u escolhido em $V \setminus V(K)$. O resultado a seguir relaciona a maximalidade de uma clique em G com a "força" da desigualdade correspondente a ela em $conv(S)$.

Proposição 3.1.2 *Seja C uma clique maximal de G, então a restrição de clique*

$$\sum_{j \in C} x_j \leq 1, \tag{3.2}$$

define uma faceta de $conv(S)$

Prova: Pela proposição 3.1.1, uma faceta de $conv(S)$ possui dimensão $n-1$ e, portanto, contém n pontos afim independentes. Como o hiperplano $\sum_{j \in C} x_j = 1$ não contém a origem, qualquer conjunto de pontos afim independentes nele também serão linearmente

independentes. Então, vamos procurar n pontos linearmente independentes de S que satisfazem 3.2 na igualdade.

Suponha, por simplicidade de notação, que $C = \{1, \ldots, k\}$. Como C é maximal, para cada $j \notin C$ existe um vértice $l(j)$ tal que $l(j) \leq k$ e $(j, l(j))$ formam um conjunto independente.

É fácil ver que os vetores característicos de empacotamento $\{1\}, \ldots, \{k\}, \{k+1, l(k+1)\}, \ldots, \{n, l(n)\}$ são linearmente independentes. □

Note que, se uma desigualdade clique tem como suporte uma clique não maximal do grafo, ao mudarmos coeficientes nulos para um a fim de transformar o grafo suporte numa clique maximal, estaremos fazendo um um *lifting* da restrição original. Ademais, ao término desta operação, chegaremos a uma desigualdade definidora de faceta em $conv(S)$.

Buracos Ímpares (Odd Holes). Um *ciclo ímpar* em um grafo é um ciclo cujos número de vértices (ou arestas, tanto faz) é ímpar. Dado um ciclo, uma *corda* neste ciclo é uma aresta que não faz parte do ciclo mas que liga dois vértices não consecutivos do mesmo. A partir daí, podemos definir um *buraco ímpar* em um grafo como sendo um ciclo ímpar sem cordas (veja exemplo na figura 3.2).

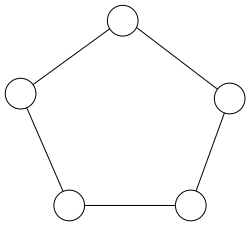


Figura 3.2: Exemplo de um buraco ímpar de tamanho 5.

Seja B um buraco ímpar do grafo G. A *desigualdade do buraco ímpar* correspondente a B é dada por:

$$\sum_{j \in B} x_j \leq \frac{|B| - 1}{2}. \tag{3.3}$$

Em outras palavras, dado um inteiro positivo k, esta desigualdade diz que no máximo k vértices de um buraco ímpar de tamanho $2k + 1$ pode pertencer a um conjunto independente. Este resultado pode ser facilmente verificado a partir de um argumento simples de contagem e, evidentemente, usando a definição de conjuntos independentes em grafos. Na seção 3.4 será mostrado um algoritmo para resolver o problema de separação para as *desigualdades de buracos ímpares* e também será mostrado que, em geral, elas não são definidoras de facetas. Para tanto, mostraremos como é possível fazer um *lifting* de tais desigualdades.

Anti-buracos Ímpares (Odd Anti-holes). Novamente, sendo k um inteiro positivo, um *anti-buraco ímpar (odd antihole)* de tamanho $2k+1$ em G nada mais é do que o complemento de um buraco ímpar de tamanho $2k+1$ (veja exemplo na figura 3.3). Associado a um *anti-buraco ímpar* $\bar{B}$ com $2k+1$ vértices definimos a desigualdade

$$\sum_{v_j \in \bar{B}} x_j \leq 2. \tag{3.4}$$

Novamente, a prova de validade desta desigualdade pode ser feita facilmente usando a definição de conjuntos independentes e um argumento simples de contagem.

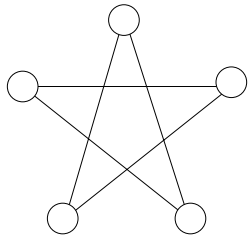


Figura 3.3: Exemplo de um *anti-buraco ímpar* de tamanho 5.

A prova de validade das desigualdades que se seguem é, em geral, um pouco mais complexa do que aquelas dos três casos anteriores. Elas podem ser encontradas em algumas das referências citadas ao final da seção anterior. Por conseguinte, vamos nos limitar a enunciá-las.

Teias (Webs). Uma *teia* $W(p,k)$, $1 \leq k \leq \frac{p}{2}$, é um grafo $T=(R,F)$ tal que $|R|=p$ e $F = \cup_{i=1}^{p}\{(i,i+k),(i,i+k+1),\ldots,(i,i-k+p)\}$. Aqui $p+1$ é identificado com o 1, $p+2$ com 2, e assim por diante (um exemplo pode ser visto na figura 3.4).

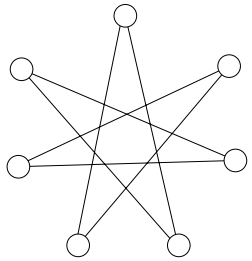


Figura 3.4: Exemplo de uma *teia* $W(7,3)$.

Associada a uma *teia* está a seguinte inequação:

$$\sum_{j \in R} x_j \leq k \tag{3.5}$$

Tal desigualdade é uma faceta se e somente se $k=1$ ou p e k são primos entre si.

Anti-teias (Antiwebs). Uma (ℓ, t)-anti-teia é um grafo formado por um conjunto de vértices $\{v_1, \ldots, v_\ell\}$, onde dois vértices v_i e v_j são adjacentes se $\min\{i-j, \ell+j-i\} \leq t-1$ (um exemplo pode ser visto na figura 3.5). Como no caso das teias, $\ell + 1$ é identificado com o 1, $\ell + 2$ com 2, e assim por diante.

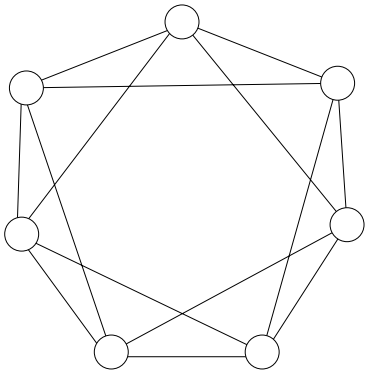


Figura 3.5: Exemplo de uma *anti-teia* $(7, 3)$.

Associada a um (ℓ, t)-*anti-teia* está a seguinte inequação:

$$\sum_{i=1}^{\ell} x_{v_i} \leq \left\lfloor \frac{\ell}{t} \right\rfloor \tag{3.6}$$

Um *anti-teia* é definidora de faceta se e somente se $\ell = t$ ou ℓ e t são primos entre si.

Roda (Wheel). Uma *roda* é um grafo $R = (V, E)$ composto por um ciclo ímpar C com a adição de um vértice adicional que é conectado a todos os outros nós do ciclo (ver figura 3.6 para um exemplo de roda). Denotemos os vértices de C pelos índices $0, 1, \ldots, 2k$ e o vértice adicional por $2k + 1$. O ciclo C é chamado de aro (*rim*) da roda, e o vértice $2k + 1$ de eixo (*hub*), enquanto as arestas conectando o vértice $2k + 1$ ao i, $i = 0, \ldots, 2k$ são chamados de raios (*spokes*). Uma roda cujo aro tem tamanho t é chamado de *t-roda* (do inglês *t-wheel*). Para tal configuração temos a seguinte inequação:

$$kx_{2k+1} + \sum_{i=0}^{2k} x_i \leq k \tag{3.7}$$

Note que esta desigualdade apresenta coeficientes de magnitudes arbitrárias (i.e., não limitados a valores binários), e pode ser obtida através de um *lifting* sequencial do eixo na desigualdade do ciclo ímpar associado ao aro. Até onde sabemos, não existem condições não triviais para identificar quando tal desigualdade é definidora de faceta.

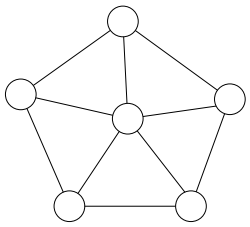
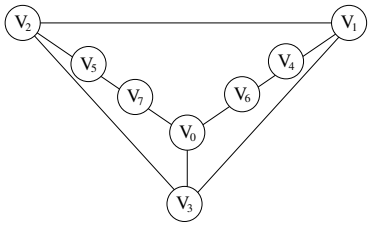


Figura 3.6: Exemplo de uma roda de tamanho 5.

Cunhas (Wedges) Para construir uma *cunha* subdivida os raios de uma 3-roda (não subdividir arestas do aro e pelo menos uma subdivisão deve realmente adicionar um nó) tal que cada face cíclica é ímpar, e considere seu complemento; o grafo resultante é uma cunha (ver exemplo do complemento de uma cunha na figura 3.7). Se particionarmos os nós de uma cunha em um conjunto de vértices $\mathcal{E}$ que possuem uma distância par dos vértices do aro original da 3-roda, deixando os vértices restantes no conjunto $\mathcal{O}$, chegamos a inequação de cunha que afirma que:

$$\sum_{i \in \mathcal{E}} x_i + \sum_{i \in \mathcal{O}} 2x_i \leq 3 \tag{3.8}$$



$\mathcal{E} = \{1, 2, 3, 6, 7\}$
$\mathcal{O} = \{0, 4, 5\}$

Figura 3.7: Exemplo de um complemento de cunha.

A desigualdade de Cunha é definidora de faceta.

Cadeias (Chains). Um $2k+1$-*cadeia* H é similar ao *anti-teia* $C(2k+1, 3)$. A diferença é que as duas cordas $(0, 2k-1)$ e $(1, 2k)$ são substituídas pela aresta $(1, 2k-1)$ (ver figura 3.8).

Associada à $2k+1$-*cadeia* está a seguinte restrição:

$$\sum_{i \in H} x_i \leq \left\lfloor \frac{2k+2}{3} \right\rfloor \tag{3.9}$$

Uma $2k+1$-*cadeia* é definidora de faceta para $conv(S)$ se e somente se $k \mod 3 = 0$.

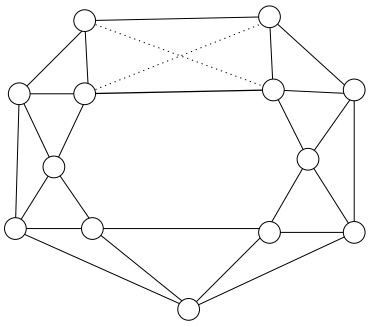


Figura 3.8: Exemplo de um *cadeia*.

3.2 Facetas de SPacks com matriz de restrições QTU

Novamente, suponha que S seja o conjunto de soluções do SPack e $conv(S)$ a sua envoltória convexa. Recordemos que o nosso objetivo é obter desigualdades válidas fortes, preferencialmente definidoras de facetas, para o caso de SPacks descritos por matrizes quase totalmente unimodulares (QTUs). Para simplificar a escrita, chamaremos de QTU-SPack o problema de empacotamento cuja matriz de restrições atende a esta propriedade.

Iniciamos nossas investigações nesta direção através da análise da descrição completa de $conv(S)$ para instâncias pequenas do SPack satisfazendo esta propriedade. Em um primeiro momento, isto foi feito com o intuito de verificar quais desigualdades dentre as que foram listadas na seção anterior apareciam na descrição linear de $conv(S)$ de um QTU-Spack. Para executar esta tarefa, utilizamos um programa especialmente voltado para o cálculo de envoltórias convexas e denominado PORTA.

O nome PORTA é a abreviação de POlyhedron Representation Transformation Algorithm e o código pode ser encontrado em

`www.zib.de/Optimization/Software/Porta/`.

Uma das funcionalidades do programa permite que, dado um conjunto de pontos (vetores) inteiros do $\mathbb{R}^n$, seja computado um sistema linear minimal com n variáveis e que descreve a envoltória convexa destes pontos. Assim, se na entrada for passado o conjunto de vetores que satisfazem a um modelo PLI, na saída temos a envoltória convexa deste conjunto. É claro que isto é uma tarefa extremamente custosa, e por isto, para poder retornar uma solução em um tempo computacional aceitável, o programa só pode ser executado para instâncias muito pequenas.

Como resultado desta tarefa, esperávamos descobrir quais famílias de desigualdades eram mais frequentemente necessárias para descrever $conv(S)$. A partir daí, o próximo passo seria obter bons algoritmos de separação para estas classes de desigualdades de modo a poder utilizá-las em algoritmos baseados no uso de planos de corte, notadamente

branch-and-cut e *cut-and-branch*.

As instâncias do QTU-SPack que pudemos testar no PORTA tinham, no máximo, 20 variáveis. Para instância maiores, o cálculo da envoltória convexa levava um tempo proibitivamente grande, inviabilizando a realização do estudo pretendido. Infelizmente, para as instâncias testadas, constatamos que apenas as desigualdades clique e de buracos ímpares apareciam no sistema linear correspondente a $conv(S)$, sendo que esta última em uma frequência extremamente baixa.

Com isto, para seguirmos com a estratégia de usar os QTU-SPacks na solução de SPacks gerais, conforme propusemos na seção 1.2, foram implementados algoritmos de separação de desigualdades clique e de buraco ímpar

Para encerrar esta seção, cabe observar que existem outros programas que, assim como o PORTA, são capazes de calcular a envoltória convexa de um conjunto de pontos. Este é o caso, por exemplo, do pacote `cdd` que pode ser encontrado em

`www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html`.

Nossa escolha pelo PORTA não teve nenhuma motivação especial mas acreditamos que os outros programas disponíveis não nos permitiriam aumentar substancialmente o tamanho das instâncias que conseguimos tratar de modo a alterar nossa constatação de que as desigualdades clique e de buracos ímpares são, de fato, as mais relevantes para o caso do QTU-SPack.

3.3 Separação de desigualdades clique

Na seção anterior vimos que as desigualdades clique são as que ocorrem mais comumente na descrição da envoltória convexa de instâncias do QTU-SPack. Como discutido no Capítulo 2, para podermos usar estas desigualdades em um algoritmo baseado em planos de corte, é preciso que tenhamos um algoritmo para resolver o problema de separação associado a esta família de desigualdades. Esta seção destina-se à descrição de um algoritmo que responde a esta necessidade.

Foi mostrado na seção 3.1.1 que o SPack pode ser reduzido ao maxIS sobre um grafo. Por outro lado, este problema pode ser formulado usando desigualdades clique de modo que os grafos suportes destas desigualdades formem uma cobertura das arestas do grafo de entrada do problema. Em [23] é descrita uma heurística, denominada CLQ2, que, dada uma solução ótima fracionária da relaxação linear corrente do maxIS, procura por uma desigualdade clique que esteja violada. A heurística CLQ2 é apresentada a seguir. Ela serviu de base para a heurística utilizada por nós nos experimentos descritos no Capítulo 4.

O algoritmo de separação recebe na entrada uma solução fracionária que denotaremos por x. O grafo de entrada do maxIS será representado aqui por $G = (V, E)$. Assim, para cada vértice i de V, temos o valor x_i associado a ele. O objetivo da rotina de separação é encontrar uma clique C tal que a soma de x_i para todo $i \in C$ seja maximizada. Ao final da execução, a rotina terá encontrado uma desigualdade violada se esta soma for superior a um. Note que, na verdade, a rotina está procurando uma clique de peso máximo em G, sendo o peso dos vértices dado pelo vetor x. É sabido que o problema de encontrar a clique de peso máximo em um grafo é $\mathcal{NP}$-difícil e isto justifica porque estamos empregando uma heurística para resolver o problema de separação.

Na descrição da heurística faremos uso da seguinte notação. Para um vértice v qualquer em V, seja $N(v) = \{u \in V : (u, v) \in E\}$ o conjunto dos vizinhos de v e $ST(v) = \{v\} \cup N(v)$. Agora, para cada $u \in V$, a heurística executa os passos descritos no algoritmo 1.

Algoritmo 1: rotina de separação CLQ2 para desigualdades clique

Entrada: grafo $G = (V, E)$, solução fracionária x, vértice $u \in V$
Saída: clique C (vértices)

```
1 C ← {u};
2 P ← N(u);
3 enquanto P ≠ ∅ faça
4 |   Escolha v em P segundo algum critério;
5 |   C ← C ∪ {v};
6 |   P ← P ∩ N(v);
7 fim enqto
8 retorna C
```

Em [23] são propostos dois critérios para realizar a escolha do vértice v feita no laço *enquanto* do algoritmo 1, dando origem as variantes CLQ2A e CLQ2B da rotina de separação das desigualdades clique. Estes dois critérios de escolha são:

(A) Escolha a variável que maximize $\{x_v : x_v < 1\}$.

(B) Escolha a variável que minimize $\{|x_v - \frac{1}{2}| : 0 < x_v < 1\}$

Na implementação que fizemos da rotina CLQ2, CLQ2B é executado apenas quando CLQ2A não encontra nenhuma desigualdade clique violada. No artigo original que propõe a rotina CLQ2 são apresentadas as motivações para o uso dos dois critérios porém nós não vamos discutí-los aqui.

A rotina CLQ2 retornará todas as desigualdades violadas que forem encontradas ao iniciarmos a clique a partir de cada um dos vértices de V. Note que CLQ2 é um algoritmo

rápido e guloso mas, sendo uma heurística, é possível que a rotina não encontre nenhuma desigualdade clique violada, mesmo que exista alguma.

Um questão relevante para o método que propomos na seção 1.2 é que a separação sempre será feita para um QTU-SPack. A seguir procuramos identificar propriedades dos grafos correspondente às instâncias do maxIS obtidos a partir de instâncias do SPack com uma matriz de restrições QTU. A idéia é explorar estas propriedades de modo a tornar mais eficiente o processo de busca por desigualdades clique que estejam violadas.

Seja A a matriz QTU correspondente ao QTU-SPack. Então, A pode ser decomposta de modo que $A = \begin{bmatrix} U \\ \pi \end{bmatrix}$, com U sendo uma matriz de rede pura refletida e π um vetor cujos elementos estão em $\{0, 1\}$. Seja ainda $G(U) = (V_U, E_U)$ o grafo associado à matriz U e $C_\pi = G(\pi) = (V_\pi, E_\pi)$ o grafo (clique) correspondente ao vetor π. Observe que o grafo $G(A) = (V, E)$ associado a A pode ser visto como uma união dos grafos $G(U)$ e $G(\pi)$ (valendo-nos de um certo abuso de linguagem, estamos assumindo que a "união de dois grafos" equivale a união dos seus conjuntos de vértices e arestas).

Como U corresponde a um poliedro inteiro, nenhuma desigualdade válida violada pode ser obtida exclusivamente através das linhas desta matriz. Em outras palavras, isto significa que nenhuma clique que nos leve a uma restrição violada será encontrada no subgrafo $G(U)$. Por outro lado, a desigualdade clique correspondente a C_π faz parte do modelo e não pode ser violada pela solução fracionária corrente.

A partir das observações anteriores, concluímos que uma condição necessária para que a desigualdade clique encontrada pela rotina de separação esteja violada é que ela possua pelo menos alguma aresta de $E_\pi \setminus E_U$, e possua pelo menos uma aresta de $E_U \setminus E_\pi$. Ou seja, a clique deve ter pelo menos alguma aresta não gerada pela matriz TU, e não pode ser a própria clique C_π.

No intuito de atender a propriedade acima, modificamos o algoritmo CLQ2 iniciando a clique C não mais com um único vértice, mas com uma tripla de vértices para a qual já sabemos de antemão que as duas condições do parágrafo anterior sejam atendidas. Assim, baseando-nos nos critérios gulosos CLQ2A e CLQ2B vistos nesta seção, idealizamos duas diferentes formas de inicializar a clique no algoritmo CLQ2:

- *(CLQ2A-INI)* Escolha os vértices i e j tal que $(i, j) \in E_\pi \setminus E_U$ que maximize $\{x_i + x_j : x_i, x_j < 1\}$.

  Em seguida escolha vértice v que maximize $\{x_v : (i, v), (j, v) \in E_U \setminus E_\pi\}$.

- *(CLQ2B-INI)* Escolha os vértices i e j tal que $(i, j) \in E_\pi \setminus E_U$ que minimize $\{|x_v - \frac{1}{2}| : 0 < x_v < 1\}$.

  Em seguida escolha vértice v que minimize $\{|x_v - \frac{1}{2}| : (i, v), (j, v) \in E_U \setminus E_\pi\}$.

Assim, as condições citadas anteriormente serão satisfeitas, e a partir deste ponto basta escolher os vértices como descrito em CLQ2A e CLQ2B.

Na nossa implementação, diferentemente do que foi proposto na heurística CLQ2 original, uma única tentativa de gerar uma restrição clique violada é feita para cada solução fracionária x obtida, ao invés de $|V|$. Para que mais cliques fossem analizadas, poderíamos ter feito uma tentativa para cada aresta de $E_\pi \setminus E_U$, ou ainda para cada tripla possível de vértices satisfazendo a inicialização descrita acima. Entretanto, como poderemos ver nos resultados mostrados no próximo capítulo, o número de cliques encontradas não se mostrou ser um problema.

3.4 Buraco Ímpar

Discutiremos agora um procedimento descrito em [18] que determina se existe uma desigualdade de buraco ímpar violada por uma solução x. Isto permitirá o uso deste tipo de desigualdade em algoritmos para o SPack baseados em planos de corte. Mais uma vez, exploraremos aqui a analogia entre os problemas SPack e maxIS discutidos na seção 3.1.1, considerando diretamente o grafo $G = (V, E)$ resultante daquela redução. Além disto, voltamos a supor a existência de uma solução ótima para a relaxação linear corrente do SPack, denotando-a por x.

Para exemplificar o algoritmo descrito abaixo, vamos utilizar a seguinte formulação para o SPack:

$$\begin{array}{rcrcrcrcrcl}
 & & x_2 & & & & + & x_5 & + & x_6 & \leq 1 \\
 & & x_2 & + & x_3 & + & x_4 & & & & \leq 1 \\
x_1 & + & x_2 & & & & + & x_5 & & & \leq 1 \\
 & & & & & & x_4 & + & x_5 & & \leq 1 \\
 & & & & x_3 & & & & + & x_6 & \leq 1
\end{array} \tag{3.10}$$

A solução x utilizada será $(0, 0.5, 0, 0.5, 0.5, 0)$. Note que tal solução é fracionária e não viola nenhuma da desigualdades da formulação dada.

O maxIS análago a este SPack é sobre o grafo dado na figura 3.9, ao qual chamaremos de grafo G_{ex}.

Inicialmente, para todo $(u, v) \in E$, seja $c_{uv} = 1 - x_u - x_v \geq 0$ o custo da aresta (u, v) e, para todo $u \in V$, crie uma cópia deste vértice, chamada de u', e coloque-a num conjunto V'. Em seguida, construa um grafo bipartido $G' = (V \cup V', E')$ onde $(u, v'), (v, u') \in E'$ se e somente se $(u, v) \in E$. Seja $c_{uv'} = c_{vu'} = c_{uv}$. Tal construção referente ao G_{ex} está na figura 3.10. Agora, note que um caminho em G' de v_0 a $v_{0'}$ é da forma $(v_0, v_{1'}, v_2, v_{3'}, \ldots, v_{2k}, v_{0'})$ corresponde a um caminho em G que visita os vértices $(v_0, v_1, v_2, v_3, \ldots, v_{2k}, v_0)$ nesta ordem, ou seja, é um ciclo com $2k + 1$ nós. Vamos

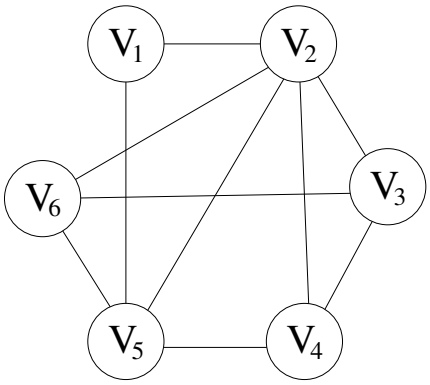


Figura 3.9: Grafo G_{ex} referente ao maxIS análogo ao SPack da formulação 3.10

considerar o caminho $(v_3, v_{2'}, v_4, v_{5'}, v_6, v_{3'})$ no grafo da figura 3.10. No grafo original isto formaria um ciclo ímpar $v_3, v_2, v_4, v_5, v_6, v_3$ de tamanho 5 no grafo G_{ex}. Tais caminhos estão representados na figura 3.11. Ademais, o peso do caminho em G' e do ciclo em G são iguais, e possuem valor:

$$2k + 1 - 2\sum_{j=0}^{2k} x_{vj}. \tag{3.11}$$

Consequentemente, o caminho de peso mínimo de v_0 a $v_{0'}$ em G' nos dá um ciclo ímpar H com $|H| = 2k + 1$ em G que maximiza $\sum_{v\in H} x_v - \frac{|H|-1}{2}$ sobre todos os ciclo ímpares em G contendo v_0. Mais que isto: podemos constatar que $\sum_{v\in H} x_v - \frac{|H-1|}{2} > 0$, ou seja, que há um ciclo violado passando por v_0 em G, se e somente se, o peso do caminho correspondente em G' é menor que 1.

Note que o caminho encontrado não possui peso mínimo. Mas o utilizaremos mesmo assim, pois ele engloba um número maior de etapas do que o caminho de peso mínimo, exemplificando melhor o algoritmo.

Suponha um ciclo ímpar de peso mínimo H contendo v_0 com $\sum_{v\in H} x_v > \frac{|H-1|}{2}$ foi encontrado. Tal ciclo estará contido em um dos três casos abaixo:

(1) $|H| = 3$. Isto significa que o ciclo é um triângulo, e que encontramos uma restrição de clique violada.

(2) $|H| \geq 5$ e H é um buraco ímpar. Uma restrição de buraco ímpar violada foi detectada e iremos tentar fazer um *lifting* desta desigualdade.

(3) $|H| \geq 5$ e o ciclo H contém uma ou mais cordas.

A corda particiona H em um ciclo ímpar H' e um caminho P com número par de

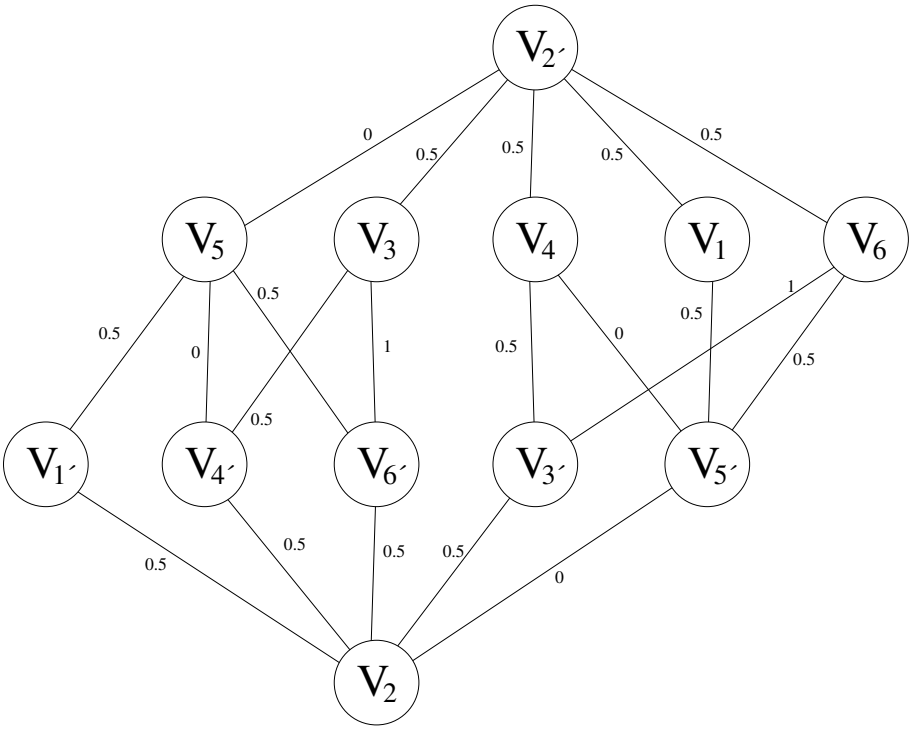
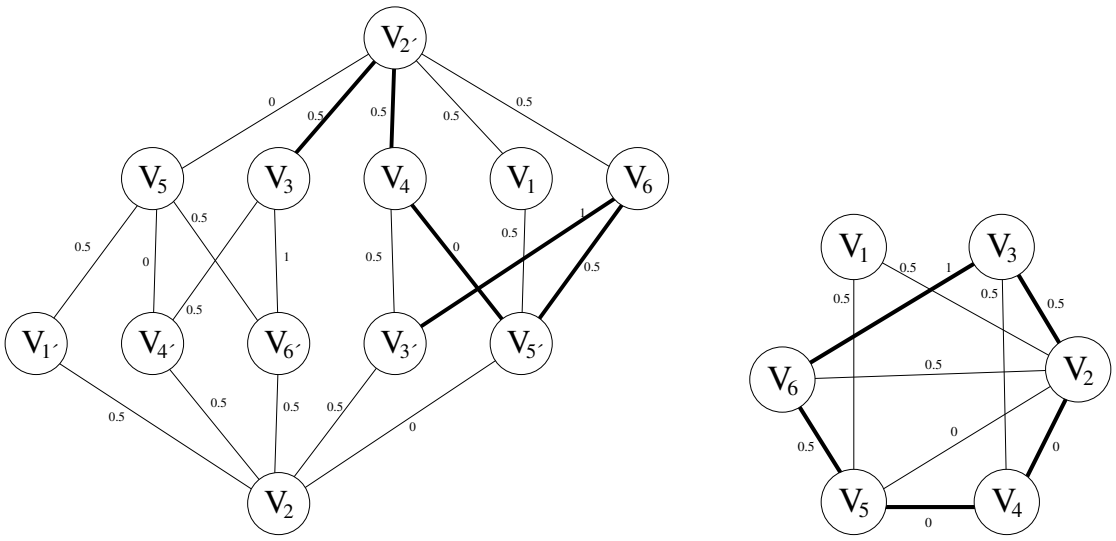


Figura 3.10: Grafo bipartido $G'_{ex} = (V \cup V', E')$



(a) Caminho de v_3 a $v_{3'}$ em G'_{ex}

(b) Ciclo ímpar en G_{ex}

Figura 3.11: Exemplos de como obter um ciclo ímpar em G'

vértices. Temos então:

$$\sum_{v\in H'} x_v + \sum_{v\in P} x_v = \sum_{v\in H} x_v > \frac{|H|-1}{2} = \frac{|H'|+|P|-1}{2}.$$

Somando-se as restrições de aresta sobre P temos que $\sum_{v\in P} x_v \leq \frac{|P|}{2}$, o que nos deixa com:

$$\sum_{v\in H_1} x_v > \frac{|H|-1}{2}.$$

Portanto, temos um ciclo ímpar H', menor que o ciclo original H, para o qual o argumento dado acima pode ser repetido, fazendo com que terminemos o algoritmo com um buraco ímpar violado.

Em qualquer dos casos, podemos aplicar o algoritmo de Floyd-Warshall para encontrarmos os menores caminhos de v a v' para todo $v \in V$, ou seja, temos um algoritmo polinomial que separa as desigualdades de buraco ímpar.

Pela figura 3.11(a), podemos verificar que o ciclo encontrado se enquadra no caso do item 3 acima. Suas cordas são formadas pelas arestas $(v_2, v_5), (v_2, v_6)$ e (v_3, v_4). Cada uma dessas cordas podem ser usadas para particionar o ciclo em um caminho e um buraco ímpar, ambos de tamanho 3.

O buraco ímpar de menor peso é obtido utilizando a corda (v_2, v_5), sendo formado pelos vértices $\{v_2, v_4, v_5\}$, e possui peso $0 < 1$. Assim, x viola a restrição $x_2 + x_4 + x_5 \leq 1$, obtida da equação 3.3, válida para o SPack.

Em relação à situação descrita no item 2, vejamos como proceder a fim de efetuar o *lifting* da desigualde de buraco ímpar. Vale observar que este passo é importante não apenas para tornar mais forte uma desigualdade violada que tenhamos encontrado. Na realidade, ainda que o procedimento anterior tenha falhado na busca por tal desigualdade, é possível que o *lifting* acabe por gerar uma desigualdade violada. Um exemplo gráfico de como tal *lifting* funciona será dado mais para frente.

Vamos nos ater aos casos onde a desigualdade retornada pelo procedimento anterior encontra um buraco ímpar com $k > 1$. O caso em que $k = 1$ corresponde a uma desigualdade clique e, como vimos, um *lifting* desta desigualdade corresponde a construir uma clique maximal que contenha o grafo suporte (clique). Considere então um buraco ímpar H com $2k + 1$ vértices, sendo $k \geq 2$. A desigualdade resultante do *lifting* da inequação original tem a forma:

$$\sum_{v\in H} x_v + \sum_{v\notin H} \alpha_v x_v \leq k,$$

onde $0 \leq \alpha_v \leq k$ é inteiro, e os valores específicos dos coeficientes α_v dependem da ordenação do conjunto $V \setminus H$. Particularmente, se $V \setminus H = \{v_1, v_2, \ldots, v_r\}, r = |V| -$

$(2k+1)$ e os coeficiente são computados na ordem $\alpha_{v_i}, i = 1, \ldots, r$, então:

$$\alpha_{v_j} = k - z_{v_j}, \text{ onde } z_{v_j} = \max \sum_{v \in H} x_v + \sum_{i=1}^{j-1} \alpha_{v_i} x_{v_i}, \tag{3.12}$$

sendo a maximização feita sobre o conjunto dos vetores de incidência de conjuntos independentes do grafo induzido pelos vértices em $(H \cup \{v_1, \ldots, v_{j-1}\}) \setminus N(v_j)$ no grafo suporte da desigualdade corrente. Esta otimização é feita considerando-se que os pesos dos vértices correspondem aos seus coeficientes na desigualdade corrente. Em outras palavras, o que se faz é resolver um problema de conjunto independente de peso máximo no subgrafo induzido mencionado acima.

No procedimento acima, podemos excluir de imediato *lifting* dos coeficientes das variáveis associadas a alguns vértices. Isto é feito da seguinte forma.

Para $v_j \notin H$, seja β_{v_j} a cardinalidade máxima de um IS no grafo induzido por $H \backslash N(v_j)$. Observe que, pela equação (3.12), não é difícil concluir que $\beta_{v_j} \leq z_{v_j}$. Com isto, temos que para todo $v_j \in V$, $\alpha_{v_j} \leq k - \beta_{v_j}$ e chegamos ao seguinte resultado.

Proposição 3.4.1 *Para todo $v_j \in V \setminus H$, se $|N(v_j) \cap H| \leq 2$, então $\alpha_{v_j} = 0$ em (3.12).*

Prova: Imediata, observando que, se não mais do que 2 nós são excluídos de H, então o subgrafo restante ainda possui um IS de tamanho k. □

Para exemplificar como funciona o *lifting*, vamos considerar o grafo G_{lif} dado na figura 3.12. Este grafo veio da relação entre o maxIS e o SPack, cuja formulação é a seguinte:

$$\begin{array}{rcrcrcrcl}
x_1 & + & x_2 & + & x_{11} & + & x_{12} & \leq & 1 \\
x_3 & + & x_4 & + & x_{11} & & & \leq & 1 \\
x_3 & + & x_4 & + & x_{10} & & & \leq & 1 \\
x_1 & + & x_9 & + & x_{12} & & & \leq & 1 \\
x_5 & + & x_6 & + & x_{12} & & & \leq & 1 \\
x_2 & + & x_3 & & & & & \leq & 1 \\
x_4 & + & x_5 & & & & & \leq & 1 \\
x_6 & + & x_7 & & & & & \leq & 1 \\
x_7 & + & x_8 & & & & & \leq & 1 \\
x_8 & + & x_9 & & & & & \leq & 1
\end{array} \tag{3.13}$$

A solução fracionária é dada por $x = (0.5, 0.5, 0, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0, 0)$, e um ciclo ímpar em G_{lif} é formado pelos vértices $H = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$. Assim, a desigualdade de buraco ímpar é

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 4 \tag{3.14}$$

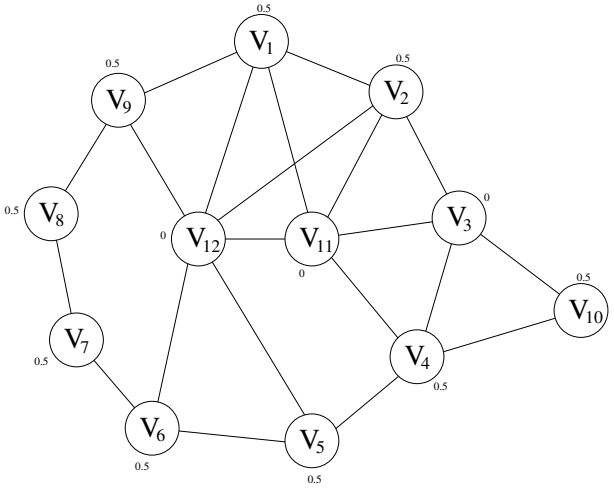


Figura 3.12: Grafo G_{lif} utilizado para exemplificar o lifting em desigualdades de buraco ímpar.

e a ordenação das variáveis a sofrer lifting será em ordem crescente de seus índices.

Pela proposição 3.4.1, o coeficiente de x_{10} na equação 3.14 continuará sendo 0.

Para calcularmos o coeficiente da variável x_{11}, devemos encontrar o valor de $z_{v_{11}}$ no grafo dado pela figura 3.13. Facilmente podemos verificar que o valor ótimo é 3. Assim, o coeficiente de $x_{11} = k - 3 = 4 - 3 = 1$.

Para a variável x_{12}, o grafo induzido que nos auxiliará no cálculo de seu coeficiente é o mostrado na figura 3.14, onde podemos ver que o valor de $z_{v_{12}} = 2$, e, portanto, o valor de $\alpha_{v_{12}} = 2$.

Deste modo, após o cálculo dos coeficientes de todas as variáveis fora de H, obtemos a seguinte desigualdade:

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{11} + 2x_{12} \leq 4$$

Apesar do resultado da proposição 3.4.1, ainda é possível que sobrem vários vértices cujas variáveis correspondentes sejam candidatas a sofrer um *lifting*. Cada permutação destas variáveis irá gerar, potencialmente, uma desigualdade diferente. Ou seja, o número de possíves desigualdades que podemos obter é, no pior caso, exponencial no tamanho do grafo. Além disto, deve ser notado que a computação de cada coeficiente envolve a resolução de um conjunto independente de peso máximo, problema este pertencente à classe $\mathcal{NP}$-difícil.

Como, na prática, não é possível testar todas as permutações das variáveis para se fazer o *lifting*, usualmente, opta-se por uma estratégia gulosa onde os vértices são ordenados de acordo com o valor da variável correspondente na solução corrente e com o seu grau. Além disto, nas implementações feitas neste trabalho, também utilizamos um limitante superior

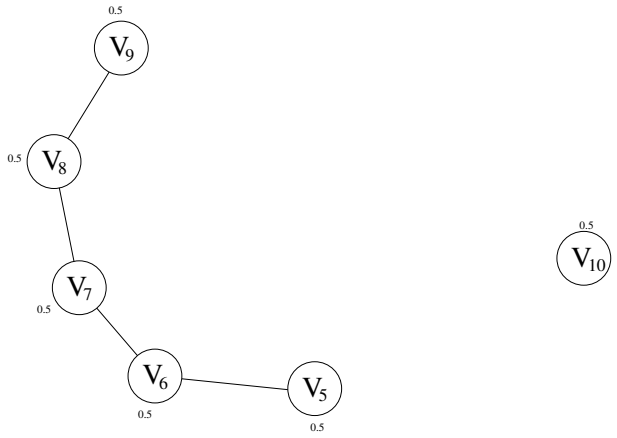


Figura 3.13: Subgrafo induzido $G_{lif}(H \cup v_{10} \setminus N(v_{11}))$

Figura 3.14: Subgrafo induzido $G_{lif}(H \cup \{v_{10}, v_{11}\} \setminus N(v_{12}))$

para o valor de z_{v_j} em (3.12) para evitar resolver exatamente os problemas de IS de peso máximo. Para tanto, fizemos uso da relaxação linear do problema IS, mais exatamente do valor ótimo associado a ela, aqui denotado por z'. Deste modo, calculamos um coeficiente α'_{v_j} cujo valor é menor do que o valor de α_{v_j}, o que nos leva a uma desigualdade mais fraca. Ou seja,

$$z' \geq z_{v_j} \qquad \Longrightarrow \qquad \alpha_{v_j} = k - z_{v_j} \geq k - z' = \alpha' v_j.$$

Logicamente, ao fazer isto, a desigualdade final que obtemos pode não definir uma faceta de $conv(S)$, porém ela sempre será válida para este poliedro.

O algoritmo descrito acima foi implementado neste trabalho. Contudo, para as instâncias em que estávamos trabalhando (oriundas de QTU-SPacks), o número de buracos ímpares encontrados foi insignificante. Conseqüentemente, apesar desta rotina de separação ser rápida, ela foi desativada da versão final dos códigos testados por nós por não compensar os esforços gastos com sua execução.

3.5 Facetas do SPack geral × facetas do QTU-SPAck

Nas duas seções anteriores apresentamos algoritmos de separação que implementamos para o caso do QTU-SPack. Por outro lado, foram exibidas na seção 3.1 outras famílias de facetas do SPack geral para as quais não implementamos rotinas de separação. Assim, é legítimo que o leitor se pergunte porquê tomamos tal decisão. Veremos nesta seção que ela foi motivada pelo fato de que o suporte de algumas destas desigualdades não pode ser um subgrafo do grafo da instância do maxIS correspondente a um QTU-SPack.

Inicialmente, vamos caracterizar os grafos que estão associados a instâncias do SPack cuja matriz de restrições é QTU. Observe que, de acordo com a discussão da seção 3.1.1, cada restrição do SPack corresponde a uma clique do grafo $G = (V, E)$ (instância do maxIS) e, na parte TU da matriz de restrições, existem no máximo dois coeficientes não-nulos por variável. Isto significa que, no subgrafo G' de G obtido apenas ao se considerar estas restrições, cada vértice faz parte de, no máximo, duas cliques. Além disto, deve ser possível atribuir sinais a cada uma destas duas cliques (que representam linhas de uma matriz TU que é uma MRPR) tal que cada vértice não pertença a duas cliques de mesmo sinal. Finalmente, podemos obter o grafo G a partir do grafo G' se adicionarmos a este último os vértices e arestas correspondentes ao grafo suporte (uma clique) da (única) restrição da matriz QTU que não faz parte da submatriz TU.

Deste modo, os grafos correspondentes a instâncias QTU-SPack podem ser caracterizados como sendo aqueles que admitem uma decomposição conforme descrito no parágrafo anterior. Dito isto, uma maneira de provar que um determinado tipo de desigualdade não pode aparecer na descrição do $conv(S)$ de um QTU-SPack é mostrar que o seu grafo suporte nunca pode ser um subgrafo de um grafo que atenda à caracterização acima. No restante desta seção usaremos este argumento para concluir que o restante das desigualdades do SPack que listamos na seção 3.1.1 não podem definir facetas de $conv(S)$ no caso do QTU-SPack.

3.5.1 Anti-buraco ímpar

Para $k = 2$, o *anti-buraco ímpar* pode ser descrito como um buraco ímpar, e sua separação pode ser feita como descrito em 3.4.

Proposição 3.5.1 *Um grafo associado a uma instância QTU-SPack cuja parte TU é uma MRPR não admite um anti-buraco com* $2k + 1$ *vértices como subgrafo se* $k \geq 3$.

Prova: Dado um *anti-buraco ímpar* $H = (V, E)$, com $V = \{v_1, v_2, \ldots, v_n\}$, ordenados em sentido horário, temos que o grau de cada vértice é dado por $\delta(v) = n - 3$.

É fácil perceber que o tamanho da maior clique em H é $\frac{n-1}{2} = k$ e que ela pode ser obtida escolhendo-se os vértices $\{1, 3, \ldots, n-2\}$, ou qualquer configuração simétrica a esta.

Como o grafo que representa a matriz QTU possui no máximo duas cliques contendo cada vértice e mais uma clique adicional referente à linha que não faz parte da matriz `TU`, então pelo menos $n - \frac{n-1}{2} = \frac{n+1}{2} = k + 1 \geq 4$[1] vértices terão que ter suas $n-3$ arestas cobertas por apenas duas cliques (cada clique máxima cobre metade das arestas que saem de um vértice). Para que isto ocorra, cada clique deve ter pelo menos $\frac{n-1}{2}$ vértices, ou seja, ambas devem ter o tamanho máximo possível em um *anti-buraco* e, além disto, uma mesma aresta não pode estar presente em duas destas cliques. Vamos denotar por S_c o conjunto de vértices que possuem todas as suas arestas cobertas por no máximo duas cliques. Nos argumentos acima vimos que $|S_C|$ precisa ser igual a $\frac{n+1}{2} = k + 1 \geq 4$. Deste modo, se mostrarmos que $|S_C| < 4$, chegaremos a uma contradição, o que permitirá concluir que um *anti-buraco ímpar* não pode ser subgrafo do grafo de entrada de um QTU-SPack.

Sem perda de generalidade, devido à simetria do grafo, vamos supor que $1 \in S_c$. A única configuração possível para que as cliques cubram todas as arestas incidentes ao vértice 1, a menos de simetria, é $C_{11} = \{1, 3, \ldots, n-2\}, C_{12} = \{1, 4, 6, \ldots, n-1\}$. Assim, a exceção dos vértices 1, 2 e n, os vértices ímpares pertencem a exatamente uma das duas cliques anteriores, e os vértices pares a outra. Vamos provar agora que, além do vértice 1, não existem outros três vértices que possam pertencer a S_c.

Lembrando que $1 \in S_C$, vamos mostrar que o vértice 2 não pode estar em S_C. Deve ser notado que, por simetria, existe uma prova análoga que mostra que o vértice n também não pode estar em S_C.

Como o vértice 2 não está presente em C_{11} ou C_{12}, iremos precisar de duas outras cliques máximas para cobrir todas as suas arestas. Para que isto ocorra, existe apenas uma configuração possível para tais cliques, que iremos denotar por $C_{21} = \{2, 4, \ldots, n-1\}$ e $C_{22} = \{2, 5, 7, \ldots, n\}$. Logo os vértices em $\{1, 2, 4, 5, \ldots, n-1\}$ já estão presentes em duas das quatro cliques que contruímos até agora e, portanto, nenhum destes vértices pode estar em uma outra clique cuja restrição associada faça parte da MRPR (se estiverem em outra clique, necessarimanete ela deve corresponder à restrição adicional).

Logo, com as cliques construídas até aqui, só os vértices 3 e n não pertencem a duas cliques de tamanho máximo ainda. Mas, para formarmos uma clique máxima com estes dois vértices, precisaríamos usar pelo menos mais $\frac{n-1}{2} - 2 = k - 2$ vértices ou, para $k > 3$, pelo menos mais um vértice. Contudo, como concluímos acima, todos os demais

[1] No melhor caso a clique adicional é máxima e esta seria a quantidade de vértices cujas arestas terão que ser todas cobertas por duas cliques máximas exclusivamente originadas da parte TU da matriz de restrições.

vértices já estão em duas cliques e, assim, não sobraram vértices para construir esta nova clique máxima. Além disto, as arestas ligando vértices pares (exceto 2) a vértices ímpares (exceto 1 e, para $k = 3$, 5) não terão sido cobertas pelas quatro cliques C_{11}, C_{12}, C_{21} e C_{22}. Com isto, nenhum vértice em $\{3, \ldots, n\}$ (exceto pelo vértice 5 quando $k = 3$) têm todas suas arestas contidas em duas cliques máximas. Logo, nenhum outro vértice além de 1 e 2 (e também 5 se $k = 3$) podem pertencer a S_c. Mas isto implica que $|S_c| < 4$, contrariamente ao que queríamos.

Na verdade, o resultado acima pode ser generalizado para concluirmos que dois vértices consecutivos ($(i-1) \mod n+1$ e $i \mod n+1$) do anti-buraco não podem pertencer simultaneamente a S_C. Isto limita o número máximo de vértices que podemos ter em S_C a $\frac{n-1}{2} = k$, ou seja, ficará faltando pelo menos um vértice em S_C para que um anti-buraco possa ser um subgrafo do grafo associado ao QTU-SPack.

□

3.5.2 Roda

Neste trabalho também não foi implementada nenhuma rotina de separação específica para rodas. Veremos a seguir que tal estrutura pode aparecer no grafo associado a uma matriz de restrição quase totalmente unimodular, porém os casos em que isto ocorre são tão específicos que não justificariam procurar separar este tipo de desigualdade.

Inicialmente, descartamos separar rodas com $k = 1$ tendo em vista que elas correspondem a cliques de tamanho 4 e, portanto, podem ser separadas pela rotina descrita na seção 3.3.

A roda com $k = 2$, ou seja, aquela onde o aro tem tamanho 5, pode aparecer como subgrafo do grafo associado a um QTU-SPack. Para observar este fato, considere a instância do QTU-SPack dada pelo sistema linear abaixo:

$$
\begin{array}{lc|ccccccccccc|cc}
 & + & x_0 & + & x_1 & & & & & & & + & x_5 & \leq & 1 \\
 & - & & & x_1 & + & x_2 & & & & & + & x_5 & \leq & 1 \\
\text{TU} & + & & & & & x_2 & + & x_3 & & & & & \leq & 1 \\
 & - & x_0 & & & & & & & + & x_4 & & & \leq & 1 \\
\text{não TU} \Rightarrow & & & & & & & & x_3 & + & x_4 & + & x_5 & \leq & 1
\end{array}
$$

Proposição 3.5.2 *O grafo associado a uma instância de um QTU-SPack não pode conter uma roda com aro de tamanho maior que 7 como um subgrafo.*

Prova:

Para uma roda tendo um aro de tamanho $2k+1 \geq 7$, sabemos que ela possuirá $2k+1$ raios, todos eles, por definição, ligando o nó $2k+1$ (o eixo) a algum nó do aro.

Como uma roda não possui nenhuma clique de tamanho maior que três, em nosso sistema de desigualdades lineares também não teremos nenhuma restrição com mais de três variáveis com coeficientes não-nulos. Além disto, qualquer restrição com suporte de tamanho três necessariamente deve conter a variável correspondente ao nó $2k+1$. Conseqüentemente teremos que cobrir todos os raios com no máximo três triângulos, todos eles contendo o nó $2k+1$. Isto porque o subgrafo correspondente à parte TU da matriz de restrições só pode ter no máximo duas cliques (neste caso, triângulos) contendo o nó $2k+1$, sobrando apenas mais a clique associada à restrição adicional da QTU para ainda conter o referido nó. Porém a cada triângulo da roda, conseguimos cobrir apenas dois novos raios. Logo, com três cliques cobriremos no máximo 6 raios e, para $k \geq 3$, não conseguimos cobrir todas arestas da roda.

□

3.5.3 Teia

Um algoritmo de separação para a desigualdade *teia* também não foi implementado, pois como pode ser visto a seguir, tal estrutura não aparece em uma QTU, ou aparece em um dos casos analisados anteriormente.

Seja $W(n,k)$ um subgrafo *teia*, como descrito na seção 3.1, com n e k primos entre si. Seja $V = (1, 2, \ldots, n)$ os vértices da teia. Para facilitar, toda vez que nos referirmos a um vértice v, estaremos nos referindo, na verdade, ao vértice $(v-1) \mod n+1$.

A seguir listamos algumas propriedades de uma *teia*.

Proposição 3.5.3 *Uma teia $W(n,k)$ com vértices no conjunto V como descrito acima é um grafo $n-2k+1$-regular. Ou seja, para todo v em V, o grau de v é dado por $\delta(v) = n-2k+1$.*

Prova: Por construção, o conjunto $N(v)$ dos vértices adjacentes a v é dado por $N(v) = \{v+k, \ldots, v-k+n\}\ \forall\ v \in V$. Portanto, $\delta(v) = v-k+n-(v+k)+1 = n-2k+1$. □

Proposição 3.5.4 *A maior clique de uma* teia $W(n,k)$ *possui tamanho* $\left\lfloor \frac{n}{k} \right\rfloor$.

Prova: Pela simetria do grafo, podemos dizer que existe uma clique máxima de W contendo o vértice 1. Portanto vamos supor, sem perda de generalidade, que o vértice 1 faz parte da maior clique de W.

Por contradição, vamos supor que exista uma clique Cl em W de tamanho maior que $\left\lfloor \frac{n}{k} \right\rfloor$. Vamos subdividir W em $\left\lfloor \frac{n}{k} \right\rfloor$ subconjuntos de tamanho k cada um e um

subconjunto de tamanho n *mod* k. Seja S o conjunto de tais subconjuntos dado por $S = \Big\{\{1,\ldots,k\},\{k+1,\ldots,2k\},\ldots,\Big\{\Big(\Big\lfloor\frac{n}{k}\Big\rfloor-1\Big)k+1,\ldots,\Big\lfloor\frac{n}{k}\Big\rfloor k\Big\},\Big\{\Big\lfloor\frac{n}{k}\Big\rfloor k+1,\ldots,n\Big\}\Big\}$. Portanto, temos que $|S| = \Big\lceil\frac{n}{k}\Big\rceil$.

Temos dois casos que poderiam acontecer para termos uma clique de tamanho superior a $\Big\lfloor\frac{n}{k}\Big\rfloor$, descritos a seguir:

1. Cada elemento de S possui pelo menos um vértice que faz parte da clique. Neste caso temos que:
$$\Big\{\Big\lfloor\frac{n}{k}\Big\rfloor k+1,\ldots,n\Big\}\cap Cl \neq \emptyset. \tag{3.15}$$
Porém, da Teoria dos Números, sabemos que $k\Big\lfloor\frac{n}{k}\Big\rfloor = n-(n \mod k) \geq n-(k-1)$, e, portanto,
$$\Big\{\Big\lfloor\frac{n}{k}\Big\rfloor k+1,\ldots,n\Big\} \subseteq \{n-k+2,\ldots,n\}.$$
Agora, pela definição de teia, $N(1) = \{k+1,\ldots,n-k+1\}$ e, com isto, concluímos que:
$$N(1)\cap\{n-k+2,\ldots,n\} = \emptyset.$$
Do resultado anterior e da hipótese de que $1 \in Cl$, chegamos a:
$$\Big\{\Big\lfloor\frac{n}{k}\Big\rfloor k+1,\ldots,n\Big\}\cap Cl = \emptyset,$$
contrariando (3.15). Ou seja, o caso descrito acima não pode ocorrer.

2. Algum elemento de S possui mais de um vértice que faz parte da clique.
Pelo item anterior, podemos ver que necessariamente tal subconjunto deve ter tamanho k. Entretanto, como os vértices de cada subconjunto são consecutivos, por construção cada um destes subconjuntos formam um conjunto independente.
Assim este caso também não pode acontecer.

Podemos facilmente ver que o conjunto de vértices $\Big\{1, 1+k, 1+2k,\ldots,1+\Big(\Big\lfloor\frac{n}{k}\Big\rfloor-1\Big)k\Big\}$ formam uma clique, que possui tamanho $\Big\lfloor\frac{n}{k}\Big\rfloor$. Consequentemente, a maior clique de uma *teia* possui tamanho $\Big\lfloor\frac{n}{k}\Big\rfloor$. □

Proposição 3.5.5 *Para $k = 1$, a* teia *corresponde a uma clique.*

Prova: $\forall\ v \in V, \delta(v) = n - 2 + 1 = n - 1$. Como $|V| = n$, W é uma clique. □

Assim, não precisamos nos preocupar em separar a desigualdade da teia quando $k = 1$ visto que este caso será tratado pelo procedimento descrito na seção 3.3.

Proposição 3.5.6 *Para $k = 2$, a* teia *corresponde a um* anti-buraco ímpar.

Prova: Para todo $v \in V$, $N(v) = \{v + 2, v + 3, \ldots, v - 2 + n\}$, ou seja, $N(v) = \{v+2, v+3, \ldots, v-3, v-2\}$ (lembre-se que o vértice i corresponde na verdade ao vértice $(i - 1) \mod n + 1$). Portanto, v é adjacente a todos os outros vértices, com exceção de $\{v + 1, v - 1\}$, formando um *anti-buraco ímpar*. □

Novamente, desta vez de acordo com o que foi exposto na seção 3.5.1, não precisamos nos preocupar com a separação das desigualdades de teia, desta vez para o caso $k = 2$.

Proposição 3.5.7 *Para n ímpar, quando $k = \left\lfloor \frac{n}{2} \right\rfloor$, a teia $W(n, k)$ corresponde a um buraco ímpar.*

Prova: Sabemos ver pela proposição 3.5.3 que W é um grafo 2-regular. Se encontrarmos um passeio fechado que percorra todos os vértices, provamos que $W(n, k)$ forma um ciclo sem cordas, e consequentemente, como n é ímpar, $W(n, k)$ é um buraco ímpar.

Como $k = \left\lfloor \frac{n}{2} \right\rfloor$, é fácil ver que $\left(1, 1 + \frac{n-1}{2}, 1 + 2.\frac{n-1}{2}, \ldots, 1 + n.\frac{n-1}{2}\right)$ é um passeio em W.

Lembrando que o vértice i corresponde ao vértice $(i-1) \mod n+1$, este passeio pode ser reescrito como:

$$\left(1, \frac{n+1}{2}, n, \frac{n-1}{2}, n-1, \ldots, 1\right).$$

Como podemos ver, tal passeio é fechado. Vamos denotá-lo por C. Se provarmos que nenhum dos vértices (com exceção do 1) é visitado mais de uma vez, provamos que este é um ciclo hamiltoniano (i.e., que passa por todos os vértices) em W. Seja então $C = (c_1, c_2, \ldots, c_{n+1})$, de modo que:

$$c_i = \begin{cases} (n - \frac{i-3}{2} - 1) \mod n + 1, & i \text{ é ímpar} \\ (\frac{n-i+3}{2} - 1) \mod n + 1, & i \text{ é par} \end{cases}$$

Para que C seja um ciclo, para todo $1 \leq i, j \leq n$ satisfazendo $c_i = c_j$ deve ser tal que $i = j$.

Vamos supor por contradição que existam $1 \leq i, j \leq n$ tal que $c_i = c_j$ com $i \neq j$. Analisaremos o caso em que i é par e j é ímpar. Os casos em que i e j tem a mesma paridade podem ser mostrados de forma análoga. Assim, temos:

$$c_i = (\frac{n-i+3}{2} - 1) \mod n+1$$

e

$$c_j = (n - \frac{j-3}{2} - 1) \mod n+1.$$

Se $c_i = c_j$ então,

$$(\frac{n-i+3}{2} - 1) \mod n+1 = (n - \frac{j-3}{2} - 1) + 1.$$

Para que a equação acima se verifique, deve existir um inteiro p tal que:

$$n - i + 1 = 2n - j + 1 + 2pn,$$

o que nos leva a $(2p+1)n = j - i$ ou, em outras palavras, que $j - i$ deve ser um múltiplo de n, o que é impossível pois ambos os valores foram escolhidos no conjunto $\{1, \ldots, n\}$ e, deste modo, a diferença entre estes dois valores será no máximo de $n-1$. Logo os valores não se repetem em C que será, assim, um ciclo hamiltoniano. Concluímos daí que W é um buraco ímpar. □

Portanto, quando $k = \lfloor \frac{n}{2} \rfloor$, a separação da desigualdade da teia $W(n,k)$ já é tratada pelo algoritmo descrito na seção 3.4.

O resultado anterior também poderia ser provado através da Teoria dos Números. Em [9], em particular na seção 31.4, podem ser encontrados resultados que nos auxiliam nesta direção. Abaixo resumimos os resultados e os passos que conduzem a esta prova alternativa da Proposição 3.5.7.

Seja $\mathbb{Z}_n$ o conjunto dado por $\{0, 1, n-1\}$ (possíveis restos da divisão de um inteiro qualquer por n). O grupo aditivo módulo n, denotado por Z_n^+, é definido pela operação *soma módulo n* [2] sobre os elementos de Z_n. O subgrupo gerado por $a \in \mathbb{Z}_n$ em $\mathbb{Z}_n^+$ é composto de todos os números da forma $(p.a) \mod n$, sendo p inteiro, e é denotado por $< a >$.

O teorema 31.20 de [9] diz que se $d = mdc(a,n)$ (sendo $mdc(a,n) \equiv$ máximo divisor comum), então $< a >=< d >= (0, d, 2d, \ldots, (\frac{n}{d} - 1)d)$.

No caso da Proposição 3.5.7 temos $n = 2k + 1$ com n e k primos ente si. Então, $d = mdc(k,n) = 1$ e, pelo resultado anterior, $< k >=< 1 >= (0, 1, \ldots, n-1) = \mathbb{Z}_n^+$. Ou seja, com n e k primos entre si, o subgrupo gerado por k é composto por todos os

[2] Ou seja, para a e b em Z_n temos $a + b \equiv (a + b) \mod n$.

números de $\mathbb{Z}_n^+$. Se tomarmos o mesmo conjunto de vértices do passeio C usado na prova da Proposição 3.5.7, veremos que os índices dos vértices que o compõem são exatamente aqueles do subgrupo gerado por k em Z_n^+ (somando-se uma unidade), ou seja, o passeio irá visitar todos os vértices da teia, formando um ciclo hamiltoniano.

Proposição 3.5.8 *Para os casos em que* $3 \leq k < \left\lfloor \frac{n}{2} \right\rfloor$, *a teia* $W(n,k)$ *não ocorre como um subgrafo do grafo associado a um QTU-SPack.*

Prova:

Como já visto anteriormente, para que tal estrutura esteja no grafo associado a uma QTU-SPack, ela deve ser obtida através da união de cliques, no máximo duas por vértice, representando a parte TU da matriz de restrições e mais uma clique adicional representando a restrição responsável por esta matriz ser uma QTU.

Pelas proposições 3.5.3 e 3.5.4 temos que, em $W(n,k)$.:

1. para todo $v \in V,\ \delta(v) = n - 2k + 1$.
2. o tamanho da maior clique é $\left\lfloor \frac{n}{k} \right\rfloor$.

Portanto, cada clique cobre no máximo $\left\lfloor \frac{n}{k} \right\rfloor - 1 < n$ arestas incidentes em cada vértice. Deste modo, a desigualdade clique adicional do problema (aquela que faz com que a matriz seja QTU) não pode conter todos os vértices da *teia*. Assim, temos que pelo menos um vértice deve ter todas suas arestas cobertas por apenas duas cliques. Logo, temos que:

$$2\left(\left\lfloor \frac{n}{k} \right\rfloor - 1\right) \geq n - 2k + 1 \Rightarrow$$

$$2\left\lfloor \frac{n}{k} \right\rfloor - 2 \geq n - 2k + 1 \Rightarrow$$

$$\underbrace{\left\lfloor \frac{n}{k} \right\rfloor}_{\in\mathbb{Z}} \geq \frac{n - 2k + 3}{2} = \frac{n+3}{2} - k$$

Como $\left\lfloor \frac{n}{k} \right\rfloor$ é inteiro, chega-se a:

$$\begin{cases} \text{se } n \text{ é par} & : \quad \left\lfloor \frac{n}{k} \right\rfloor \geq \frac{n+4}{2} - k = \frac{n}{2} - k + 2 \\ \text{se } n \text{ é ímpar} & : \quad \left\lfloor \frac{n}{k} \right\rfloor \geq \frac{n-1+4}{2} - k = \frac{n-1}{2} - k + 2 \end{cases} \Rightarrow$$

Então, em qualquer dos casos, vale a seguinte desigualdade:

$$\left\lfloor \frac{n}{k} \right\rfloor \geq \left\lfloor \frac{n}{2} \right\rfloor - k + 2,$$

ou ainda,

$$\left\lfloor \frac{n}{k} \right\rfloor - \left\lfloor \frac{n}{2} \right\rfloor \geq 2 - k. \tag{3.16}$$

Seja $\left\lfloor \frac{n}{2} \right\rfloor = \frac{n}{2} - \epsilon_1$ e $\left\lfloor \frac{n}{k} \right\rfloor = \frac{n}{k} - \epsilon_2$, com $0 \leq \epsilon_1, \epsilon_2 < 1$. Então:

$$\left\lfloor \frac{n}{k} - \frac{n}{2} \right\rfloor = \left\lfloor \left\lfloor \frac{n}{k} \right\rfloor + \epsilon_2 - \left\lfloor \frac{n}{2} \right\rfloor - \epsilon_1 \right\rfloor = \left\lfloor \frac{n}{k} \right\rfloor - \left\lfloor \frac{n}{2} \right\rfloor + \lfloor \epsilon_2 - \epsilon_1 \rfloor$$

$$\epsilon_1 - \epsilon_2 < 1 \Rightarrow \lfloor \epsilon_2 - \epsilon_1 \rfloor \leq 0$$

Consequentemente,

$$\left\lfloor \frac{n}{k} - \frac{n}{2} \right\rfloor \geq \left\lfloor \frac{n}{k} \right\rfloor - \left\lfloor \frac{n}{2} \right\rfloor \tag{3.17}$$

De (3.16) e (3.17), temos:

$$\left\lfloor \frac{n}{k} - \frac{n}{2} \right\rfloor \geq \left\lfloor \frac{n}{k} \right\rfloor - \left\lfloor \frac{n}{2} \right\rfloor \geq 2 - k \Rightarrow$$

$$\left\lfloor \frac{2n - kn}{2k} \right\rfloor \geq 2 - k$$

Mas, como $3 \leq k < \frac{n}{2}$, $\frac{n}{2k} > 1$ e $2 - k < 0$. Deste modo chega-se a:

$$\frac{n}{2k}(2 - k) < 2 - k \Rightarrow \left\lfloor \frac{n}{2k}(2 - k) \right\rfloor < 2 - k.$$

Assim chegamos a uma contradição e concluímos que não podemos cobrir todas as arestas de um vértice de uma teia com apenas duas cliques. □

3.5.4 Demais desigualdades

Não conseguimos resultado similares que mostram que outras desigualdades vistas na seção 3.1 como, por exemplo, as desigualdades da cunha, não podem ocorrer no caso do QTU-SPack. Além disto, os resultados das seções anteriores praticamente reduziram nossa busca por desigualdades violadas aos casos das cliques e dos buracos ímpares. Infelizmente, estes resultados têm um certo impacto negativo sobre a nossa idéia original de

trabalhar com matrizes QTU pois, os bons resolvedores de PLI já possuem rotinas prontas de separação para pelo menos uma destas classes de desigualdades, quando não para as duas. Mesmo assim, usar as QTUs ainda pode ser uma idéia interessante pois muitas vezes na prática as instâncias do SPack estão associadas a grafos grandes que ocupam bastante memória. Ao trabalharmos só com as QTUs, precisaríamos de menos espaço para armazenamento das informações. Além disto, as desigualdades separadas a partir das QTUs podem vir a ser menos densas do que aquelas provenientes do grafo associado à instância completa. Isto facilitaria o cálculo das relaxações lineares, eventualmente às custas de alguma perda no valor dos limitantes duais. Estas idéias foram exploradas nos experimentos que reportaremos no próximo capítulo.

Capítulo 4

Implementações e Resultados Computacionais

Neste capítulo estão descritos, em pseudo-código, os algoritmos utilizados neste trabalho, assim como os resultados obtidos através deles.

Os códigos foram feitos em linguagem C, compilados e executados em duas máquinas:

- a máquina *M1* – Intel(R) Pentium(R) 4 CPU 2.66 GHz, 1GB de Memória RAM e sistema operacional Ubuntu–Linux com kernel versão 2.6.15-29-686.

  gcc versão 4.0.3-1ubuntu5.

- a máquina *M2* – Intel(R) Core(TM)2 Quad CPU 2.4 GHz, 4 Gb de Memória RAM e sistema operacional Ubuntu–Linux versão 2.6.27-14-generic.

  gcc versão 4.3.2-1ubuntu12

Para nos auxiliar na resolução dos PLIs foram feitas chamadas a funções da biblioteca do resolvedor *xpress*-Optimizer, cujas informações mais detalhadas podem ser obtidas em `http://www.dashoptimization.com/home//products/products_optimizer.html`.

Nos experimentos foram utilizados três blocos de instâncias, em um total de 106, cujos nomes vão de i01 a i106. Os blocos são, respectivamente, as instâncias i01 a i41, i42 a i72, e por fim i73 a i106. Todas essas instâncias foram criadas por um gerador automático, codificado em linguagem PASCAL pelo professor Abilio Lucena (UFRJ). Este programa retorna um vetor de custos c e uma matriz A, recebendo como entrada os seguintes parâmetros referentes à A: número de linhas ($nrows$), número de colunas ($ncols$), número mínimo e máximo de linhas com coeficientes não-nulos por coluna ($minrows$ e $maxrows$ respectivamente). Além disso, como essa matriz será construída a partir de um problema geométrico definido sobre pontos gerados aleatoriamente no plano, existe um parâmetro que especifica o valor máximo de uma coordenada de um desses pontos ($maxcoord$). Por

último tem-se o parâmetro correspondente à semente do gerador de números "aleatórios" utilizado no algoritmo ($iseed$).

O objetivo inicial era a construção de instâncias para o problema de partição. Para que tais instâncias sempre possuíssem pelo menos uma solução viável, as $k = \lceil \frac{nrows}{maxrows} \rceil$ primeiras colunas são construídas de tal modo que a solução $x_i = \begin{cases} 1, \text{se } i \leq k \\ 0, \text{caso contrário} \end{cases}$ seja sempre viável. Para isto, a submatriz M formada pelas k primeiras colunas é formada pelos coeficientes $m_{ij} = \begin{cases} 1, (j-1) \times k < i \leq j \times k \\ 0, \text{caso contrário} \end{cases}$.

Para gerar as outras colunas da matriz são escolhidas aleatoriamente algumas linhas para terem seus coeficientes não-nulos, de tal modo que o total delas esteja entre $minrows$ e $maxrows$.

Para cada coluna $col_i, i = 1, \ldots ncols$ de A é escolhido um ponto aleatoriamente no plano $P = \{(x, y) : 0 \leq x, y \leq maxcoord\}$. Adiciona-se então um ponto inicial qualquer $(x_0, y_0) \in P$. Através destes pontos é construído um grafo completo $G = (V, E)$, onde para cada ponto (x_i, y_i) temos um vértice v_i. O peso das arestas (v_i, v_j) é dada pela distância entre (x_i, y_i) e (x_j, y_j). Para cada coluna de A encontramos um grafo gerador de G, $G(col_i) = (V' \in V, E' \in E)$, onde $V' = \{v_j : a_{ji} = 1\}$, sendo que a_{ji} é o coeficiente de A na linha j e coluna i. Seja T_i uma árvore geradora mínima de $G(col_i)$ e seu peso w_i dado pela soma dos pesos de suas arestas, definimos o vetor de custos da seguinte forma: $c_i = w_i$.

Cada bloco de instâncias agrupa uma família com alguma característica em especial. No primeiro grupo encontram-se instâncias aleatórias, cujo tempo de resolução no $xpress$ versão 2007A em sua configuração padrão, e na nossa máquina de teste *M1* não foi inferior a 900 segundos. Já no segundo grupo, adicionamos a propriedade das matrizes de restrições não possuirem densidade superior a 5%, e no terceiro grupo adicionamos a propriedade de que as instâncias deveriam ser resolvidas em um intervalo de tempo entre 900 e 1800 segundos, pelo $xpress$ em sua configuração padrão na versão 2007A mas, dessa vez, na máquina *M2*.

Para analisar e comparar os resultados, as medidas de tempo foram sempre calculadas em segundos.

4.1 Grafos para representar instâncias do SPack

Durante a realização deste trabalho, pudemos perceber a importância de se representar as instâncias do Spack através de uso de grafos. Isso se deve ao fato de que a visualização do problema e de suas propriedades ficam muito mais fáceis desta maneira. Porém, deve-se tomar cuidado neste capítulo pois trabalhamos com grafos semelhantes para diferentes

finalidades, o que pode gerar confusão ao leitor.

Um dos grafos, ao qual chamaremos de *Grafo por linha*, é utilizado basicamente na heurística para encontrar a maior submatriz MRPR na matriz de restrições do SPack. O outro, aqui chamado de *Grafo por coluna*, é utilizado mais particularmente na resolução exata do problema do SPack, como no preprocesso descrito na seção 4.2, e na geração dos cortes que reforçam as formulações PLI do problema. A seguir descrevemos esses dois grafos detalhadamente.

4.1.1 Grafo por linha

Antes de passarmos a descrever o grafo por linha, introduzimos a seguinte definição .

Definição 4.1.1 *Um grafo sinalizado genérico é representado por uma tripla* $G = (V, E, s)$*, onde* s *é uma função de sinalização das arestas, i.e.,* $s : E \rightarrow \{+, -\}$*.*

O grafo por linha de uma matriz é um grafo sinalizado, não direcionado, onde cada linha da matriz de restrições representa um vértice no grafo. Nele não são pemitidas arestas paralelas de mesmo sinal ou laços.

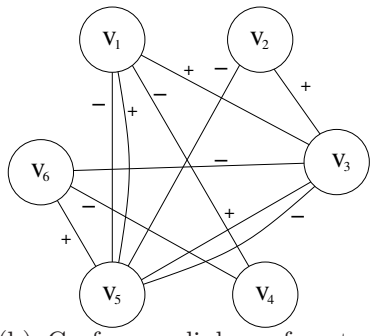
Dois vértices i e j possuem uma aresta conectando-os se e somente para alguma coluna k de A, $a_{ik} \neq 0$ e $a_{jk} \neq 0$, sendo que ela possuirá sinal positivo se os coeficientes forem diferentes, e sinal negativo caso contrário. Note que esta representação supõe que a matriz de restrições possui somente coeficientes de valores em $\{0, 1, -1\}$ e que, no caso particular do SPack, as arestas terão todas sinais positivos pois a matriz de restrições é binária.

A construção deste grafo pode ser vista no algoritmo 2.

Para ilustrar melhor o grafo formado por este algoritmo, segue um exemplo na figura 4.1.

$$\begin{pmatrix} x_1 & & & -x_4 & & & \leq & 1 \\ & x_2 & & & & +x_6 & \leq & 1 \\ & -x_2 & +x_3 & +x_4 & & & \leq & 1 \\ x_1 & & & & +x_5 & & \leq & 1 \\ & & -x_3 & +x_4 & & +x_6 & \leq & 1 \\ & & x_3 & & +x_5 & & \leq & 1 \end{pmatrix}$$

(a) Sistema de inequações



(b) Grafo por linha referente à matriz da figura 4.1(a)

Figura 4.1: Exemplo de um *Grafo por linha*.

Algoritmo 2: ConstroiGrafoLinha(A)

Entrada: matriz A $m \times n$, com $A[i,j] \in \{0, 1, -1\}$
Saída: grafo sinalizado $G = (V, E)$

```
1  V ← {1, 2, ..., m};
2  para c ← 1 até n faça
3  |  para i ← 1 até m − 1 faça
4  |  |  para j ← i + 1 até m faça
5  |  |  |  se A[i, c] ≠ 0 e A[j, c] ≠ 0 então
6  |  |  |  |  se A[i, c] = A[j, c] então
7  |  |  |  |  |  Adiciona aresta (i, j, −) em E se ela não existir;
8  |  |  |  |  senão
9  |  |  |  |  |  Adiciona aresta (i, j, +) em E se ela não existir;
10 |  |  |  |  fim se
11 |  |  |  fim se
12 |  |  fim para
13 |  fim para
14 fim para
15 retorna G;
```

4.1.2 Grafo por coluna

O grafo por coluna é representado por um grafo não direcionado, onde cada coluna da matriz de restrições representa um vértice no grafo. Dois vértices i e j serão adjacentes se e somente se para alguma linha k da matriz $a_{ki} = a_{kj} = 1$. Em outras palavras, cada restrição formará uma clique no grafo.

Note que esta representação assume que a matriz possui coeficientes cujos valores estão em $\{0, 1\}$, como no caso do SPack.

O algoritmo 3 realiza a construção do grafo por coluna para uma matriz binária A passada como entrada.

Um exemplo de um grafo por coluna é dado na figura 4.2.

4.2 Preprocesso

Um preprocesso normalmente consiste em um algoritmo que muda a instância de entrada com o objetivo de deixá-la mais fácil ou simples de resolver. Para que este objetivo possa ser alcançado, esta etapa deve equilibrar o tempo gasto com o ganho de tempo decorrente de suas alterações. Assim, é preferível que eles sejam rápidos, embora preprocessos mais demorados sejam aceitos caso o ganho resultante de sua aplicação seja alto.

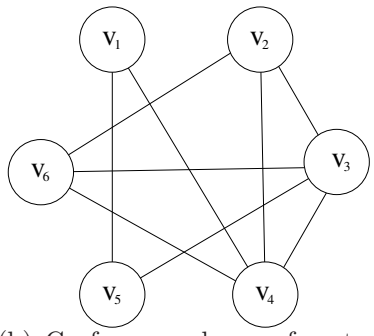
Algoritmo 3: ConstroiGrafoColuna(A)

Entrada: matriz A $m \times n$, com $A[i,j] \in \{0,1\}$
Saída: grafo sinalizado $G = (V, E)$

```
1  V ← {1, 2, ..., n};
2  para l ← 1 até m faça
3  |  para i ← 1 até n − 1 faça
4  |  |  para j ← i + 1 até n faça
5  |  |  |  se A[i, c] ≠ 0 e A[j, c] ≠ 0 então
6  |  |  |  |  Adiciona aresta (i, j) em E;
7  |  |  |  fim se
8  |  |  fim para
9  |  fim para
10 fim para
11 retorna G;
```

$$\begin{pmatrix} x_1 & & & -x_4 & & & \leq & 1 \\ & x_2 & & & & +x_6 & \leq & 1 \\ & -x_2 & +x_3 & +x_4 & & & \leq & 1 \\ x_1 & & & & +x_5 & & \leq & 1 \\ & & -x_3 & +x_4 & & +x_6 & \leq & 1 \\ & & x_3 & & +x_5 & & \leq & 1 \end{pmatrix}$$

(a) Sistema de inequações



(b) Grafo por coluna referente à matriz da figura 4.2(a)

Figura 4.2: Exemplo de um *Grafo por linha*.

Preprocessar as instâncias de entrada pode ser muito vantajoso em alguns casos. Em um problema de PLI, em geral os preprocessos tentam obter um modelo melhor, a partir do inicial. Eles podem, por exemplo, inserir novas restrições ou alterar as já existentes. Podem também torná-lo menor, eliminando algumas restrições, fixando variáveis, ou modificando seus limitantes.

Neste projeto utilizamos um algoritmo que tenta melhorar a formulação fortalecendo suas desigualdades, conforme descrito a seguir.

4.2.1 Lifting por Cliques Maximais

Seja A a matriz de restrições de uma instância do SPack. Sabemos que a cada restrição do problema corresponde uma clique no grafo por coluna de A. Vimos no Capítulo 3 que essa

desigualdade só poderá definir uma faceta da envoltória convexa das soluções do SPack se a clique associada a ela for maximal. Assim, parece natural que a matriz A deva ser tal que todas as desigualdades nela representadas tenham cliques maximais como grafo suporte. Contudo, em geral, isso não ocorre. Além disso, verificamos que os resolvedores comerciais que tínhamos disponíveis também não garantiam que essa propriedade fosse verificada. Finalmente, pareceu-nos razoável que a nossa busca por uma matriz MRPR que serviria de base para a definição das matrizes QTU deveria ser feita sobre uma formulação mais forte.

Os fatores listados anteriormente levaram-nos a desenvolver um preprocessamento em que, na saída, todas as restrições do modelo matemático do SPack estivessem associadas a cliques maximais. Para isso, a cada clique C associada a uma restrição r de A, verificamos para cada vértice $v \in V \setminus C$ se ele poderia ser adicionado a C tal que este continuasse sendo uma clique. Caso fosse possível fazíamos $C = C \cup \{v\}$, repetindo o processo até que nenhum novo vértice v pudesse ser incluído em C. Evidentemente, as variáveis correspondentes aos vértices adicionados a C passam a ter coeficiente de valor um em r, ou seja, sofrem um *lifting*. O método acima está descrito no algoritmo 4.

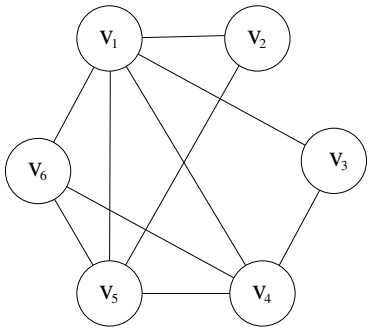
Note que ao fazer o preprocesso é possível que a clique resultante contenha uma ou mais cliques suporte de restrições do modelo original além daquela que a gerou. Isso torna redundante essas últimas, sendo necessário retirá-las posteriormente. Esse fato é ilustrado pelo exemplo da figura 4.3.

$$\left(\begin{array}{c|cccccc} (1) & & & & x_4 & + x_5 & & \leq & 1 \\ (2) & & & & x_4 & & + x_6 & \leq & 1 \\ (3) & x_1 & & & & + x_5 & + x_6 & \leq & 1 \\ (4) & x_1 & & + x_3 & + x_4 & & & \leq & 1 \\ (5) & x_1 & + x_2 & & & + x_5 & & \leq & 1 \end{array}\right)$$

(a) Matriz original A

$$\left(\begin{array}{c|cccccc} (1) & x_1 & & & + x_4 & + x_5 & + x_6 & \leq & 1 \\ (2) & x_1 & & + x_3 & + x_4 & & & \leq & 1 \\ (3) & x_1 & + x_2 & & & + x_5 & & \leq & 1 \end{array}\right)$$

(b) Matriz preprocessada A'



(c) $G[A] = G[A']$

Figura 4.3: Exemplo de lifting que deixa restrições redundantes.

Por motivo de simplificação, denotaremos daqui em diante a instância original por $Inst$ e sua matriz de restrição por A, e a instância preprocessada por $Inst'$ e sua matriz de restrição por A'.

Elaboramos um conjunto de instâncias para testarmos a mudança deste preprocesso na formulação do problema. Notou-se que a matriz de restrições e, conseqüentemente, o poliedro de soluções viáveis foram modificados drasticamente em alguns casos. Na

Algoritmo 4: Lifting por cliques maximais

Entrada: Matriz A $m \times n$
Saída: Matriz A alterada
Dados: Grafo $G = (V, E)$, conjunto S de inteiros, vetor de vértices v (em ordem crescente de prioridade, booleano *adiciona*)

```
1  G ← ControiGrafoColuna(A);
2  para cada linha l de A faça
3  |  S ← índices dos coeficientes não-nulos da linha i;
4  |  para i ← 1 até n faça
5  |  |  adiciona ← verdadeiro;
6  |  |  para cada s ∈ S faça
7  |  |  |  se (v[i], s) ∉ E então
8  |  |  |  |  adiciona ← falso;
9  |  |  |  fim se
10 |  |  fim para cada
11 |  |  se adiciona = verdadeiro então
12 |  |  |  S ← S + {v[i]};
13 |  |  |  A[i, v[i]] = 1;
14 |  |  fim se
15 |  fim para
16 fim para cada
17 para cada linha l de A faça
18 |  se l for redundante então
19 |  |  Retire linha l da A;
20 |  fim se
21 fim para cada
22 retorna A
```

tabela 4.1 estão os dados onde podemos verificar tais alterações. As colunas representam, respectivamente, o nome da instância utilizada, o número de linhas, a densidade de A em porcentagem, o valor da relaxação linear de $Inst$, o número de linhas e densidade também em porcentagem de A', o valor da relaxação linear de $Inst'$, e o número de colunas do problema, o qual não muda durante este processo.

Podemos notar que, para densidades de A superiores a 15%, o preprocesso gerou A' com densidades extremamente altas (a maioria maior que 95%), sendo que para tais casos o número de linhas de A' também foi reduzido drasticamente, chegando a ficar com menos de 50 linhas em sua maioria. Deduzimos com isto que, nesta situação, o poliedro formado ficou menor e, portanto, o problema ficou mais fácil de ser resolvido. Esta afirmação pode ser comprovada olhando-se o valor do ótimo das relaxações lineares. Além de, nos casos citados acima, estes valores terem se reduzido, ou seja, o limitante dual ficou mais próximo do ótimo inteiro, na maior parte dos casos ele também foi inteiro. Devido à forma como as instâncias foram geradas, com os custos das variáveis inteiros, isto pode ser um indicativo de que este seja também o valor ótimo do problema inteiro.

Já para instâncias de baixa densidade (inferior a 10%) não foram observadas grandes mudanças no valor ótimo da relaxação linear. Como a formulação após o *lifting* por clique maximais pode ser potencialmente diferente da original, tentamos resolver o primeiro bloco de instâncias antes e depois da alteração para verificar se o procedimento de preprocesso era vantajoso. Para isso precisaríamos comparar o tempo gasto para se resolver $Inst$ com o tempo necessário para resolver $Inst'$. Nos dois casos estamos nos referindo ao tempo necessário para calcular exatamente os dois modelos usando um resolvedor de PLI, no caso o *xpress*. Além disso, para o $Inst'$ este tempo deve ser acrescido daquele despendido pelo preprocesso. Iremos denotar tais tempos por *tempo de preprocesso* e *tempo de resolução*, sendo o *tempo total* dado pela soma dos dois tempos anteriores.

Porém, não podemos esquecer que ainda estamos tratando de um problema NP-difícil e, portanto, pode ocorrer que o tempo se torne proibitivamente alto. Assim, limitamos o tempo de execução do *xpress* em 900 segundos nesses testes. Utilizamos para esta análise o *xpress* versão 2007A, a versão mais nova disponível em nossos laboratórios na época, e mantivemos sua configuração padrão. Consideramos que o uso do preprocesso foi vantajoso se o tempo total para resolver $Inst'$ foi menor que o tempo para resolver $Inst$, ou ainda, se conseguimos resolver $Inst'$ em um tempo total menor que 900 segundos e não conseguimos resolver $Inst$ neste mesmo intervalo de tempo.

Na tabela 4.2 estão os dados obtidos pela resolução de $Inst$ e $Inst'$ apenas das instâncias em que o preprocesso foi desvantajoso. Suas colunas são, respectivamente, o nome da instância, o tempo que o *xpress* levou para resolver $Inst$, o tempo de resolução de $Inst'$, o tempo de preprocessamento, o tempo total gasto para resolvermos $Inst'$ e, por último, a diferença de tempo de resolução dos dois problemas, em outras

instância	original			preprocessadas			#colunas
	#linhas	dens.(%)	rel. lin.	#linhas	dens.(%)	rel. lin.	
i01	300	25.94	13642.62	235	95.77	7330	500
i02	300	28.05	11523.17	4	99.50	7330	500
i03	300	23.27	12498.63	23	98.87	9602	500
i04	300	21.94	15737.15	3	99.56	11106	600
i05	300	25.08	14174.41	3	99.56	11106	600
i06	300	14.36	22203.79	300	78.27	19539	700
i07	300	22.65	17429.59	4	99.46	15551	700
i08	300	5.22	45195.62	300	5.23	45195.62	799
i09	300	18.53	22381.64	20	99.25	21237	799
i10	300	25.14	18984.48	3	99.67	14468	799
i11	300	5.22	51546.42	300	5.23	51546.41	899
i12	300	20.15	24575.27	8	99.47	20030	900
i13	300	25.14	21366.03	3	99.67	16280	799
i14	300	2.03	101241.17	299	2.04	101241.17	999
i15	300	5.21	57978.52	300	5.22	57978.52	999
i16	300	10.16	41667.69	300	14.47	41170.41	1000
i17	300	15.17	31591.62	300	85.30	28575	999
i18	300	20.13	27519.07	8	99.52	22268	981
i19	300	25.14	23758.45	3	99.67	18104	799
i20	400	0.95	59643.99	387	0.98	59641.40	499
i21	400	6.44	26484.40	400	6.77	26064.26	500
i22	400	10.27	21177.36	400	38.64	19010	499
i23	400	15.17	17111.28	46	98.25	13961	500
i24	400	21.40	14247.82	4	99.25	12627	500
i25	500	22.17	15513.90	4	99.25	14607	500
i26	500	5.15	32149.00	500	5.28	31958.90	500
i27	500	11.08	21980.48	500	78.66	21105	500
i28	500	15.08	18832.14	6	98.83	17940	500
i29	500	19.10	16714.95	5	99.20	16477	499
i30	500	25.13	14472.42	3	99.47	11255	500
i31	400	6.41	32770.04	400	6.53	32609.65	600
i32	400	11.35	24114.56	400	55.97	22847	599
i33	300	1.21	110227.07	298	1.21	110227.06	998
i34	300	4.36	61079.47	300	4.36	61079.47	999
i35	300	6.88	49339.84	300	6.89	49339.84	999
i36	400	3.25	97594.63	400	3.25	97594.63	1200
i37	400	5.74	72350.24	400	5.74	72350.23	1199
i38	400	7.01	65030.13	400	7.02	65028.14	1200
i39	500	3.11	144070.21	500	3.11	144070.21	1499
i40	500	5.11	108738.97	500	5.11	108738.97	1500
i41	500	7.14	90143.23	500	7.24	89924.49	1500

Tabela 4.1: Instâncias preprocessadas.

palavras, o tempo gasto pelo *xpress* com a instância original subtraído do tempo total gasto para se resolver a instância preprocessada.

instância	tempo				diferenca de tempo
	original	preprocessada			
	xpress	*xpress*	preprocesso	total	(or - prep)
i17	334	0	401	401	-67
i35	243	252	3	255	-12
i38	377	398	11	409	-32

Tabela 4.2: Dados das instâncias cujo preprocesso foi desvantajoso.

Note que em apenas três instâncias, das 41 testadas, fazer o preprocesso piorou o tempo necessário para resolver o problema. Além do número ser pequeno, ainda temos o fato que a diferença de tempo entre eles não foi muito grande, chegando a um máximo de 67 segundos. No caso do problema i17 isto se deu pois o preprocesso levou muito tempo para ser realizado, tendo este tempo sido maior que o tempo gasto para resolver o problema original. Os outros dois casos provavelmente o preprocesso alterou, mas não melhorou significativamente a formulação fazendo com que a relaxação linear apenas ficasse mais difícil de ser resolvida, levando a um desempenho pior a longo prazo.

Na tabela 4.3 encontramos os dados referentes às instâncias em que o preprocesso foi vantajoso. Note que alguns dos problemas não foram resolvidos até o limite de 900 segundos dados ao *xpress*. Nesta situação os tempos foram marcados com um "*" ao lado (como, por exemplo, em i10, i13 e i18) e o cálculo da diferença de tempo de resolução de $Inst$ e $Inst'$ não se aplica, já que em um caso ele foi resolvido e em outro não. Por este motivo, existe um "N/A" marcado na coluna de diferença de tempo sempre que esta situação se verificou.

Podemos perceber que aproximadamente 66% das instâncias testadas encontram-se nesta tabela, o que nos leva a concluir que o preprocesso pode ser extremamente vantajoso. Note ainda que 54% das instâncias tiveram suas soluções ótimas encontradas já na resolução da relaxação linear do problema. Ademais, 12% das instâncias não conseguiram ser resolvidas em 900 segundos pelo *xpress* sem o preprocesso, enquanto que $Inst'$ foi resolvido em um tempo total que variou de 60 a 519 segundos.

Para as demais 11 instâncias, este experimento foi inconclusivo, pois o *xpress* não conseguiu resolver tanto $Inst$ quanto $Inst'$ em menos de 900 segundos. Poderíamos analisar os limitantes que foram encontrados nos dois casos, porém, apesar deles darem uma medida da qualidade da formulação, melhores limitantes em um determinado momento não necessariamente resultam em um menor tempo de resolução do problema no final. Assim, decidimos não levá-los em consideração na análise do desempenho do preprocesso.

instância	tempo				diferenca de tempo (or - prep)
	original	preprocessada			
	Xpress	Xpress	preprocesso	total	
i01	81	0	35	35	46
i02	149	0	38	38	111
i03	159	0	36	36	123
i04	235	0	77	77	158
i05	431	0	85	85	346
i06	267	0	84	84	183
i07	611	0	152	152	459
i09	439	0	240	240	199
i10	1046*	0	262	262	N/A
i12	540	0	388	388	152
i13	1049*	0	262	262	N/A
i16	196	102	27	27	169
i18	1041*	0	519	519	N/A
i19	791	0	262	262	529
i21	38	34	0	0	38
i22	114	0	9	9	105
i23	464	0	40	40	424
i24	686	0	49	49	637
i25	918*	0	60	60	N/A
i26	39	33	0	0	39
i27	413	0	30	30	383
i28	655	0	50	50	605
i29	300	0	55	55	245
i30	971*	0	66	66	N/A
i31	59	56	1	1	58
i32	273	0	32	32	241
i37	326	318	5	5	321

Tabela 4.3: Dados das instâncias cujo preprocesso foi vantajoso.

Um outro fato que pode ser observado é que o preprocesso é um método muito vantajoso nos problemas cuja densidade é grande, sendo que em sua maioria o problema pode ser resolvido considerando-se apenas a sua relaxação linear. Por outro lado, nas instâncias cuja densidade é pequena o preprocesso levou um tempo muito pequeno para ser realizado.

Vale ressaltar que, este preprocesso também foi estudado em um trabalho conjunto com o doutorando André Ciré, do IC- UNICAMP. Nos testes adicionais que foram realiza-

dos pudemos perceber que este preprocesso apresenta basicamente dois comportamentos. Quando as matrizes possuem densidades maiores, ele reduz a formulação de tal forma que ela pode ser resolvida pela sua relaxação linear. Já nas matrizes de menores densidades, ele altera muito pouco a formulação, porém gasta um tempo insignificante para ser realizado.

Entretanto, ao rodarmos tal preprocesso em versões mais recentes do *xpress*, como os problemas conseguem ser resolvidos em um tempo significativamente menor, o preprocesso se tornou menos vantajoso.

São exibidos na tabela 4.4 os resultados do preprocesso com a versão 2008A do *xpress*. Deve-se notar que só foram explicitados as instâncias de testes que foram resolvidas em alguma das formulações.

Podemos perceber através desta tabela que, em geral, o tempo gasto apenas para resolver o problema foi menor nos casos preprocessados. Portanto, se esta etapa fosse feita de maneira mais rápida, obteríamos um ganho de desempenho.

É fato que, em alguns casos, o preprocesso não foi capaz de diminuir o tempo gasto para encontrar o ótimo. Além disto, um grande número de instâncias reais do SPack possuem matrizes de restrições com densidades extremamente pequenas, fazendo com que a formulação após o preprocesso não sofra grandes alterações. Mesmo levando em considereção estes fatos, achamos que vale a pena aplicar o preprocesso, pois o seu custo/benefício pode ser considerado bom.

Após estes testes, especificamos um pouco melhor as instâncias utilizadas em nossos experimentos. Assim, decidimos restringir os testes a instâncias cujas matrizes de restrições não tivessem densidades superiores a 5%. As razões principais na qual esta decisão se apoia são:

1. na maioria das instâncias reais do SPack a matriz de restrição tem uma densidade não superior a este percentual e

2. para instâncias com matrizes de alta densidade, o preprocesso conduz a um modelo que, muitas vezes, é resolvido já na sua relaxação linear, tornando-as desinteressantes para os objetivos deste trabalho.

Ao nos limitarmos a instâncias com as características acima, novamente comprovamos que o preprocesso altera muito pouco o poliedro formado. Porém, em todos os casos testados o preprocesso não demorou mais do que 5 segundos, sendo que na grande maioria demorou menos de 1 segundo, justificando assim o uso deste método em nossos experimentos.

instância	tempo				diferenca de tempo
	original	preprocessada			
	Xpress	Xpress	preprocesso	total	(or - prep)
i01	81	0	35	35	46
i02	63	0	38	38	25
i03	43	0	36	36	7
i04	100	0	77	77	23
i05	106	0	85	85	21
i06	90	0	84	84	6
i07	145	0	152	152	-7
i09	72	0	240	240	-168
i10	247	0	262	262	-15
i12	328	0	388	388	-60
i13	277	0	262	262	15
i16	362	139	27	166	196
i17	227	0	401	401	-174
i18	419	0	519	519	-100
i19	272	0	262	262	10
i21	68	45	0	45	23
i22	48	0	9	9	39
i23	57	0	40	40	17
i24	62	0	49	49	13
i25	91	0	60	60	31
i26	43	68	0	68	-25
i27	62	0	30	30	32
i28	76	0	50	50	26
i29	32	0	55	55	-23
i30	89	0	66	66	23
i31	79	94	1	95	-16
i32	64	0	32	32	32
i35	288	418	3	421	-133
i37	584	401	5	406	178
i38	555	802	11	813	-258

Tabela 4.4: Preprocesso XPRESS versão 2008A.

4.3 O Problema de Encontrar a Maior MRPR

Como já foi visto anteriormente, encontrar a maior submatriz que nos leve a um poliedro inteiro é um problema difícil e que possui uma grande aplicação na resolução de PLIs. Uma das famílias dessas matrizes são as MRPRs. Por esta razão existem na literatura diversas heurísticas para encontrá-la. Denotaremos por $maxMRPR$ o problema de se encontrar a maior MRPR em uma matriz de restrições. Algumas heurísticas para esse problema estão descritas a seguir.

4.3.1 A heurística de Gulpinar et al.

O algoritmo 7, denotado por *GGMZ* foi implementado baseado no artigo [19]. Seus principais conceitos são descritos abaixo.

Considere um grafo sinalizado $G(A) = (V, E, s)$, associado à matriz de restrições A de um SPack, e W um subconjunto não-vazio de V. A seguinte notação será utilizada:

- $\nu(A)$: o número máximo de linhas de uma submatriz MRPR de A.
- $\alpha(G)$: o tamanho do maior conjunto independente em G.
- $\mu(G)$: número de arestas negativas de G.
- Para um subconjunto não vazio W de V, define-se $\overline{W} = V \setminus W$.
- G^-: grafo obtido através de G pela remoção de todas as arestas positivas.
- G^W: grafo obtido através de G trocando-se os sinais das arestas que ligam W a $\overline{W}$.
- o valor de $\overline{\alpha}(G)$ será dado por:

$$\overline{\alpha}(G) = \max\{\alpha((G^W)^-) : W \subseteq V\} \tag{4.1}$$

A heurística implementada é baseada no resultado do seguinte teorema:

Teorema 4.3.1 $\nu(A) = \overline{\alpha}(G(A))$.

Este teorema é obtido a partir dos próximos dois lemas. Um grafo sinalizado $G = (V, E, s)$ é balanceado (do inglês *balanced*) se existe um subconjunto $W \subseteq V$ tal que todas as arestas no subgrafo de G induzido por W e $\overline{W}$ são positivas e todas as arestas entre W e $\overline{W}$ são negativas; observe que, neste caso, $\mu(G^W) = 0$. Seja $\eta(G)$ a maior ordem de um subgrafo balanceado induzido de G.

Lema 4.3.1 *Para um grafo sinalizado* $G = (V, E, s), \eta(G) = \overline{\alpha}(G(A))$.

Lema 4.3.2 *Para uma matriz* A *com coeficientes em* $\{-1, 0, 1\}$*, temos que* $\nu(A) = \eta(G(A))$.

Assim, a rigor, poderíamos encontrar a maior submatriz MRPR em A se obtivéssemos o maior conjunto independente em $\left(G(A)^W\right)^-$ para todo $W \subseteq V$. Obviamente que essa tarefa não pode ser feita assim pois o número de subconjuntos W é exponencial em $|V|$ e o problema de encontrar o conjunto independente máximo de um grafo é NP-difícil. Uma alternativa para essa dificuldade é relaxarmos a necessidade de resolução exata do problema, ou seja, usar o Teorema 4.3.1 como base para procurarmos uma MRPR em A com uma grande quantidade de linhas sem, contudo, garantir que ela é máxima.

Para entendermos melhor a heurística, precisaremos dos seguintes conceitos:

Chamaremos de grafo de rede o grafo H para o qual existe uma MRPR A tal que $H = G(A)$. Pelo teorema 4.3.1, um grafo sinalizado H é um grafo de rede se e somente se existe um conjunto W de vértices de H tal que $\mu(H^W) = 0$.

Lema 4.3.3 *Toda árvore sinalizada* T *é um grafo de rede.*

Assim temos que para toda árvore sinalizada T, existe um conjunto $W \subseteq V$ tal que $\mu(T^W) = 0$.

O algoritmo GGMZ é uma heurística para o $maxMRPR$ que tem por base o seguinte teorema.

Teorema 4.3.2 *Para um grafo conexo sinalizado* G *sem arestas paralelas e com uma árvore geradora* T *de* G*, as seguintes afirmações são equivalentes: (a)* G *é um grafo de rede; (b)* G *é um grafo balanceado; (c)* G *não possui um ciclo com um número ímpar de arestas negativas; (d) se* $W \subseteq V(G)$ *tal que* $\mu(T^W) = 0$ *então* $\mu(G^W) = 0$.

Assim, uma heurística rápida para encontrar W que maximize o maior conjunto independente em $\left(G(A)^W\right)^-$, seria encontrar W que maximize $\left(G'(A)^W\right)^-$, sendo $G'(A)$ um subgrafo de $G(A)$, apenas removendo-se algumas arestas deste último, tal que $G'(A)$ forme um grafo de rede.

Assim, podemos fazer $G'(A)$ como uma árvore geradora T de $G(A)$. Desta maneira, basta construirmos W de tal forma que $\mu(T^W) = 0$.

Dito isto, os passos básicos de GGMZ podem ser resumidos como abaixo.

1. Construir um grafo por linha $G = (V, E, sinal)$, baseado na matriz de restrições A (algoritmo 2).
2. Encontrar uma floresta geradora T em G.
3. Computar $W \subseteq V$ tal que $\mu(T^W) = 0$ (Algoritmo 5).
4. Achar o maior conjunto independente possível I no grafo G^{W^-}.

Os vértices de I correspondem às linhas de A que formam uma MRPR.

Podemos perceber que diferentes configurações de T retornam diferentes conjuntos W, que influenciam na MRPR retornada. Em [16] existe um estudo sobre diferentes tipos de árvores utilizadas e a MRPR retornada pelo GGMZ. A conclusão relatada neste trabalho foi que a construção da floresta que apresentou melhores resultados foi obtê-la através de uma busca em profundidade com prioridade nos nós de menor grau. Porém, ao aumentarmos a densidade da matriz de maneira aleatória, verificou-se que a construção mais vantajosa foi obter a floresta através de uma busca em largura com prioridade nos nós de menor grau.

Entretanto, GGMZ demora um tempo muito pequeno para ser executado mesmo se repetido para todos os tipos de árvores testados em [16] (menos de um segundo na maior parte dos casos testados). Assim, optamos por utilizar a maior MRPR retornada entre todas essas tentativas.

Algoritmo 5: ConstroiConjuntoW(W, T, v)

Entrada: árvore T e vértice v
Saída: conjunto W de vértices

```
1  para cada filho f de v em T faça
2  |  se sinal(f, v) = + e v ∈ W então
3  |  |  W ← W ∪ {f};
4  |  senão
5  |  |  se sinal(f, v) = − e v ∉ W então
6  |  |  |  W ← W ∪ {f}
7  |  |  fim se
8  |  fim se
9  |  W ← ConstroiConjuntoW(W, T, f);
10 fim para cada
11 retorna W;
```

Por motivos de simplificação iremos denotar por $(i, j, s), s = \{+, -\}$ a aresta (i, j) que tenha o sinal s.

A contrução de $(G^W)^-$ utilizado na heurística proposta (GGMZ), mostrada no algoritmo 7 abaixo, é dada pelo algoritmo 6.

Algoritmo 6: ConstroiGrafoGˆWˆ-(G, W)

Entrada: Grafo $G = (V, E)$, conjunto W
Saída: Grafo $(G^W)^-$

```
1  para cada aresta (i, j) de G faça
2      se i e j estão em conjuntos diferentes então
3          se sinal de (i, j) = − então
4              Retire a aresta (i, j, −) de G;
5          senão
6              Altere a aresta (i, j, +) para (i, j, −);
7          fim se
8      senão
9          se sinal de (i, j) = + então
10             Retire a aresta (i, j, +) de G;
11         fim se
12     fim se
13 fim para cada
14 retorna G;
```

Algoritmo 7: Heurística para a maxMRPR com elementos em $\{0, 1, -1\}$ (GGMZ)

Entrada: matriz A com elementos em $\{0, 1, -1\}$
Saída: conjunto de linhas TU que formam uma MRPR
Dados: Grafo G, árvore T, conjunto W, conjunto I

```
1 G ← ConstroiGrafoLinha(A);
2 T ← ConstroiArvoreGeradora(G);
3 v ←raiz de T;
4 W ← ConstroiConjuntoW(W, T, v);
5 G ← ConstroiGrafoGˆWˆ-(G, W);
6 I ← ConjuntoIndependente(G);
7 retorna I
```

Note que as funções $ConstroiArvoreGeradora(G)$, que constrói uma árvore geradora de G e $ConjuntoIndependente(G)$, que tenta encontrar o maior conjunto independente em G, podem ser implementadas de diversas maneiras e levar, assim, a diferentes conjunto I. Na literatura podemos encontrar inúmeros algoritmos para cada um desses problemas.

4.3.2 MRPR e o Problema do Conjunto Independente

Note que na matriz de coeficientes do SPack, temos apenas coeficientes não-negativos. Portanto, a heurística descrita na seção anterior não é específica para esta situação já que se aplica a matrizes com coeficientes em $\{-1, 0, 1\}$. Assim, a seguir estão descritos duas heurísticas para se encontrar a $maxMRPR$ para o caso de coeficientes no conjunto $\{0, 1\}$. Estas heurísticas foram projetadas pelos graduandos do IC-UNICAMP Rafael Forte Araújo Cavalcanti e João Marcos da Cunha Silva, ambos durante o período de suas iniciações Científicas orientadas pelo professor Cid Carvalho de Souza. Grande parte dos resultados apresentados a seguir foram retirados de [4] e [12].

Lema 4.3.4 *Seja A uma matriz $m \times n$ contendo apenas elementos de valor em $\{0, 1\}$ e $G(A) = (V, E, s)$ o grafo sinalizado associado a A. Então $G(A)$ possui somente arestas negativas e, para todo $W \subseteq V$, as arestas em $(G^W)^-$ são as mesmas de $G(A)$, excluídas as arestas no corte $\delta(W, \overline{W})$.*

Prova: Por contradição, considere uma aresta positiva $(i, j) \in E$. Pela definição de grafo sinalizado, $\exists\ k \in \{1, \ldots, n\}$ tal que $a_{ik} = -a_{jk} \neq 0$ na matriz original. Mas, se a_{ik} e a_{jk} são não-nulos, eles somente podem assumir valor 1. Portanto, por definição, a aresta (i, j) tem que ser negativa. Agora, pela forma de construção de $(G^W)^-$, inverte-se o sinal das arestas do corte $\delta(W, \overline{W})$ que, neste caso, passam a ser positivas sendo então removidas. □

Como a matriz A possui elementos apenas de valores 0 ou 1, o problema de encontrar o valor de $\overline{\alpha}(G(A))$ está relacionado ao $maxMRPR$ através do seguinte teorema.

Teorema 4.3.3 *O problema $maxMRPR$ restrito a matrizes que possuem apenas elementos em $\{0, 1\}$ é o mesmo que encontrar dois conjuntos independentes disjuntos no grafo sinalizado $G(A)$ tal que a soma de seus tamanhos seja máxima.*

Prova: Seja $\overline{\alpha}(G(A))$ o valor da solução do $maxMRPR$ dado por H, um conjunto independente de $(G^W)^-$ tal que $\alpha((G^W)^-) = |H| = \overline{\alpha}(G(A))$. Seja ainda z o valor da solução do problema de dois conjuntos independentes disjuntos de soma de cardinalidades máximas, tendo uma solução ótima H_1 e H_2, ou seja, $z = |H_1| + |H_2|$.

- $\overline{\alpha}(G(A)) \leq z$: Seja $H_1{}' = H \cap W$ e $H_2{}' = H \cap \overline{W}$. Como $H_1{}'$ e $H_2{}'$ são subconjuntos de um conjunto independente de $(G^W)^-$, eles também são conjuntos independentes de $(G^W)^-$. Por outro lado, do lema 4.3.4, dados dois vértices quaisquer em $W(\overline{W})$, a aresta entre eles existe em $(G^W)^-$ se e somente se ela existe em $G(A)$. Logo $H_1{}'$ e $H_2{}'$ são conjuntos independentes em $G(A) \Rightarrow \overline{\alpha}(G(A)) = \alpha((G^W)^-) = |H_1{}'| + |H_2{}'| \leq z$.

- $\overline{\alpha}(G(A)) \geq z$: Seja H_1 e H_2 um par ótimo de conjuntos independentes de $G(A)$ de modo que $z = |H_1| + |H_2|$ e impomos que $W' = H_1$. Calculando $(G^W)^-$, as únicas arestas que sobram são aquelas entre os vértices de $\overline{W'} = V \setminus H_1$. Então, como H_2 é um conjunto independente de $G(A)$, H_2 também deve ser um conjunto independente de $(G^W)^-$. Além disso, do lema 4.3.4, os vértices de H_2 não estão conectados aos de H_1 em $(G^W)^-$.

Assim, $z = |H_1| + |H_2| \leq \alpha((G^{W'})^-) \leq \overline{\alpha}(G(A))$. □

Teorema 4.3.4 *Seja A uma matriz com elementos em $\{0, 1\}$. Uma submatriz de rede pura pode ser obtida a partir de uma matriz de rede pura refletida, A^r de A, através da aplicação da operação de reflexão nas linhas associadas aos vértices de um dos conjuntos independentes do grafo sinalizado $G(A) = (V(A), E(A), s)$, definidos no teorema 4.3.3.*

Prova: Os conjuntos independentes H_1 e H_2 tem cada vértice seu associado a uma linha na matriz A. As linhas associadas a H_1 formam uma submatriz A_1 de A. Essa matriz possui no máximo uma entrada 1 por coluna, não havendo nenhuma aresta entre quaisquer vértices de H_1 em $G(A)$ ($H_1 \subseteq G(A)$ é conjunto independente), o que significa que $\nexists\, i, j \in A_1$ *tq* $a_{ik} = a_{jk} \neq 0$. Analogamente, pode-se associar H_2 a uma submatriz A_2 de A. A submatriz A^r formada pelas linhas das matrizes A_1 e A_2 possui no máximo duas entradas 1 por coluna (uma associada a A_1 e outra associada a A_2). A construção de uma matriz de rede pura pode ser feita a partir de A^r aplicando uma operação de reflexão em cada linha da matriz A^r que pertence a A_2. Claramente a definição de matriz de rede pura está respeitada já que para cada coluna há no máximo uma entrada -1, associada a A_2 e no máximo uma entrada 1 associada a A_1. A^r então é uma matriz de rede pura refletida. □

Vamos denotar o problema de se encontrar dois conjuntos independentes disjuntos em um grafo cuja soma de suas cardinalidades sejam máximas de $max2IS$.

Durante o período de desenvolvimento das iniciações científicas citadas anteriormente, foram feitos testes comparando os desempenhos de algoritmos com o desempenho de GGMZ, para problemas onde os coeficientes estavam apenas em $\{0, 1\}$, que é o caso deste projeto. Foi relatado que estas heurísticas mais específicas para as instâncias utilizadas geram resultados melhores dos que as obtidas pelo GGMZ.

Segue abaixo uma breve descrição dos algoritmos implementados pelo Rafael Cavalcanti e pelo João Silva.

Relaxacão Lagrangeana

O trabalho de Rafael Cavalcanti propõe resolver de forma exata o $max2IS$, através de uma relaxação Lagrangeana. Um esboço do algoritmo resultante da aplicação desta técnica é dado a seguir:

Algoritmo 8: Relaxação Lagrangeana para o maxMRPR

1 Encontre uma formulação em PLI para o problema.
2 Identifique um subconjunto das restrições da formulação que ao ser removido do modelo leva a um problema relaxado que pode ser resolvido facilmente.
3 Mova as restrições complicadoras para a função objetivo multiplicando-as por um vetor de penalizações conhecido como multiplicadores de Lagrange.
4 **enquanto** *critério de parada não for satisfeito* **faça**
5 | Resolva de maneira exata o problema relaxado.
6 | Atualize o vetor de multiplicadores de Lagrange.
7 **fim enqto**

Suponha que a formulação seja dada da seguinte maneira:

$$\begin{aligned} z &= \max c^T x \\ \text{s. a.} \quad Ax &\geq b \qquad (4.2) \\ Bx &\geq d \qquad (4.3) \\ x &\in \mathbb{Z}^n_+. \end{aligned}$$

A solução gerada no último passo é um limitante superior para o valor ótimo do problema. Assim, se considerarmos o problema original, a aplicação da Relaxação Lagrangeana poderia ser feita da seguinte forma: inicialmente devemos escolher um subconjunto das restrições e atribuir a elas custos de penalização, que irão formar o conjunto dos multiplicadores de Lagrange associados na FO, de modo que o novo problema seja de fácil resolução. Assim, suponha que a equação (4.3) represente as restrições complicadoras do problema. Então, relaxando-as e depois fazendo as penalizações na função objetivo teremos o seguinte sub-problema lagrangeano:

$$\begin{aligned} z1(u) &= \max c^T x + u(d - Bx) \\ \text{s. a.} \quad Ax &\geq b \qquad (4.4) \\ x &\in \mathbb{Z}^n_+ \end{aligned}$$

onde u é o vetor dos multiplicadores de Lagrange, e $L(x,u) = c^T x + u(d - Bx)$ é chamada *Função Lagrangeana*. Temos assim um problema mais simples de ser resolvido. Se as

penalizações correspondentes ao vetor u forem não-negativas, pode-se mostrar que $z1(u) \geq z$, ou seja, $z1(u)$ será um limite superior para z. Em um cenário ideal, a escolha dos mulitiplicadores de Lagrange deve ser efetuado de tal forma que se obtenha o menor limitante superior possível, ou seja, devemos resolver o seguinte problema:

$$\begin{array}{rrcl} \min\limits_{u \geq 0} & w & = & \max cx + u(d - Bx) \\ \text{Sujeito a} & Ax & \geq & b \\ & x & \in & \mathbb{Z}_+^n \end{array} \tag{4.5}$$

Este problema é chamado de *Dual Lagrangeano*. Idealmente, o valor ótimo do problema Dual Lagrangeano deve ser igual ao valor ótimo do PLI original. Mas, em geral, esses dois valores são diferentes e, neste caso, observamos um *salto de dualidade* entre eles, que é a diferença entre os dois valores ótimos, ou ainda $w - z$.

As definições abaixo são utilizadas na descrição da relaxação Lagrangeana implementada neste projeto:

- V é o conjunto de vértices do grafo $G(A)$ com $|V| = n$ e esses vértices são identificados cada um por $i = 1, \ldots, n$.
- C é o conjunto de cliques correspondente a uma cobertura de cliques maximais em relação às arestas efetuada em $G(A)$ com $|C| = c$. Cada clique é identificada pelo seu índice l, de modo que $C = \{C_1, \ldots, C_c\}$.
- $x_{ik} = 1$ caso o vértice i pertença ao conjunto independente $k = 1, 2$.

Assim podemos definir a formulação para o max2IS da seguinte maneira:

$$z = \max \sum_{k=1}^{2} \sum_{i=1}^{n} x_{ik} \tag{4.6}$$

$$\sum_{i \in C_l} x_{ik} \leq 1 \text{ para } l = 1, \ldots, c \text{ e } k = 1, \ldots, 2 \tag{4.7}$$

$$x_{i1} + x_{i2} \leq 1 \text{ para } i = 1, \ldots, n \tag{4.8}$$

$$\sum_{i=1}^{n} x_{i1} \leq \sum_{i=1}^{n} x_{i2} \tag{4.9}$$

$$x_{ik} \in \mathbb{B} \text{ para } i = 1, \ldots, n \text{ e } k = 1, \ldots, 2. \tag{4.10}$$

Na verdade, a restrição (4.9) não é estritamente necessária para o modelo. Ela visa apenas eliminar parte da simetria ao impor que um dos conjuntos independentes seja

maior que o outro (nesse caso, o segundo). Removendo-se essa restrição temos o seguinte modelo para o max2IS:

$$
\begin{aligned}
z &= \max \sum_{k=1}^{2} \sum_{i=1}^{n} x_{ik} && (4.11)\\
x_{i1} + x_{i2} &\leq 1 \text{ para } i = 1, \ldots, n && (4.12)\\
\sum_{i \in C_l} x_{ik} &\leq 1 \text{ para } l = 1, \ldots, c \text{ e } k = 1, \ldots, 2 && (4.13)\\
x_{ik} &\in \mathbb{B} \text{ para } i = 1, \ldots, n \text{ e } k = 1, \ldots, 2. && (4.14)
\end{aligned}
$$

Utilizando a notação matricial, é obtida a seguinte formulação equivalente (com (4.11), (4.12), (4.13) equivalentes, respectivamente, a (4.15), (4.16), (4.17)) :

$$
\begin{aligned}
z &= \max x && (4.15)\\
Ax &\leq b && (4.16)\\
Bx &\leq d && (4.17)\\
x &\in \mathbb{B}^{2n} && (4.18)
\end{aligned}
$$

onde x_i é a variável associada ao vértice i no primeiro conjunto independente e x_{i+n} é a variável associada ao vértice i no segundo conjunto independente.

Aplicando a técnica de relaxação lagrangeana, dualiza-se as restrições (4.17) chegando-se ao seguinte problema lagrangeano:

$$
\begin{aligned}
z1(u) &= \max x + u(d - Bx) \\
Ax &\leq b && (4.19)\\
x &\in \mathbb{B}^{2n}
\end{aligned}
$$

onde u é o vetor dos multiplicadores de Lagrange e $L(x, u) = x + u(d - Bx)$ é a *função lagrangeana*. Claramente, fixado o vetor u, esse problema é facilmente resolvido por inspeção. O dual lagrangeano, que leva ao vetor u que gera o melhor limitante superior possível de ser obtido pelo método, pode ser resolvido através do clássico método de otimização por subgradientes.

GRASP

O trabalho do João Silva propõe resolver o max2IS utilizando uma meta-heurística de GRASP (*Greedy Randomized Adaptative Search Procedure*) baseado em [15].

O GRASP é uma meta-heurística geral para problemas combinatórios, em que cada iteração consiste, basicamente, em duas fases: construção e busca local. Na construção,

monta-se uma solução viável por meio de alguma estratégia gulosa. Já na busca, a vizinhança dessa solução é investigada até um máximo (ou mínimo) local ser encontrado. Em cada iteração compara-se a solução encontrada na fase de busca anterior com a atual, e guarda-se a melhor. As iterações se repetem até que um limite no número de iterações, passado como parâmetro de entrada, seja atingido.

No algoritmo 9 está mostrado o algoritmo genérico de construção do GRASP (*ConstrucãoAleatóriaGulosa(S)*). Trata-se de um algoritmo que utiliza uma estratégia gulosa modificada de modo que, em uma iteração, não escolhe-se necessariamente para ingresso na solução o elemento que melhor satisfaz à função critério f. A escolha é feita de forma aleatória em uma lista restrita de candidatos (*LRC*) contendo um pequeno subconjunto dos melhores elementos de acordo com a função f. Uma vez feita esta escolha, um elemento é retirado da LRC e é incorporado à solução. Em função disso, os custos de inserção dos elementos restantes na solução são reavaliados, e a LRC atualizada. Esse é o aspecto adaptativo da heurística.

Algoritmo 9: ConstrucãoAleatóriaGulosa(conjunto S)

Entrada: Conjunto S dos elementos candidatos a entrarem na solução
Saída: Conjunto *solucao* de elementos

```
1  solucao ← ∅;
2  para cada s ∈ S faça
3  |  Avalia-se o custo de incremento de f(s);
4  fim para cada
5  enquanto solucao não está completa faça
6  |  Constrói-se a lista restrita de candidatos LRC;
7  |  Seleciona-se um elemento s da LRC aleatoriamente;
8  |  solucao ← solucao ∪ {s};
9  |  S ← S \ {s};
10 |  para cada s ∈ S faça
11 |  |  Reavalia-se o custo de incremento de f(s);
12 |  fim para cada
13 fim enqto
14 retorna solucao;
```

Após construída uma solução, inicia-se a fase de busca na sua vizinhança. A vizinhança de uma solução é definida por algum tipo de perturbação pré-fixada que é feita sobre ela (por exemplo utilizando uma troca entre um elemento presente e outro ausente na solução). Esse processo está descrito na função *BuscaLocal(solucao)* do algoritmo 10.

O algoritmo de busca trabalha de maneira iterativa. Ele adota uma estratégia clássica de descida (ou subida), cessando quando não é encontrada outra solução melhor na vizinhança V.

Algoritmo 10: BuscaLocal(*solucao*)

Entrada: Conjunto *solucao* representando uma solução válida, conjunto V formado pela vizinhança de *solucao*
Saída: Conjunto *solucao* alterado

```
1 enquanto solucao não é uma solução ótima local faça
2 |  Acha s' ∈ V, com f(s') < f(solucao);
3 |  solucao ← s';
4 fim enqto
5 retorna solucao;
```

O procedimento completo do GRASP é descrito no algoritmo 11.

Algoritmo 11: GRASP

Entrada: Funçao f, conjunto S de variáveis, e o número máximo de iterações max
Saída: Conjunto *melhorSolucao*

```
1  melhorSolucao ← ∅;
2  para k ← 1 até max faça
3  |  solucao ← ConstruçãoAleatóriaGulosa(S);
4  |  V ← vizinhança(solucao);
5  |  solucao ← BuscaLocal(solucao, V );
6  |  se solucao melhor que melhorSolucao então
7  |  |  melhorSolucao ← solucao;
8  |  fim se
9  fim para
10 retorna melhorSolucao;
```

Para o GRASP associado ao max2IS foram implementadas três estratégias para a fase de construção aleatória gulosa de uma solução:

1. Estratégia 1: Inicialmente procura-se encontrar um conjunto independente máximo do grafo original e, em seguida, procura-se encontrar o conjunto independente máximo do grafo induzido por todos os vértices que não estão no primeiro conjunto.

2. Estratégia 2: a construção dos dois conjuntos independentes é realizada simultaneamente. Após escolher um vértice para o primeiro conjunto independente, considera-se os vizinhos desse vértice que têm os menores graus naquele momento, que são admissíveis para o segundo conjunto independente, (a quantidade de vizinhos considerados depende do tamanho da instância), e escolhe-se, aleatoriamente, um desses vértices para integrar o segundo conjunto independente. Uma vez que não exista

mais vértices admissíveis para o primeiro conjunto independente, verifica-se se ainda há algum admissível para o segundo. Se houver, escolhe-se o de menor grau dentre esses, faz-se as atualizações necessárias, e repete-se o procedimento até acabar os vértices admissiveis também para o segundo conjunto.

3. Estratégia 3: a construção dos conjuntos independentes também é realizada simultaneamente. A diferença dessa implementação para a anterior está no seguinte fato: após escolher um vértice para o primeiro conjunto independente, os vértices que passam a formar uma lista, para posterior escolha aleatória de um deles, não precisam mais ser vizinhos do vértice que acabou de integrar o primeiro conjunto independente. Assim, são considerados os vértices de menor grau do grafo inteiro.

Já a busca local foi feita através dos seguintes passos:

1. Remove-se dois vértices, v_1 e v_2 de S, o conjunto independente resultante da fase de construção do GRASP. Seja S' o conjunto assim obtido.

2. Seja Adm o conjunto de vértices admissíveis relativos a S'. Note que Adm contém os dois vértices retirados de S.

3. Aplica-se um procedimento (exato) de busca para achar um conjunto independente *máximo* N no grafo induzido por Adm no grafo original (normalmente muito pequeno).

4. Se $|N| > 2$, atualiza-se $S = S' \bigcup N$. Note que nesse caso o tamanho do conjunto independente aumenta. A busca local termina quando todos os pares de vértices de S foram testados e nenhuma melhora é possível, ou quando encontra-se uma melhora.

CLIQUER

Como, potencialmente, o comportamento do algoritmo proposto aqui é fortemente dependente da maior matriz TU encontrada, já que é ela quem vai definir quem serão as QTUs utilizadas para a obtenção dos cortes, podemos supor que uma escolha diferente desta matriz resulte em uma mudança no desempenho do algoritmo. Portanto, talvez compense encontrar uma TU maior, mesmo as custas de um maior esforço computacional.

Para testarmos esta afirmação utilizamos o programa CLIQUER que é composto de uma série de subrotinas em `C`, desenvolvidas por Patric Östergård, para encontrar cliques de tamanho máximo em um grafo arbitrário, com peso nas arestas. Em

`http://users.tkk.fi/pat/cliquer.html`,

podem ser encontrados a descrição do CLIQUER, bem como, o seu manual e os arquivos para serem baixados.

Tal ferramenta foi utilizada para encontrar dois ISs maximais para o grafo por linha G do problema, do mesmo modo que a *Estratégia 1* da fase de construção do GRASP descrita anteriormente. Para isso, descobrimos qual a clique máxima C do grafo complementar $\bar{G}$ de G, ou seja, qual o maior conjunto independente em G. Em seguida fizemos $G = G \setminus C$ e novamente encontramos qual a maior clique C_2 de $\bar{G}$.

Assim obtivemos dois conjuntos independentes disjuntos, ou seja, uma solução para o max2IS. Note que apesar de cada um dos conjuntos ser um conjunto independente máximo no grafo em que eles foram encontrados, a solução calculada não é necessariamente a solução ótima para o max2IS e, conseqüentemente, para o maxMRPR.

Na tabela 4.5 pode-se comparar a qualidade das soluções alcançadas pela heurística GGMZ utilizada neste projeto, com aquelas obtidas usando o CLIQUER, como descrito acima. Estes testes foram realizados com as instâncias de i73 a i106, e apenas explicitamos os resultados em que o CLIQUER conseguiu terminar em apenas algumas horas. O tempo gasto pela heurística GGMZ foi omitida da tabela, pois este pode ser considerado insignificante.

Percebe-se claramente que o CLIQUER obteve resultados significativamente melhores que os alcançados pela heurística. Porém, nosso intuito com estes experimentos não era, eventualmente, decidir pelo uso do CLIQUER como um método para encontrar uma solução para o maxMRPR. Além dele ser extremamente demorado, já que é um algoritmo exato para encontrar cliques, um problema NP-completo, ele não necessariamente encontra a solução ótima da maior matriz de rede pura refletida. Mas estes testes servem para avaliar a qualidade da solução encontrada pela heurística GGMZ que foi empregada aqui.

Podemos perceber que se investíssemos mais nesta etapa do processo poderíamos obter TUs maiores. Porém, como veremos mais para frente, concluímos que tal esforço não valeria a pena, justificando a decisão de mantermos o uso da heurística GGMZ.

4.4 Algoritmos de resolução para o SPack

Nesta seção discutiremos os resultados obtidos com os algoritmos utilizados na resolução do SPack.

4.4.1 XPRESS padrão

Em sua configuração padrão, o XPRESS utiliza mais artifícios do que simplesmente resolver um problema linear inteiro através de um B&B com relaxação linear. Na verdade, o pacote inclui uma série de subrotinas auxiliares que realizam tarefas como, por exemplo,

nome	#TU		tempo
	cliquer	heurística	cliquer
i78	87	74	16623
i79	67	57	467
i80	71	61	505
i85	49	41	5415
i86	46	36	45
i87	49	43	170
i88	56	47	21892
i89	35	28	7
i90	56	48	4574
i91	33	27	2
i92	32	25	3
i94	37	30	11
i95	35	30	15
i96	23	19	0
i97	43	36	142
i98	41	36	165
i99	27	22	1
i100	27	22	1
i101	45	38	937
i102	31	24	5
i103	31	25	3
i104	47	39	14888
i105	34	28	34
i106	24	19	0

Tabela 4.5: Comparação entre o CLIQUER e a heurística GGMZ.

executar heurísticas para encontrar soluções viáveis e, conseqüentemente, melhorar o limitante primal. Apesar dessa característica do pacote, por motivo de simplificação, quando mencionarmos a resolução do problema através de um B&B, estaremos nos referindo de fato à resolução pelo XPRESS usando sua configuração padrão.

4.4.2 Branch-and-Cut

Infelizmente, no caso do SPack, encontramos apenas duas famílias de desigualdades que poderiam ser empregadas como cortes visando a melhoria do modelo linear: as cliques e os buracos ímpares. Além disso, apenas as desigualdades cliques conseguiram ser encontradas em um número razoável de instâncias (em menos de 5% das instâncias encontrou-se pelo menos um buraco ímpar que mudasse a solução corrente). Portanto, todos os algoritmos citados neste projeto em que são aplicados planos de corte, esses últimos correspondem a desigualdades clique.

Assim, a maneira que encontramos para melhorar o desempenho do algoritmo de planos de corte com tão poucas opções foi variar as estratégias de separação e adição das desigualdades clique. Uma dessas formas dá origem a um algoritmo *branch-and-cut*, aqui denotado por B&C.

Em um algoritmo B&C, como visto na seção 2.1.4, a cada nó da árvore de enumeração aplica-se um algoritmo de planos de corte puro (sem enumeração). Para tanto, resolve-se uma relaxação linear do problema e, em seguida, faz-se uma separação que procura identificar cortes violados pela solução ótima da relaxação linear corrente. Se tais cortes forem encontrados, eles são inseridos na relaxação, repetindo-se o processo até que a separação não seja mais capaz de encontrar cortes violados. Nesse último caso, passa-se a exploração de um novo nó da árvore de enumeração.

No caso do SPack, as desigualdades que serão separadas são as desigualdades clique. Na seção 3.3 foram discutidos alguns algoritmos para separar tais desigualdades de forma heurística. Vale observar que os cortes retornados por tais rotinas dependem essencialmente da solução fracionária recebida na entrada.

A tabela 4.6 mostra os resultados obtidos para as instâncias de i42 a i72 pela nossa implementação do B&C descrita acima ao ser executado na máquina *M1*. A estratégia adotada foi a de incluir todas as restrições violadas encontradas para cada solução da relaxação linear. Infelizmente, como se percebe, nenhuma dessas instâncias foram resolvidas em um tempo menor que 1800 segundos, o tempo limite que estipulamos.

O motivo principal deste comportamento é que gastamos muito tempo procurando restrições violadas que, apesar de tornarem inviável a solução ótima da relaxação linear corrente, não melhoraram muito a formulação. Um indício que nos levou a essa conclusão é o pequeno número de nós abertos durante a enumeração, que pode ser observado na

coluna "#nos" da tabela 4.6.

Outra característica que deve ser levada em conta é o fato de que cada novo corte dá origem à uma nova restrição na formulação. Portanto, ao inserir todos cortes violados que encontramos e iterar o algoritmo de planos de corte até que a separação falhe, podemos acabar criando um modelo linear muito grande que exigirá um maior tempo de processamento para ser resolvido.

Pensando nisso, usando a mesma máquina anterior, testamos também versões do B&C com a adição de um número limitado de cortes por nó. Na tabela 4.7 podemos comparar os resultados obtidos, dentro de um limite de tempo de 900 segundos, para o B&B e para o B&C restrito à inserção de uma ou dez cliques violadas por iteração. Nesta tabela apresentamos para cada entrada o gap de dualidade no momento em que o processamento foi interrompido, o número de nós abertos e tempo total gasto para cada um dos três algoritmos.

Apesar de não ter sido verificado nenhuma melhora com relação ao B&B, podemos perceber que adicionando-se apenas uma clique e reotimizando o problema, o gap obtido ao final de 900 segundo foi, em geral, melhor do que adicionando-se 10, assim como o número de nós analisados foi maior.

Em vista dos resultados anteriores, passamos a investigar a possibilidade de usar as matrizes QTU de uma outra forma diferente daquela que havíamos imaginado originalmente. Nesse sentido, pareceu-nos natural o uso de mútltiplas matrizes TU ao invés de uma única. Assim, chegamos a uma outra implementação do método de planos de corte onde a busca por desigualdades violadas é feita da seguinte forma. Seja M a maior submatriz MRPR encontrada pela heurística GGMZ na matriz de restrições original $A = A_0$. Retiramos de A_0 as linhas em M, formando a matriz A_1, e tentamos novamente resolver o maxMRPR de A_1 através da mesma heurística. Repetimos este processo até que um critério de parada seja satisfeito. Obtemos assim, ao invés de uma única, um conjunto de submatrizes MRPR. Com isso, a busca por desigualdades violadas na fase de separação, é repetida para cada uma dessas submatrizes TU separadamente. O intuito é que, através disto, seja possível buscar uma violação sobre um número muito maior de cortes, aumentando-se, assim, a probabilidade de melhorar a formulação fazendo com que o algoritmo como um toda convirja mais rapidamente.

Percebe-se que, pelo método proposto anteriormente, o tamanho das submatrizes TU encontradas vão diminuindo a cada iteração, já que a matriz na qual elas são procuradas são sempre submatrizes próprias das matrizes das iterações anteriores. Também pensando nesse fato, sugerimos critérios de parada no processo de geração de matrizes TU que eram baseados no tamanho e na quantidade total de submatrizes MRPRs que é gerada. Dentre esses critérios, o que acabou retornando o maior número de MRPRs, foi aquele baseado estritamente no tamanho da submatriz retornada pela heurística. Nele, a busca por

inst.	#nos	tempo	gap (%)
i42	324	1800	10.9
i43	80	1800	19.6
i44	281	1800	17.8
i45	132	1800	18.3
i46	78	1800	21.3
i47	40	1800	21.3
i48	330	1800	17.3
i49	186	1800	23.5
i50	68	1800	20.0
i51	46	1800	19.9
i52	24	1800	20.7
i53	249	1800	20.5
i54	54	1800	22.5
i55	24	1800	20.2
i56	21	1800	18.9
i57	13	1800	20.4
i58	25	1800	77.4
i59	14	1800	74.7
i60	16	1800	69.4
i61	5	1800	65.5
i62	4	1800	63.9
i63	3	1800	58.1
i64	1	1800	61.3
i65	18	1800	73.9
i66	12	1800	59.6
i67	9	1800	61.6
i68	7	1800	66.3
i69	7	1800	57.2
i70	4	1800	75.2
i71	5	1800	60.7
i72	2	1800	56.2

Tabela 4.6: Branch-and-cut (UTU).

inst.	B&B			1 clique			10 cliques		
	gap(%)	#nos	tempo	gap(%)	#nos	total	gap(%)	#nos	total
i42	5.52	35400	900	5.00	41000	900	6.19	32600	900
i43	13.56	23500	900	12.85	25100	900	13.26	23200	900
i44	4.03	16900	900	7.92	19000	900	13.62	20800	900
i45	9.23	14200	900	12.74	14900	900	15.44	15200	900
i46	11.39	13400	900	15.79	14400	900	15.21	13900	900
i47	14.01	11700	900	15.49	12100	900	14.67	11500	900
i48	0.93	13000	900	0.95	13200	900	10.11	15000	900
i49	7.38	9600	900	8.17	10200	900	12.11	10900	900
i50	11.01	8400	900	12.10	8600	900	14.01	8100	900
i51	14.34	7500	900	14.44	7500	900	14.47	6300	900
i52	16.05	6600	900	16.68	6300	900	14.58	6100	900
i53	0.00	1000	111	0.03	11400	900	5.60	8800	900
i54	10.11	5500	900	13.11	6100	900	13.06	5700	900
i55	13.74	4700	900	15.22	4900	900	17.35	4800	900
i56	15.43	4100	900	15.53	4100	900	16.66	3500	900
i57	17.31	3800	900	17.31	3800	900	17.35	2800	900
i58	0.31	1600	900	2.50	1400	900	4.72	1200	900
i59	6.26	1000	900	7.67	1000	900	10.25	900	900
i60	9.57	800	900	10.87	800	900	11.10	300	900
i61	12.30	700	900	13.74	500	900	13.63	100	900
i62	14.15	600	900	15.65	400	900	15.64	0	900
i63	16.75	600	900	16.83	100	900	16.93	0	900
i64	17.97	600	900	17.93	100	900	18.07	0	900
i65	0.00	83	60	0.00	65	27	3.34	700	900
i66	4.50	600	900	5.38	600	900	6.71	500	900
i67	8.35	500	900	8.42	400	900	10.92	0	900
i68	10.97	400	900	12.25	300	900	12.41	0	900
i69	12.14	400	900	13.73	100	900	13.99	0	900
i70	14.16	400	900	14.14	100	900	14.16	0	900
i71	15.80	300	900	15.87	0	900	15.90	0	900
i72	16.89	200	900	16.92	0	900	16.93	0	900

Tabela 4.7: Branch-and-cut para as instâncias i42 a i76.

matrizes MRPR não para enquanto o tamanho da última matriz encontrada for maior que $\frac{1}{3}$ do tamanho da primeira. Na tabela 4.8 são mostrados os tamanhos das TUs encontradas para o último bloco de instâncias (i73 a i106) ao se usar o critério mencionada acima.

Como se vê na tabela 4.8, em algumas instâncias, a primeira submatriz encontrada foi muito pequena em relação ao número de linhas da matriz de restrições original. Com isso, o número total de MRPRs geradas acabou se tornando demasiadamente grande. Note-se inclusive que, para que fosse possível explicitar seus tamanhos em uma única tabela, foi preciso agrupar informações em uma única célula. Por este motivo, utilizamos a representação $n_{(\times k)}$, quando necessária, para quando $k > 1$ submatrizes do mesmo tamanho n foram encontradas para uma certa instância. Por exemplo, para $i106$, temos uma célula cujo conteúdo é $13_{(\times 3)}$, o que significa que foram encontradas três submatrizes MRPR de tamanho 13 para essa instância.

Ao executarmos o algoritmo B&C com as múltiplas TUs, percebemos que seu desempenho também foi ruim. Estes testes foram feitos com o terceiro bloco de instâncias, e seus resultados são exibidos nas tabelas 4.9 e 4.10. A tabela 4.9 mostra os resultados obtidos ao tentarmos resolver as instâncias através de um algoritmo de B&C utilizando múltiplas TUs geradas segundo o critério de parada explicitado anteriormente. Já na tabela 4.10, adicionamos ao critério de parada a restrição de que poderiam ser geradas no máximo 3 submatrizes TU.

Como podemos ver, em ambas abordagens, não conseguimos resolver nenhuma instância no limite de tempo estipulado. Porém, os resultados apresentados na tabela 4.9 são piores do que aqueles mostrados na tabela 4.10, se considerarmos os gaps obtidos após 1800 segundos. Novamente isto se deve ao fato de termos demorado muito tempo encontrando cortes em cada nó, como pode ser inferido pelo número de nós abertos. Com isto concluímos que nossos esforços em aumentar o número de cortes não refletiu em sua qualidade, fazendo com que o algoritmo não apresentasse nenhuma melhora.

Os resultados mostrados até aqui indicam que o algoritmo de B&C baseado nas QTUs não obteve nenhum ganho com relação ao B&B. Ficou bastante claro que para o algoritmo ter um bom desempenho precisamos equilibrar bem os esforços gastos para encontrar cortes e para resolver o SPack de fato. Um caminho que nos pareceu razoável de ser tentado nessa direção foi utilizar um algoritmo *cut-and-branch*, ou seja, um algoritmo onde a busca por desigualdades violadas fica restrita ao primeiro nó da enumeração. É sobre isso que iremos discutir agora.

4.4.3 Cut-and-branch

Para os testes descritos nesta subseção, utilizaremos as instâncias i73 a i106. Todos os resultados reportados foram obtidos em experimentos realizados na máquina *M2*. Na

nome	tamanho das TUs													
i73	133	69	–	–	–	–	–	–	–	–	–	–	–	–
i74	116	73	47	–	–	–	–	–	–	–	–	–	–	–
i75	90	71	48	35	–	–	–	–	–	–	–	–	–	–
i76	80	57	42	30	–	–	–	–	–	–	–	–	–	–
i77	93	65	40	–	–	–	–	–	–	–	–	–	–	–
i78	74	52	38	–	–	–	–	–	–	–	–	–	–	–
i79	57	48	34	23	–	–	–	–	–	–	–	–	–	–
i80	61	47	35	22	–	–	–	–	–	–	–	–	–	–
i81	65	60	50	38	33	27	24	–	–	–	–	–	–	–
i82	76	61	48	38	28	–	–	–	–	–	–	–	–	–
i83	75	60	48	38	27	–	–	–	–	–	–	–	–	–
i84	78	65	53	38	32	–	–	–	–	–	–	–	–	–
i85	41	37	35	29	27	24	22	22	20	16	14	–	–	–
i86	36	33	27	24	21	19	16	16	13	–	–	–	–	–
i87	43	36	29	24	23	21	19	16	–	–	–	–	–	–
i88	47	42	36	30	27	26	19	15	–	–	–	–	–	–
i89	28	26	23	21	20	18	18	16	14	14	13	12	10	9
i90	48	42	35	30	27	23	20	16	–	–	–	–	–	–
i91	27	24	22	20	18	18	17	17	14	14	12	13	11	–
i92	25	24	21	21	19	18	17	16	14	13	12	11	$10_{(x2)}$	$8_{(x3)}$
i93	50	42	35	33	29	24	21	19	16	–	–	–	–	–
i94	30	27	26	23	19	19	18	17	16	14	12	12	12	$10_{(x2)}$
i95	30	26	23	23	20	19	18	17	15	14	13	13	$11_{(x2)}$	10
i96	19	17	16	16	$15_{(x2)}$	$14_{(x2)}$	$13_{(x2)}$	12	$11_{(x2)}$	$10_{(x2)}$	$9_{(x2)}$	$8_{(x4)}$	7	$6_{(x2)}$
i97	36	32	29	27	23	21	21	17	16	14	14	13	12	–
i98	36	31	27	27	23	22	20	17	16	16	14	12	–	–
i99	22	21	19	19	17	$16_{(x3)}$	15	14	14	13	$12_{(x2)}$	$10_{(x2)}$	$9_{(x2)}$	$7_{(x4)}$
i100	22	21	20	18	18	16	$15_{(x3)}$	14	$13_{(x2)}$	$12_{(x2)}$	11	$10_{(x2)}$	9	$7_{(x4)}$
i101	38	35	30	28	24	24	21	19	18	15	15	15	12	–
i102	24	22	20	$19_{(x3)}$	18	17	16	$14_{(x2)}$	$13_{(x2)}$	$12_{(x2)}$	$11_{(x2)}$	10	9	$8_{(x2)}$
i103	25	24	19	19	18	18	16	16	$14_{(x4)}$	13	13	$12_{(x2)}$	$10_{(x2)}$	$8_{(x3)}$
i104	39	36	33	28	26	25	23	21	19	17	16	14	13	13
i105	28	$23_{(x2)}$	22	20	19	18	$17_{(x2)}$	16	15	14	$13_{(x3)}$	12	$10_{(x2)}$	$9_{(x2)}$
i106	19	18	16	16	$15_{(x2)}$	$14_{(x2)}$	$13_{(x3)}$	$12_{(x3)}$	$11_{(x2)}$	$10_{(x2)}$	$9_{(x4)}$	$8_{(x2)}$	$7_{(x2)}$	$6_{(x3)}$

Tabela 4.8: Tamanho das múltiplas TUs.

inst.	#nos	tempo	gap (%)
i73	1	1800	8.2
i74	1	1800	14.0
i75	1	1800	12.8
i76	1	1800	14.3
i77	1	1800	12.9
i78	1	1800	7.0
i79	1	1800	16.1
i80	1	1800	20.3
i81	1	1800	31.9
i82	1	1800	24.4
i83	1	1800	23.8
i84	1	1800	19.7
i85	4	1800	31.7
i86	1	1800	20.5
i87	1	1800	34.9
i88	1	1800	30.8
i89	1	1800	44.0
i90	1	1800	30.5
i91	1	1800	45.6
i92	1	1800	41.3
i93	1	1800	34.5
i94	1	1800	47.6
i95	1	1800	36.9
i96	1	1800	36.7
i97	1	1800	34.0
i98	1	1800	46.9
i99	1	1800	60.8
i100	1	1800	51.1
i101	1	1800	56.6
i102	1	1800	42.4
i103	1	1800	34.7
i104	1	1800	39.7
i105	1	1800	62.1
i106	1	1800	42.3

Tabela 4.9: Branch-and-cut com múltiplas TUs.

inst.	#nos	tempo	gap (%)
i73	5502	1800	1.9
i74	5195	1800	3.2
i75	1271	1800	6.9
i76	5766	1800	7.5
i77	4568	1800	4.0
i78	2320	1800	4.2
i79	1927	1800	7.5
i80	1894	1800	6.1
i81	6653	1800	11.2
i82	1237	1800	9.6
i83	1027	1800	5.9
i84	1245	1800	11.7
i85	1887	1800	15.8
i86	2146	1800	14.2
i87	712	1800	19.7
i88	1220	1800	12.6
i89	184	1800	31.7
i90	896	1800	15.1
i91	112	1800	34.4
i92	75	1800	39.4
i93	541	1800	16.2
i94	78	1800	45.8
i95	68	1800	30.8
i96	65	1800	34.5
i97	748	1800	28.5
i98	1525	1800	22.6
i99	194	1800	53.1
i100	518	1800	42.8
i101	211	1800	39.0
i102	1542	1800	28.0
i103	687	1800	31.0
i104	441	1800	30.1
i105	481	1800	40.7
i106	140	1800	31.9

Tabela 4.10: Branch-and-cut com até 3 TUs.

tabela 4.11 estão os resultados de suas resoluções através de um B&B, sem nenhuma adição de cortes ou qualquer outro auxílio por parte do usuário. Tais dados serão utilizados como base para comparar a melhora obtida através do C&B implementado.

Nos experimentos com o algoritmo B&C concluímos que a inserção de poucas restrições em um nó da enumeração, antes de reotimizá-lo, teve um desempenho melhor. Contudo, como neste caso a execução do algoritmo de planos de corte fica limitada ao nó raíz, decidimos que tal cuidado não seria necessário. Assim, os testes realizados com o C&B não terão nenhuma restrição quanto ao número de inserções de cortes violados.

O primeiro experimento realizado foi utilizando-se uma única TU. Na tabela 4.12 são mostrados os resultados alcançados, sendo que as colunas informam, respectivamente: o nome da instância, o tamanho da maior TU encontrada, o número de cortes adicionados no nó raiz, o valor absoluto da diferença entre o resultado da relaxação linear do problema preprocessado com e sem os cortes, o tempo gasto na procura e inserção dos cortes e o tempo total gasto para se encontrar a solução ótima (soma do tempo de preprocesso com o tempo de procura e inserção dos cortes mais o tempo de resolução pelo *xpress*).

Pelos dados da tabela 4.12 é possível notar que, apesar de termos conseguido algumas melhoras, não é possível afirmar que o C&B teve um melhor desempenho em relação ao B&B. Entretanto, ele pareceu mais promissor comparado ao B&C. Decidimos então investir um maior esforço neste método.

Por este motivo, exploramos outros critérios para a procura dos cortes. Tais critérios estão descritos abaixo juntamente com indicação da tabela nas quais os resultados dos respectivos experimentos são reportados.

1. UTU: utilizando uma única MRPR – tabela 4.12.

2. MTU 1/3: utilizando múltiplas TUs tal que seus tamanhos não sejam menores que $\frac{1}{3}$ da maior MRPR – tabela 4.13.

3. MTU 1/2 (max 3): utilizando no máximo 3 TUs tal que seus tamanhos não sejam menores que $\frac{1}{2}$ da maior MRPR – tabela 4.14.

4. MTU 80% (max 3): utilizando no máximo 3 TUs tal que seus tamanhos não sejam menores que 80% da maior MRPR – tabela 4.15.

A escolha dos critérios descritos acima foi motivada pela possibilidade de se chegar a uma boa variedade na quantidade de TUs geradas a partir da matriz de restrições originais e, consequentemente, usadas na separação das QTUs. Note-se que, com UTU essa quantidade é sempre um, enquanto que com a MTU 1/3 ela, em geral, é bem alta. Já para a MTU 1/2 (max 3) a quantidade é quase sempre três e com a MTU 80% (max 3) ela é um valor intermediário entre aqueles da UTU e da MTU 1/2 (max 3). Pelos dados

nome	Número de			densidade (%)		Relax.	limitantes		tempo
	lin.	col.	elem.	matriz	grafo	linear	primal	dual	total
i73	300	529	1582	1.00	2.97	105729.46	103045	103055.30	1039
i74	300	459	1651	1.20	4.20	96204.08	92029	92038.11	1529
i75	290	459	1909	1.43	5.73	88089.87	83163	83171.28	1631
i76	250	399	1668	1.67	6.65	76115.52	71527	71534.12	1752
i77	250	410	1479	1.44	5.00	79661.83	76078	76085.60	1207
i78	210	410	1473	1.71	6.06	70610.12	66983	66989.66	976
i79	200	379	1577	2.08	8.45	64531.85	59772	59777.97	1356
i80	200	359	1495	2.08	8.45	64016.07	59159	59164.67	1052
i81	330	360	2236	1.88	10.87	79544.65	71272	71278.94	1110
i82	300	390	2019	1.73	8.46	81240.37	73713	73719.81	1032
i83	300	400	2069	1.72	8.46	81600.1	74369	74376.43	1040
i84	315	399	2069	1.65	8.15	83692.41	77647	77654.76	1074
i85	330	400	3379	2.56	19.21	73505.67	63543	63548.89	1214
i86	240	379	2592	2.85	17.79	62598.99	54199	54204.39	1265
i87	260	379	2591	2.63	16.34	64736.65	57542	57547.32	1363
i88	280	339	2314	2.44	15.17	69187.67	59557	59562.95	1438
i89	280	340	3293	3.46	28.02	58676.87	47726	47730.75	1130
i90	280	359	2431	2.42	14.90	70559.99	60998	61004.03	1322
i91	280	380	3683	3.46	28.27	60349.1	48289	48293.82	1586
i92	280	400	3830	3.42	27.67	62819.94	50757	50762.01	1148
i93	300	379	2594	2.28	14.42	73032.14	63833	63839.27	1472
i94	300	379	3651	3.21	26.26	64805.54	53416	53421.17	1757
i95	300	399	3822	3.19	26.00	66249.45	55198	55202.78	1320
i96	300	399	4774	3.99	36.10	60563.95	48068	48072.78	1409
i97	320	360	3445	2.99	24.56	67279.89	55146	55151.35	1526
i98	320	380	3634	2.99	24.60	68262.15	55927	55932.56	1464
i99	320	379	4578	3.77	35.10	62059.98	48543	48547.75	971
i100	320	399	4792	3.75	34.87	62418.32	48543	48547.81	1745
i101	340	380	3625	2.81	23.28	71293.89	58631	58636.85	1701
i102	340	379	4552	3.53	32.87	68219.29	55464	55469.43	1024
i103	340	399	4766	3.51	32.78	68628.67	56088	56089	935
i104	360	399	3812	2.65	22.08	75940.6	63545	63551.29	1471
i105	360	399	4797	3.34	31.98	70085.81	55275	55279.84	1598
i106	360	400	5808	4.03	41.49	66873.66	52046	52050.85	1266

Tabela 4.11: Dados das instâncias i73 a i106 resolvidos pelo XPRESS versão 2008A em sua configuração padrão.

instância	tamanho TU	número de cortes	dif. relax. linear	tempo	
				corte	total
i73	133	5	64.05	0	683
i74	116	5	7.25	0	1060
i75	90	16	182.85	0	1800
i76	80	12	62.92	0	911
i77	93	15	145.04	0	1800
i78	74	15	61.95	0	741
i79	57	28	283.17	0	899
i80	61	27	123.83	0	1108
i81	65	32	361.88	0	1188
i82	76	36	373.6	0	1608
i83	75	35	300.29	0	960
i84	78	32	249.7	0	322
i85	41	62	504.28	1	1800
i86	36	40	283.09	0	1800
i87	43	47	177.12	1	1021
i88	47	39	352.78	0	903
i89	28	119	640.63	2	601
i90	48	55	265.63	0	1800
i91	27	140	981.52	4	1800
i92	25	131	718.61	4	1800
i93	50	61	234.69	0	1800
i94	30	151	963.95	3	938
i95	30	141	754.59	5	1269
i96	19	267	985.86	12	1627
i97	36	152	799.68	4	1800
i98	36	139	561.7	3	1800
i99	22	192	839.4	8	1212
i100	22	206	823.39	14	1800
i101	38	141	873.06	3	1800
i102	24	178	964.69	6	1080
i103	25	159	776.71	7	1282
i104	39	132	773.5	3	1800
i105	28	180	1156.18	7	1800
i106	19	325	1327.84	18	1295

Tabela 4.12: C&B (UTU).

nome	número de cortes	dif. relax. linear	tempo		
			TU	cortes	total
i73	7	64.05	0	0	587
i74	11	8.41	0	0	1012
i75	48	199.73	0	0	1182
i76	28	67.61	0	0	1800
i77	38	162.08	0	0	1800
i78	39	61.95	0	0	909
i79	89	330.06	0	1	1444
i80	92	155.78	0	0	1005
i81	135	396.9	0	1	967
i82	119	373.62	0	1	876
i83	108	305.7	0	1	1800
i84	131	480.73	0	1	812
i85	428	517.49	0	8	1695
i86	222	327.14	0	4	1800
i87	247	205.11	0	3	1615
i88	252	403.3	0	3	1398
i89	888	604.17	0	21	1800
i90	226	427.73	0	4	1800
i91	970	770.55	1	37	1800
i92	945	716.63	1	35	1800
i93	265	261.99	0	3	1800
i94	1330	993.82	1	35	1800
i95	1203	839.82	1	43	1800
i96	2866	1029.86	3	116	1800
i97	923	830.28	1	24	1800
i98	838	686.14	1	34	1800
i99	2173	848.19	2	95	1800
i100	1915	827.38	3	89	1800
i101	943	864.38	1	23	1800
i102	2170	1026.86	3	98	1800
i103	1445	800.64	3	64	1800
i104	880	816.26	1	26	1800
i105	1792	1192.54	3	81	1800
i106	4546	1385.13	6	554	1800

Tabela 4.13: C&B (MTU 1/3).

nome	Tamanho			número	dif. relax.	tempo		
	TU 1	TU 2	TU 3	de cortes	linear	corte	TU	total
i73	133	69	–	7	64.052	0	0	589
i74	116	73	–	9	8.41	0	0	791
i75	90	71	48	42	199.73	0	0	1720
i76	80	57	42	27	67.61	0	0	909
i77	93	65	–	30	162.08	0	0	1800
i78	74	52	38	39	61.95	0	0	915
i79	57	48	34	74	330.06	1	0	1404
i80	61	47	35	80	155.78	0	0	1650
i81	65	60	50	95	396.9	1	0	1601
i82	76	61	48	84	373.62	1	0	1098
i83	75	60	48	79	305.7	1	0	1274
i84	78	65	53	98	480.73	1	0	578
i85	41	37	35	157	510.2	3	0	1800
i86	36	33	27	98	327.14	2	0	1710
i87	43	36	29	142	205.11	2	0	1371
i88	47	42	36	137	403.3	1	0	1497
i89	28	26	23	317	622.02	7	0	1800
i90	48	42	35	139	427.73	2	0	1236
i91	27	24	22	355	769.38	11	0	1800
i92	25	24	21	299	723.54	9	0	1800
i93	50	42	35	145	261.99	1	0	1646
i94	30	27	26	429	958.39	10	0	1800
i95	30	26	23	373	771.03	10	0	1800
i96	19	17	16	620	1016.28	26	1	1800
i97	36	32	29	317	796.12	6	0	1800
i98	36	31	27	370	681.32	10	0	1492
i99	22	21	19	569	858.5	28	1	1463
i100	22	21	20	614	853.63	26	1	1800
i101	38	35	30	359	870.83	7	0	1800
i102	24	22	20	491	992.06	12	1	1168
i103	25	24	19	471	837.66	22	1	1053
i104	39	36	33	305	810.5	7	0	1800
i105	28	23	22	466	1199.81	25	1	1800
i106	19	18	16	891	1343.27	51	2	1800

Tabela 4.14: C&B (MTU 1/2 (max 3)).

nome	Tamanho			número	dif. relax.	tempo		
	TU 1	TU 2	TU 3	de cortes	linear	cortes	TU	total
i73	133	–	–	5	64.05	0	0	671
i74	116	–	–	6	8.41	0	0	860
i75	90	–	–	19	192.25	0	0	1605
i76	80	–	–	15	67.61	0	0	1433
i77	93	–	–	16	137.09	0	0	1249
i78	74	–	–	14	41.43	0	0	1439
i79	57	48	–	52	330.06	0	0	909
i80	61	–	–	34	153.87	0	0	943
i81	65	60	–	68	396.9	0	0	1800
i82	76	61	–	62	373.62	0	0	1800
i83	75	60	–	59	305.7	1	0	1800
i84	78	65	–	74	480.73	0	0	563
i85	41	37	35	157	510.2	3	0	1800
i86	36	33	–	71	327.14	1	0	1800
i87	43	36	–	108	205.11	1	0	1233
i88	47	42	–	97	403.3	1	0	1161
i89	28	26	23	317	622.02	7	0	1800
i90	48	42	–	106	427.73	1	0	1212
i91	27	24	22	355	769.38	10	0	1800
i92	25	24	21	299	723.54	9	0	1800
i93	50	42	–	106	261.99	1	0	1562
i94	30	27	26	429	958.39	9	0	1800
i95	30	26	–	247	761.03	7	0	1613
i96	19	17	16	620	1016.28	26	1	1800
i97	36	32	29	317	796.12	6	0	1800
i98	36	31	–	268	644.37	9	0	1800
i99	22	21	19	569	858.5	28	1	1464
i100	22	21	20	614	853.63	26	0	1800
i101	38	35	–	255	870.83	5	0	1800
i102	24	22	20	491	992.06	12	1	1170
i103	25	24	–	320	773.43	15	1	1185
i104	39	36	33	305	810.5	7	0	1800
i105	28	23	–	332	1199.81	18	1	1800
i106	19	18	16	891	1343.27	52	2	1800

Tabela 4.15: C&B (MTU 80% (max 3).

das tabelas 4.13, 4.14 e 4.15. podemos verificar que a quantidade de submatrizes MRPR usadas está intimamente relacionada com o número de cortes encontrados e, em escala um pouco menor, com o tempo gasto procurando-os. Entretanto, não parece haver uma relação direta com o tempo de convergência do algoritmo.

Além disto, algumas características muito interessantes e que não previámos, foram observadas. Uma delas é que a melhora obtida na relaxação linear após a inserção de cortes, assim como o número de cortes inseridos, não estão diretamente relacionados com o tempo de resolução das instâncias. Ou seja, formulações teoricamente mais fortes não necessariamente aceleram a resolução do problema, e matrizes maiores (com um número maior de restrições) não necessariamente atrasam esse processo.

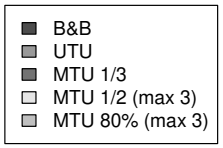
Para podermos analisar melhor os comportamentos descritos, foram construídos os gráficos da figura 4.4, que mostram o tempo gasto por cada abordagem para cada instância. Por esses gráficos percebemos que o desempenho da nossa implementação do algoritmo piora com o aumento da matriz de restrições. Entretanto, fica mais claro que ocorreram diferentes comportamentos para cada tática, e que não é possível afirmar que uma é melhor que a outra. Mesmo assim, é importante salientar que, em quase 50% dos testes, pelo menos uma das implementações do C&B se saiu melhor que o B&B. Isso talvez seja um indicativo de que o uso de QTUs possa vir a ser aprimorado além do que foi possível conseguir nesse projeto tornando-o, quem sabe, uma forma atrativa de resolução do SPack.

Um último ponto que julgamos conveniente estudar foi a sensibilidade dos nossos algoritmos à escolha da submatriz TU que serve de base para a formação das matrizes QTU usadas na busca por desigualdades violadas. Isso é feito na próxima subseção.

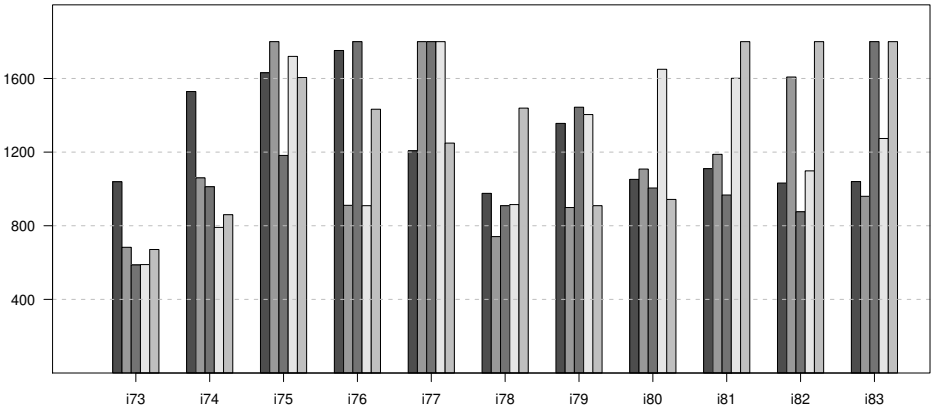
4.5 Sensibilidade do método à escolha da matriz TU

Como dito anteriormente, nesta seção estamos interessados em avaliar o desempenho do nosso algoritmo com relação às TUs empregadas como base nas rotinas de separação das desigualdades cliques. Todas as tabelas apresentadas aqui possuem basicamente as mesmas colunas. São elas: o nome da instância, o tamanho da TU (tam TU), o número de cortes inseridos (no. cortes), a diferença entre os valores ótimos da relaxação linear do problema original e da relaxação do problema com os cortes adicionados (dif. cortes), o gap de dualidade (gap (%)) e o tempo total gasto para chegar neste gap (tempo total). Entretanto, na tabela 4.16, o nome das colunas foram abreviadas, como por exemplo, ct. é a abreviação de cortes e tmp de tempo.

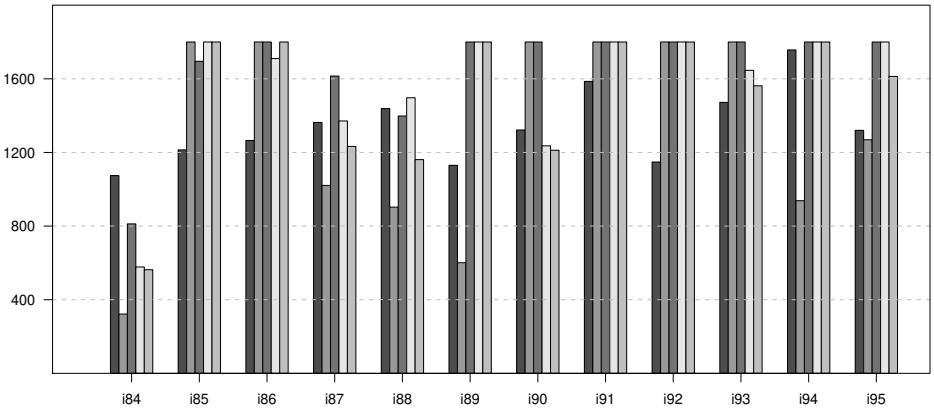
Nesta mesma tabela vemos o comportamento do algoritmo C&B quando resolvemos o mesmo problema considerando TUs diferentes, porém de mesmo tamanho. Através dela podemos perceber que, apesar do número de cortes encontrados terem sido em geral bem



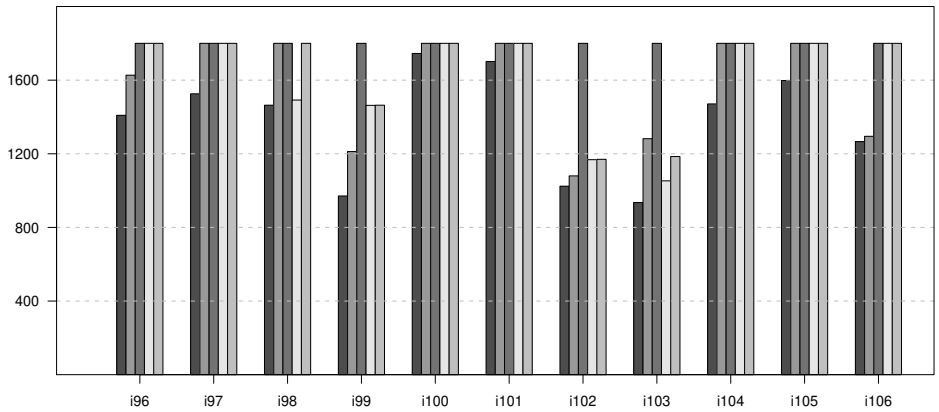
(a) Legenda dos gráficos abaixo



(b) Tempo gasto para as instâncias de i73 a i83.



(c) Tempo gasto para as instâncias de i84 a i94.



(d) Tempo gasto para as instâncias de i95 a i106.

Figura 4.4: Tempo gasto pelo B&B e B&C.

próximos, a escolha da TU influencia muito na convergência do algoritmo, como é o caso, por exemplo, da instancia i76. Entretanto, os dados coletados não possibilitaram concluir quais propriedades são desejáveis na TU para que o C&B encontre a solução mais rápido.

Um outro aspecto interessante a ser analisado é o comportamento do algoritmo quando variamos o tamanho da submatriz TU utilizada. Como já vimos do experimento anterior que TUs do mesmo tamanho já podem levar o algoritmo a se comportar de modo bastante distinto, preocupamo-nos nesse experimento em garantir, tanto quanto possível, que as submatrizes TUs de diferentes tamanhos tinham muitas características em comum. O modo encontrado por nós para assegurar tal propriedade foi o seguinte. A partir da matriz de restrições original do problema, aplicamos a heurística GGMZ para encontrar uma grande submatriz TU, idealmente aquela de tamanho máximo, e denotada aqui por T_0. Em seguida, uma segunda submatriz TU, a qual chamaremos de T_1, é gerada tomando-se $\frac{2}{3}$ das linhas de T_0. Finalmente, uma terceira submatriz TU T_2 é gerada tomando-se metade das linhas de T_1 (note que T_2 tem $\frac{1}{3}$ das linhas de T_0).

Na tabela 4.17 podemos analisar o comportamento do C&B para uma mesma instância mas com TUs de tamanhos diferentes geradas conforme a descrição do parágrafo anterior. Concluímos que o comportamento do algoritmo é muito sensível ao tamanho da matriz TU encontrada, porém, ao contrário do que imaginávamos, uma matriz TU maior não necessariamente nos leva a uma convergência mais rápida do algoritmo C&B. Isso pode ser notado observando-se os dados relativos à instância i78 cujo menor tempo foi conseguido com a menor TU, ou à instância i97 cujo menor tempo de resolução foi conseguido através da matriz de tamanho intermediário. Novamente não conseguimos estabelecer nenhum padrão que justifique essa diferença de comportamento. Ou seja, não fomos capazes de estabelecer nenhuma relação entre características da instância ou mesma da submatriz TU que indicassem que o algoritmo C&B iria ter um desempenho melhor ou pior.

Por último, para ilustrar a dificuldade que temos para obter um tipo de relação como aquela mencionada no parágrafo anterior, acrescentamos ao texto as tabelas 4.18 e 4.19. Elas possibilitam analisar se pequenas mudanças de tamanho da submatriz TU podem gerar grandes mudanças no comportamento do algoritmo B&C. Para isso selecionamos as TUs de maneira semelhante à descrita anteriormente, porém removendo-se apenas algumas linhas (2, 4 e 8 respectivamente). Vê-se que, de fato, o algoritmo C&B é extremamente sensível a pequenas mudanças no tamanho da TU (considere, por exemplo, a instância i84).

Os experimentos relatados nessa subseção mostram que o desempenho do algoritmo C&B é profundamente afetado pela escolha da submatriz TU. Entretanto, não chegamos a uma conclusão sobre como isso deve ser feito. Não resta dúvida que esse assunto precisa ser melhor estudado para que se possa chegar a uma implementação eficiente do método que propusemos nessa dissertação.

inst.	tam. TU	T_1				T_2				T_3			
		no. ct.	dif. ct.	gap (%)	tmp. total	no. ct.	dif. ct..	gap (%)	tmp. total	no. ct.	dif. ct.	gap (%)	tmp. total
i73	33	0	0	0.01	887	0	0	0.01	892	0	0	0.01	893
i74	47	2	1	0.01	1215	2	1	0.01	1214	2	0	0.01	1166
i75	48	8	134	0.01	1095	8	134	1.94	1800	10	109	2.39	1800
i76	42	10	58	0.02	1800	6	45	0.01	1434	6	41	0.01	989
i77	40	7	101	0.01	1582	9	132	0.01	1430	8	132	0.37	1800
i78	38	8	41	0.01	1167	12	62	0.43	1800	10	62	0.01	1165
i79	34	16	108	1.8	1800	11	115	0.01	978	19	132	0.01	922
i80	35	17	91	0.01	1123	15	54	0.01	1473	20	91	0.01	700
i81	50	31	446	0.01	1654	23	440	0.01	808	26	473	0.01	1215
i82	48	27	342	0	1109	20	275	0.01	1370	18	265	0.01	1197
i83	48	31	232	0.01	889	22	290	0.01	1158	23	245	0.01	1494
i84	53	23	255	0.01	1026	19	223	0.01	916	18	255	0.01	421
i85	35	53	461	0.01	1462	56	499	0.01	1532	60	461	0.01	1090
i86	27	34	275	0.01	1709	26	268	0.01	1580	29	275	0.01	1258
i87	29	25	157	0.01	688	36	163	0.01	665	32	163	0.01	1104
i88	36	36	353	0.01	783	35	225	0.01	1247	35	333	0.01	1198
i89	23	131	624	0.01	742	115	600	0.01	593	121	605	0.01	839
i90	35	42	281	0.01	1617	38	174	0.01	1695	38	281	0.01	1369
i91	22	115	742	5.59	1800	145	739	2.25	1800	133	748	5.56	1800
i92	21	125	744	4.26	1800	120	692	6.68	1800	109	695	6.28	1800
i93	35	44	226	2.81	1800	46	220	2.21	1800	40	209	1.28	1800
i94	26	169	890	0.01	1362	149	923	0.01	1297	144	882	0.01	1355
i95	23	133	761	0.01	1281	144	792	0.01	1771	147	738	0.01	1491
i96	16	253	1072	0.01	1545	240	968	0.73	1800	212	944	4.8	1800
i97	29	137	770	0.01	1553	138	734	5.34	1800	114	700	0.01	1058
i98	27	93	526	0.01	1575	98	526	0.01	1251	89	471	0.01	1504
i99	19	190	840	2.61	1800	197	856	0.01	1634	222	857	0.01	1507
i100	20	214	823	2.62	1800	231	827	6.37	1800	200	831	3.25	1800
i101	30	118	827	5.19	1800	127	795	6.44	1800	121	796	5.31	1800
i102	20	157	870	0.01	949	163	890	0.01	1278	156	881	0.01	929
i103	19	172	783	1.6	1800	133	676	0.01	1262	143	676	3.33	1800
i104	33	119	783	6.86	1800	123	750	1.35	1800	101	589	2.52	1800
i105	22	171	1090	0.01	1516	183	1093	5.24	1800	160	1068	5.4	1800
i106	16	305	1305	0.01	1308	342	1326	0.01	1496	281	1180	0.01	1370

Tabela 4.16: Tabela com a resolução da mesma instância usando TUs diferentes, porém de mesmo tamanho.

inst.	TU total					$\frac{2}{3}$ da TU					$\frac{1}{3}$ da TU				
	tam TU	no. cortes	dif. cortes	gap (%)	tempo total	tam TU	no. cortes	dif. cortes	gap (%)	tempo total	tam TU	no. cortes	dif. cortes	gap (%)	tempo total
i73	133	5	64.1	0.0	671	88	5	64.1	0.0	671	44	0	0.0	0.0	884
i74	116	6	8.4	0.0	857	77	5	8.4	0.0	1046	38	2	1.2	0.0	1221
i75	90	19	192.3	0.0	1613	60	9	134.2	0.0	1342	30	5	57.5	2.6	1800
i76	80	15	67.6	0.0	1433	53	11	58.7	0.0	994	26	2	2.8	2.6	1800
i77	93	16	137.1	0.0	1255	62	11	125.6	0.0	819	31	5	93.5	0.0	1363
i78	74	14	41.4	0.0	1443	49	10	40.7	0.0	1134	24	4	6.3	0.0	1075
i79	57	25	308.5	0.0	1286	38	18	108.1	0.4	1800	19	7	93.9	0.0	1492
i80	61	34	153.9	0.0	953	40	17	123.1	0.0	1080	20	12	81.4	0.0	867
i81	65	41	353.9	0.0	738	43	28	286.7	0.0	1765	21	8	104.8	0.0	1019
i82	76	32	341.9	0.0	1236	50	27	341.9	0.0	1110	25	18	265.8	1.7	1800
i83	75	40	301.9	0.8	1800	50	29	232.2	0.0	992	25	15	186.7	0.0	1271
i84	78	44	476.2	0.0	431	52	23	238.7	0.0	1025	26	10	190.2	0.0	362
i85	41	59	482.6	0.0	1245	27	38	347.5	3.0	1800	13	22	277.0	0.0	1618
i86	36	40	327.1	1.7	1800	24	30	234.4	2.4	1800	12	13	125.2	0.0	1430
i87	43	54	197.7	0.0	1149	28	29	153.8	0.0	1633	14	13	73.1	0.0	864
i88	47	56	403.3	0.0	830	31	39	358.1	0.0	895	15	13	30.5	0.0	868
i90	48	61	427.7	0.0	1097	32	40	280.6	0.0	1346	16	20	148.8	0.0	1364
i93	50	58	262.0	2.8	1800	33	45	222.4	2.5	1800	16	26	219.3	0.0	1797
i97	36	126	798.1	0.0	1198	24	124	704.5	0.0	843	12	54	459.1	0.0	1096
i98	36	146	645.8	0.0	1415	24	92	519.9	0.0	1474	12	47	400.1	4.9	1800
i101	38	122	841.7	5.4	1800	25	90	744.0	3.6	1800	12	66	619.2	4.1	1800
i104	39	120	801.3	5.2	1800	26	109	621.7	6.8	1800	13	61	450.4	2.6	1800

Tabela 4.17: C&B para TUs de tamanhos diferentes e proporcionais.

inst.	TU total					TU - 2				
	tam TU	no. cortes	dif. cortes	gap (%)	tempo total	tam TU	no. cortes	dif. cortes	gap (%)	tempo total
i73	133	5	64.05	0.01	671	131	5	64.05	0.01	673
i74	116	6	8.41	0.01	857	114	6	8.41	0.01	860
i75	90	19	192.25	0.01	1613	88	19	192.25	0.01	1603
i76	80	15	67.61	0.01	1433	78	15	67.61	0.01	1442
i77	93	16	137.09	0.01	1255	91	15	136.02	0.31	1800
i78	74	14	41.43	0.01	1443	72	14	41.43	0.01	1449
i79	57	25	308.54	0.01	1286	55	28	283.38	1.43	1800
i80	61	34	153.87	0.01	953	59	34	153.87	0.01	939
i81	65	41	353.94	0.01	738	63	40	353.94	0.01	1587
i82	76	32	341.93	0.01	1236	74	32	341.93	0.01	1192
i83	75	40	301.92	0.82	1800	73	39	301.92	2.17	1800
i84	78	44	476.22	0.01	431	76	43	474.74	0.01	894
i85	41	59	482.63	0.01	1245	39	53	487.04	0.01	1476
i86	36	40	327.14	1.73	1800	34	37	327.01	0.01	1072
i87	43	54	197.68	0.01	1149	41	57	189.72	0.01	861
i88	47	56	403.3	0.01	830	45	54	402.37	0.01	1275
i89	28	128	572.87	0.01	1256	26	108	585.8	0.01	694
i90	48	61	427.73	0.01	1097	46	56	351.6	0.01	1409
i91	27	153	770.38	3.89	1800	25	132	746.58	3.66	1800
i92	25	122	748.43	4.17	1800	23	128	709.13	3.85	1800
i93	50	58	261.99	2.84	1800	48	57	261.99	0.01	1624
i94	30	175	946.94	0.01	1655	28	172	929.13	0.01	911
i95	30	153	832.58	0.01	1091	28	139	765.95	1.99	1800
i96	19	279	1067.93	3.89	1800	17	279	1012.89	3.95	1800
i97	36	126	798.13	0.01	1198	34	122	678.87	3.37	1800
i98	36	146	645.83	0.01	1415	34	147	644.23	0.01	1289
i99	22	216	844.89	0.01	1383	20	209	848.32	0.01	1397
i100	22	207	833.61	2.71	1800	20	214	804.49	2.6	1800
i101	38	122	841.71	5.37	1800	36	117	848.32	3.76	1800
i102	24	161	969.89	0.01	865	22	151	913.5	0.01	1312
i103	25	182	820.89	0.01	1634	23	163	783.88	0.38	1800
i104	39	120	801.27	5.15	1800	37	112	556.56	6.01	1800
i105	28	190	1149.85	5.69	1800	26	191	1158.33	2.51	1800
i106	19	302	1327.25	1.68	1800	17	279	1234.07	0.01	1459

Tabela 4.18: C&B utilizando TUs de tamanhos ligeiramente diferentes (parte 1).

inst.	TU - 4					TU - 8				
	tam TU	no. cortes	dif. cortes	gap (%)	tempo total	tam TU	no. cortes	dif. cortes	gap (%)	tempo total
i73	129	5	64.05	0.01	678	125	5	64.05	0.01	673
i74	112	6	8.41	0.01	858	108	6	8.41	0.01	856
i75	86	18	192.25	2.43	1800	82	18	192.25	2.42	1800
i76	76	14	67.61	0.01	822	72	14	67.61	0.01	829
i77	89	14	136.02	0.01	893	85	14	136.02	0.01	896
i78	70	14	41.43	0.01	1449	66	11	40.67	0.01	859
i79	53	28	281.97	0.01	1598	49	26	232.22	0.01	814
i80	57	33	153.87	0.01	1530	53	31	151.62	0.01	738
i81	61	39	353.94	0.01	833	57	39	351.72	1.46	1800
i82	72	31	341.93	0.01	1312	68	29	341.93	0.01	1245
i83	71	37	301.92	1.98	1800	67	35	301.92	0.73	1800
i84	74	43	474.74	0.01	891	70	40	474.74	0.01	351
i85	37	56	477.6	3.54	1800	33	48	412.37	0.01	1700
i86	32	41	288.49	0.01	1549	28	34	274.64	2.74	1800
i87	39	38	170.05	0.01	1033	35	42	165.91	0.01	1209
i88	43	47	347.88	0.01	922	39	44	354.27	0.01	814
i89	24	104	565.74	0.01	720	20	117	601.32	0.01	1460
i90	44	53	351.6	0.01	1318	40	48	253.68	0.01	1083
i91	23	137	750.94	6.57	1800	19	135	755.19	2.69	1800
i92	21	125	744.02	4.2	1800	17	107	549.42	0.68	1800
i93	46	58	253.41	2.94	1800	42	58	242.27	0.01	1475
i94	26	169	890.03	0.01	1366	22	113	852.22	0.01	1510
i95	26	142	760.04	0.01	1786	22	122	685.1	0.01	1336
i96	15	232	917.51	5.81	1800	11	204	807.73	4.25	1800
i97	32	138	723.89	0.01	986	28	134	643.84	0.01	1777
i98	32	140	718.78	2.94	1800	28	117	548.65	3.3	1800
i99	18	179	825.79	0.01	1300	14	167	812.31	4.97	1800
i100	18	194	786.38	4.61	1800	14	199	812.14	4.98	1800
i101	34	111	844.27	0.95	1800	30	118	826.66	5.1	1800
i102	20	157	870.49	0.01	994	16	157	830.05	0.01	785
i103	21	180	838.42	2.09	1800	17	114	709.66	0.01	1287
i104	35	131	781.76	4.51	1800	31	110	740.24	3.41	1800
i105	24	189	1138.64	7.12	1800	20	168	1091.28	4.41	1800
i106	15	313	1309.19	0.01	1363	11	301	1281.47	0.01	1071

Tabela 4.19: C&B utilizando TUs de tamanhos ligeiramente diferentes (parte 2).

Capítulo 5

Conclusões e considerações finais

Nesta dissertação introduzimos o conceito de matrizes quase-totalmente unimodulares (QTUs). Essas matrizes possuem uma linha que, se for removida, resulta numa matriz que é totalmente unimodular. Essa definição foi introduzida com o intuito de resolver instâncias genéricas do problema de empacotamento (SPack). A pesquisa efetuada aqui foi inspirada no trabalho de Crowder et al [10] onde o conhecimento sobre o politopo da mochila binária foi aplicado para resolver problemas de programação inteira gerais usando algoritmos de planos de corte. No caso do SPack, fizemos um estudo poliedral para identificar facetas do politopo associado a instâncias do problema quando a matriz de restrições é QTU, chamadas de QTU-SPack. Isso permitiu o desenvolvimento de algoritmos de planos de corte para o SPack nos moldes do que foi feito em [10]. Os principais ingredientes desses algoritmos são: (1) identificação de uma grande submatriz TU T da matriz de restrições A correspondente a instância original do SPack e (2) resolução do SPack usando um algoritmo de planos de corte no qual a separação de desigualdades válidas é feita sobre a instância do QTU-SPack T_i obtida ao se considerar apenas as linhas da matriz A composta por T e uma linha qualquer i em A mas não em T, sendo essa separação feita para todas essas linhas.

Além de introduzirmos o conceito de QTU, realizar o estudo poliedral dos politopos correspondentes e propor os algoritmos de planos de corte como sugeridos acima, também implementamos tais algoritmos e realizamos vários experimentos para avaliar o seu desempenho computacional.

Diferentes implementações foram testadas usando os planos de corte derivados das matrizes QTUs. Dentre eles, destacamos os algoritmos *branch-and-cut* (B&C) e *cut-and-branch* (C&B). Nenhum deles obteve resultados de qualidade superior aqueles alcançados por um resolvedor comercial (XPRESS) ao executar o algoritmo *branch-and-bound* (B&B) na sua configuração padrão. Acredita-se que isso se deve ao elevado custo da separação das desigualdades e também ao fato das relaxações lineares se tornarem mais difíceis de

serem resolvidas a medida que novas restrições são acrescidas a elas. Talvez por isso, o algoritmo C&B mostrou-se vantajoso quando comparado ao B&C já que cortes só são adicionados no primeiro nó da árvore de enumeração.

Em uma tentativa de prover maior diversidade nos cortes que são gerados, verificou-se a possibilidade de usar várias submatrizes TUs para aplicar o método proposto aqui. De fato, isso levou a uma melhora no desempenho do C&B. Inclusive, considerando as três diferentes estratégias propostas nesse trabalho para geração de múltiplas TUs, isso permitiu que em cerca de 50% das instâncias testadas chegássemos a resultados melhores do que aqueles obtidos pelo B&B padrão do XPRESS. Contudo, essas estratégias individualmente continuaram sendo dominadas pelo resolvedor comercial.

Outros experimentos que realizamos mostraram que há uma forte dependência entre o desempenho do C&B e a escolha da submatriz TU usada para compor as matrizes QTUs empregadas na separação, mesmo quando o tamanho dessa submatriz permanece inalterado. Além disso, foi mostrado experimentalmente que submatrizes TUs maiores não necessariamente conduzem a melhores resultados, ainda que haja indicações de que elas não devam ser muito pequenas. Na verdade, observamos que ao removermos apenas algumas poucas linhas da submatriz TU já podemos levar o algoritmo a ter um comportamento bastante diverso.

Também é digno de nota que fizemos experimentos com um preprocessamento das instâncias do SPack utilizando um *lifting* por cliques maximais que se mostrou muito vantajoso, notadamente para instâncias onde a matriz de restrições tem densidade acima de 5%. Embora esse preprocesso não tenha muita influência quando a densidade da matriz de restrições é muito baixa, nesses casos ele leva um tempo praticamente insignificante para ser efetuado e pode ser executado sem prejuízo para o tempo total de resolução do problema. Curiosamente, algumas versões do resolvedor comercial XPRESS parecem não executar esse tipo de preprocessamento o que, pelo menos nas instâncias cuja matrizes são mais densas, acarreta um maior tempo de resolução do SPAck.

Deixamos para um trabalho futuro estudar melhor as propriedades das instâncias do SPack, a fim de entender quais delas provocam maiores alterações no desempenho do algoritmo. Além disso, ainda é preciso entender melhor como a escolha da submatriz TU se reflete na qualidade dos cortes obtidos e, finalmente, seria interessante saber porque algumas TUs menores contidas em outras TUs maiores fazem com que algumas instâncias sejam resolvidas em um tempo menor.

Um outro tópico para um estudo mais aprofundado seria tentar encontrar famílias de matrizes TUs diferentes, isto é, que não fiquem restritas apenas às MRPRs já que, neste caso, vimos que a grande maioria das restrições das envoltórias convexas limitam-se às desigualdades clique. Apesar de serem possíveis de ocorrer, as restrições de buraco ímpar aparecem com pouca freqüência na envoltória convexa das QTUs. Além disso, mostramos

nessa dissertação que uma boa parte das restrições conhecidas na literatura para o SPack não podem ocorrer neste caso particular. Porém, nunca é demais lembrar que, nessas provas, assim como em todas instâncias utilizadas nos nossos experimentos, foram testados apenas QTUs cujas TUs de base eram MRPRs. Ao mudarmos esta propriedade talvez novas restrições válidas fortes, ou até mesmo facetas, possam ser encontradas, nos levando a outros algoritmos de separação.

No geral, apesar de o objetivo principal (encontrar e adicionar cortes automaticamente de forma a aumentar o desempenho do algoritmo padrão do XPRESS) não ter sido plenamente alcançado, as conclusões obtidas foram interessantes e apontam para trabalhos futuros.

Referências Bibliográficas

[1] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58:295 – 324, 1993.

[2] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Elsevier Science Publishing Co., Inc., 1982.

[3] R. Borndörfer. *Aspects of Set Packing, Partitioning, and Covering*. Shaker Verlag, Aachen, 1998. Ph.D. thesis, Technische Universität Berlin.

[4] R. F. A. Cavalcanti. Algoritmos de programação linear inteira para identificação de matrizes de rede em sistemas lineares. Relatório final de Iniciação Científica (FAPESP), 2007. processo: 2006/00793-7.

[5] E. Cheng and W. H. Cunningham. Separation problems for the stable set polytope. In *Proceedings of the 4th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 65–79, London, UK, 1995. Springer-Verlag.

[6] E. Cheng and W. H. Cunningham. Wheel inequalities for stable set polytopes. *Mathematical Programming*, 77(3):389–421, 1997.

[7] E. Cheng and S. de Vries. Antiweb-wheel inequalities and their separation problems over the stable set polytopes. *Mathematical Programming*, 92(1):153–175, 2002.

[8] E. Cheng and S. de Vries. On the facet-inducing antiweb-wheel inequalities for stable set polytopes. *SIAM Journal on Discrete Mathematics*, 15(4):470–487, 2002.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Algoritmos Teoria e Prática*. Campus, 2002.

[10] H. P. Crowder, E. Johnson, and M. W. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.

[11] L. Cánovas, M. Landete, and A. Marín. New facets for the set packing polytope. *Oper. Res. Lett.*, 27(4):153–161, 2000.

[12] J. M. da Cunha Silva. Heurística grasp para identificação de matrizes de rede em sistemas lineares. Relatório final de Iniciação Científica (FAPESP), 2007. processo: 2006/00952-8.

[13] P. T. da Silva Tavares. *Matrizes Totalmente e Quase Totalmente Unimodulares*. 2005. Dissertação de messtrado, Universidade de Coimbra.

[14] C. C. de Souza. *The Graph Equipartition Problem: Optimal Solutions, Extensions and Applications*. PhD thesis, Université Catholique de Louvain, 1993.

[15] T. Feo, M. Resende, and S. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.

[16] C. A. Furushima. Estudo de uma heurística para a extração de submatrizes de rede puras em programação linear inteira. Relatório final de Iniciação Científica (PIBIC–CNPq), 2007.

[17] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[18] M. Grötschel, L. Lovasz, and A. Schrijver. Geometric algorithms and combinatorial optimization. *Springer Verlag*, 1988.

[19] N. Gülpinar, G. Gutin, G. Mitra, and A. Zverovitch. Extracting pure network submatrices in linear programs usign signed graphs. *Discrete Applied Mathematics*, 137:359–372, 2004.

[20] R. Karp. Reducibility among combinatorial problems. In M. Jünger, T. Liebling, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi, and L. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, chapter 8 (Part I), pages 219–241. Springer–Verlag, 2010.

[21] A. T. Murray and R. L. Church. Facets for node packing. *European Journal of Operational Research*, 101(3):598–608, September 1997.

[22] G. L. Nemhauser and L. E. T. Jr. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, 5(1):199–215, 1973.

[23] 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(5):443–457, 1992.

[24] G. L. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, New York, 1988.

[25] M. W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 6(1):48–61, 1974.

[26] M. W. Padberg. (1,k)-configurations and facets for packing problems. *Mathematical Programming*, 18(1):94–99, 1980.

[27] L. E. Trotter. A class of facet producing graphs for vertex packing polyhedra. *Discrete Mathematics*, 12:373–388, 1975.

[28] L. A. Wolsey. Further facet generating procedures for vertex packing polytopes. *Mathematical Programming*, 11(1):158–163, 1976.