Uso de cortes canônicos no método de ramificação local para problemas inteiros 0–1 mistos

Rafael Francisco dos Santos

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Uso de cortes canônicos no método de ramificação local para problemas inteiros 0–1 mistos

Rafael Francisco dos Santos[1]

Dezembro de 2006

Banca Examinadora:

- Dr. Cid Carvalho de Souza (Orientador)
- Dr. Vinícius Amaral Armentano
  Departamento de Engenharia de Sistemas
  Faculdade de Engenharia Elétrica e de Computação – UNICAMP
- Dr. Orlando Lee
  Instituto de Computação – UNICAMP
- Dr. Flávio Keidi Miyazawa (Suplente)
  Instituto de Computação – UNICAMP

[1]**Projeto apoiado pela FAPESP (processo 04/12890-1).**

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Bibliotecária: Maria Júlia Milani Rodrigues – CRB8a / 2116

Santos, Rafael Francisco dos

Sa59u Uso de cortes canônicos no método de ramificação local para problemas inteiros 0-1 mistos / Rafael Francisco dos Santos -- Campinas, [S.P. :s.n.], 2006.

Orientador : Cid Carvalho de Souza

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Ramificação local. 2. Programação inteira. 3. Heurística. I. Souza, Cid Carvalho de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

(mjmr/imecc)

Título em inglês: Use of canonical cuts in the local branching method for mixed 0-1 integer problems

Palavras-chave em inglês (Keywords): 1. Local branching. 2. Integer programming. 3. Heuristic.

Área de concentração: Otimização combinatoria

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Cid Carvalho de Souza (IC-UNICAMP)
Prof. Dr. Vinícius Amaral Armentano (FEEC-UNICAMP)
Prof. Dr. Orlando Lee (IC-UNICAMP)

Data da defesa: 21-12-2006

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 21 de dezembro de 2006, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Vinicius Amaral Armentano
FEEC – UNICAMP.

Prof. Dr. Orlando Lee
IC – UNICAMP.

Prof. Dr. Cid Carvalho de Souza
IC – UNICAMP.

Uso de cortes canônicos no método de ramificação local para problemas inteiros 0–1 mistos

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Rafael Francisco dos Santos e aprovada pela Banca Examinadora.

Campinas, 21 de Dezembro de 2006.

Dr. Cid Carvalho de Souza (Orientador)

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

Resumo

Nesta dissertação propomos um uso mais geral dos Cortes Canônicos (CCs) introduzidos por Balas e Jeroslow ([2]) no método de Ramificação Local (RamLoc) de Fischetti e Lodi ([6]). A ramificação local é uma heurística de propósito geral para Programação Inteira Mista (MIP) que explora vizinhanças definidas através da adição de inequações lineares ao modelo original. Estas inequações determinam subproblemas que são computados mais rapidamente pelos resolvedores de MIP. Uma análise da execução da RamLoc indicou que, em algumas situações, ela acrescenta cortes de ramificação local muito superficiais (i.e., que descartam poucas soluções) e que estes cortes ocorrem com grande freqüência.

Como os cortes de ramificações locais de Fischetti e Lodi são casos especiais dos CCs para programação inteira 0–1, nós propomos a incorporação de CCs mais profundos (i.e., que descartam mais soluções) à RamLoc. Executamos o algoritmo resultante sobre 25 das 29 instâncias testadas em [6] e obtivemos melhores resultados do aqueles alcançado pela RamLoc original e pelo resolvedor comercial de MIP `XPRESS` com seus parâmetros *default*.

Uma outra investigação que empreendemos foi a inclusão dos CCs na heurística para modelos gerais de programação inteira mista RINS ([3]). Esta heurística surgiu durante o desenvolvimento desta dissertação e apresentou um bom desempenho. Realizamos alguns testes com as mesmas instâncias sobre as quais a RamLoc foi executada e obtivemos resultados promissores.

Por fim, além da utilização dos CCs em heurísticas, criamos uma estratégia de ramificação que pode ser embutida nos algoritmos de *branch-and-cut* dos resolvedores de MIP. Denominamos esta estratégia de *dive branching* e a implementamos no resolvedor `XPRESS`. Em experimentos conduzidos com o mesmo conjunto de instâncias anteriores, obtivemos resultados de melhor qualidade do que aqueles produzidos pelo `XPRESS` com seus parâmetros *default*.

Abstract

In this dissertation we propose a broader usage of the Canonical Cuts (CC) introduced by Balas and Jeroslow ([2]) in the Local Branching method (LB) of Fischetti and Lodi ([6]). The LB is a general purpose heuristic for Mixed Integer Programming (MIP) that explores neighborhoods defined by the addition of linear inequalities to the original model. These inequalities determine subproblems that are computed more quickly by MIP solvers. An analysis of the execution of LB indicated that, in some situations, it adds local branching cuts that are too superficial (i.e., chopping off few solutions) and that these cuts happen very often.

Since the local branching cuts of Fischetti and Lodi are special cases of CCs for 0–1 integer programming, we propose to incorporate deeper CCs (i.e, chopping of more solutions) to LB. We executed the resulting algorithm on 25 out of the 29 instances tested in [6] and we obtained better results than those attained by the original LB and by the `XPRESS` commercial MIP solver under default settings.

Another research that we carried out was the inclusion of CC to the RINS heuristics for general mixed integer programs ([3]). This heuristic appeared during the development of this dissertation and showed a good performance. We carried out some tests with the same instances on which LB was tested and the results are promising.

Finally, besides using the CCs in heuristics, we created a branching strategy that can be embedded to the branch-and-cut algorithms of the MIP solvers. We called it *dive branching* and implemented it in the `XPRESS` solver. In experiments with the same set of instances as before, we obtained results of better quality than those produced by `XPRESS` with default settings.

Agradecimentos

Gostaria de agradecer a todas as pessoas que estiveram comigo e contribuíram para a construção desta dissertação:

- a Deus, pela presença constante nos momentos de certeza ou de dúvidas;
- aos meus pais João e Maria pelo incentivo, carinho e amor;
- aos meus irmãos Rodrigo, André e Laura pela amizade, companheirismo e apoio;
- à minha esposa Lílian pelo amor e compreensão;
- às minhas duas outras mães Salete e Ângela e aos meus sogros Salim e Zony;
- ao Prof. Dr. Cid Carvalho de Souza por ter me guiado nesta jornada e por sua amizade;
- aos malditos Celso, Flávio, Fernandão, Carlos, Didjei, Baboo, Chose, Quintão, Anibal, Chenca, Bozo, Paixão, Pagodinho, Mário e Poly simplesmente por serem "MALDITOS" e grandes amigos;
- aos amigos do mestrado e doutorado do IC pelas longas conversas e dicussões no bandejão e no instituto;
- ao pessoal do laboratório *Loco* – Carlos, Para, Gury, Nilton, Daniel, Peterson, André e Toni pelas conversas e ajudas do dia-dia;
- ao pessoal do *trekking*... "Mamão tá no bote!!!!" ... pelas aventuras e correrias;
- aos professores e funcionários do instituto de computação e a Unicamp por este ambiente acadêmico agradável;
- à FAPESP – Fundação de Amparo à Pesquisa do estado de São Paulo pelo auxílio financeiro;
- aos professores da FIC – Faculdades Integradas de Caratinga pelo incentivo e por me introduzirem à pesquisa durante a graduação.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A programação inteira mista (MIP) é uma das mais importantes ferramentas para resolver problemas difíceis de otimização. Um problema de MIP é definido por um conjunto de variáveis (x) e um conjunto de restrições lineares nestas variáveis ($Ax = b$). Uma grande variedade de problemas de otimização pode ser modelado desta forma e, assim, a MIP constitui-se numa poderosa ferramenta para atacar problemas desta natureza. Para resolver uma MIP a maioria dos resolvedores atuais utilizam algoritmos de *branch-and-bound* ou de *branch-and-cut* para a enumeração do espaço de soluções. Nestes algoritmos a exploração da árvore de espaço de estados é guiada pela solução da relaxação contínua do modelo original e cada nó da árvore pode dar origem a até dois outros nós representando subespaços disjuntos do espaço de soluções representado por ele. Esta técnica é muito eficiente quando a relaxação contínua do problema é uma boa aproximação da envoltória convexa das possíveis soluções inteiras ou, alternativamente, quando a relaxação pode ser apertada pela adição de planos de corte.

No entanto, apesar da grande evolução dos resolvedores de MIP, obter soluções exatas dos modelos resultantes de aplicações reais é muito difícil, pois a maioria dos problemas é NP-difícil. Em muitos casos, a simples obtenção de uma boa solução já é um desafio. Estes resolvedores são ferramentas muito sofisticadas que tentam calcular uma solução ótima para o modelo MIP de entrada em um tempo aceitável de computação ou, quando isto não é possível, uma solução heurística com um erro prático aceitável. Eles permitem aos usuários manipularem alguns parâmetros de suas heurísticas internas para que possam modificar a estratégia de busca por soluções sub-ótimas. Infelizmente, muitas vezes esta liberdade é insuficiente, pois o algoritmo continua lento para encontrar soluções de boa qualidade. Nestes casos, geralmente recorre-se a heurísticas específicas para o problema dado como entrada as quais, na maioria das vezes, não podem ser aplicadas a outros problemas mais gerais.

No que se refere à resolução de MIPs gerais, as pesquisas há muitos anos vem con-

centrando o seu foco na obtenção de soluções ótimas. Contudo, na prática, a incerteza sobre dados de entrada muitas vezes não justifica o esforço despendido para se comprovar a otimalidade de uma solução. Além disso, não raro é preciso que se chegue à uma boa solução num tempo muito curto.

Na década de 90 a Programação por Restrições (PR) despontou como uma alternativa interessante para a MIP. A facilidade de expressar um modelo usando MIP é grande mas é até maior na PR. Além disso, em várias aplicações industriais, notadamente em problemas de escalonamento e seqüenciamento, a PR mostrou-se mais ágil que a MIP, atingindo boas soluções em menor tempo.

Por outro lado, a grande desvantagem da PR em relação a MIP é a inexistência de limitantes duais fortes para guiar a busca, o que torna a tarefa de provar a otimalidade muito mais complexa para a PR. Assim, numa tentativa de recuperar o terreno perdido para a PR, alguns grupos ligados à área de MIP passaram a aumentar os esforços na direção de obter soluções viáveis para MIPs gerais a um baixo tempo computacional, diversificando a pesquisa, antes quase que totalmente centrada na prova de otimalidade.

Nos últimos anos, várias heurísticas e meta-heurísticas para MIPs gerais tem sido propostas como [6], [3], [10], [1], [13], [7], [8], [11]. A introdução de algumas destas heurísticas nos resolvedores comerciais de MIP tem melhorado consideravelmente o seu desempenho. Nesta dissertação, iremos trabalhar com duas delas: a Ramificação Local (RamLoc), do inglês *Local Branching* [6], proposta por Fischetti e Lodi em 2003 e a *RINS - Relaxation Induced Neighborhood Search* [3] proposta por Danna, Rothberg e Le Pape em 2004.

Neste trabalho, vamos denotar os MIPs onde existam variáveis binárias como MIPs 0–1. Em particular, estamos interessados em heurísticas para modelos MIPs 0–1 puros, ou seja, aqueles onde todas variáveis inteiras são binárias.

Ramificação Local

A RamLoc é um método heurístico para MIPs 0–1 que tem como objetivo acelerar a busca por soluções sub-ótimas. Para isso, ele utiliza um resolvedor genérico de MIP como uma ferramenta "tática" para explorar eficientemente um sub-espaço de soluções que é definido através de uma ramificação simples realizada por um nível "estratégico". O procedimento adotado no método é inspirado em técnicas bem conhecidas de meta-heurísticas de busca local, onde as vizinhanças são obtidas introduzindo-se desigualdades inválidas no modelo MIP original denominadas de *cortes de ramificações locais*. A combinação da estratégia de ramificação de alto nível para definir a vizinhança com a estratégia tática de baixo nível para explorá-la, resulta em um esquema geral que acelera a busca por soluções sub-ótimas produzindo soluções melhores.

Analisando o esquema de funcionamento do método de ramificação local, pode-se notar que os planos de cortes acrescentados nas ramificações do alto nível são muito superficiais (descartam poucas soluções) e que estes cortes ocorrem com grande freqüência. Assim, é possível que a introdução de cortes mais profundos (que descartem mais soluções) possam melhorar consideravelmente a eficiência do método.

Balas e Jeroslow [2] apresentaram em 1972 uma classe de cortes para MIPs 0–1 puros chamados de *cortes canônicos*. Os cortes de ramificações locais propostos por Fischetti e Lodi são casos especiais dos cortes canônicos. Assim, nesta dissertação, optou-se por sanar a dificuldade descrita no parágrafo anterior através da generalização do método de ramificação local incorporando a ele cortes canônicos mais profundos do que aqueles usados na versão originalmente proposta por Fischetti e Lodi.

RINS

A RINS, *Relaxation Induced Neighborhood Search*, é uma heurística de busca local que explora vizinhanças que são criadas avaliando as informações da relaxação do nó corrente da árvore de *branch-and-cut* junto com a melhor solução inteira viável encontrada até aquele momento. As variáveis que possuem o mesmo valor em ambas as soluções são fixadas, criando uma vizinhança. Uma grande vantagem da RINS é que ela pode ser inserida facilmente nos algoritmos de *branch-and-cut* dos resolvedores de programação inteira mista (MIP). Devido às vizinhanças da RINS serem escritas como versões mais restritas do MIP original, as soluções inteiras encontradas durante a enumeração destas vizinhanças são soluções viáveis para o modelo original. Deste modo, elas podem ser utilizadas na atualização dos limitantes primais para podar nós na árvore de *branch-and-cut* do resolvedor, o que ajuda a acelerar o método.

Estudando a RINS, vimos que a exploração das vizinhanças só possui um único objetivo que é melhorar a solução utilizada para podar a árvore de *branch-and-cut*. Então, se conseguirmos uma desigualdade inválida para o modelo MIP original mas que, quando acrescentada a este último, faz com que ele represente a vizinhança explorada, podemos aplicá-la à formulação com o objetivo de eliminar soluções explicita ou implicitamente visitadas.

Determinar tal inequação é bem difícil. Porém, se restringimos a RINS para que fixe somente as variáveis 0–1, as vizinhanças criadas serão hiperplanos canônicos e a inequação que procuramos será um corte canônico, conforme definido abaixo.

Cortes Canônicos

Após Balas e Jeroslow [2] estudarem algumas propriedades dos hipercubos com n-

dimensões, propuseram uma classe de hiperplanos paralelos chamados hiperplanos canônicos. As inequações associadas a estes hiperplanos, chamadas de cortes canônicos (CCs), podem ser utilizadas como planos de cortes para resolver problemas gerais de programação inteira 0–1.

Consideremos um hipercubo $n-$dimensional W_n

$$W_n = \{x \in \mathbb{R}^n \mid 0 \leq x_j \leq 1 \, , j \in \mathcal{N}\}, \tag{1.1}$$

onde $\mathcal{N} = \{1, \ldots, n\}$. Podemos ver que todo vértice x do W_n é um vetor 0–1 que satisfaz n inequações de (1.1) na igualdade. Seja V o conjunto de vértices de W_n.

Uma *face canônica* F^q do W_n de dimensão q, para $0 \leq q < n$, é um conjunto de pontos de W_n que satisfazem exatamente $n-q$ inequações de (1.1) na igualdade. Todos os pontos em $V \cap F^q$ possuem $n-q$ componentes idênticas (aquelas associadas com $n-q$ equações $x_j = 0$ ou $x_j = 1$ que definem a face F^q)[1].

Para $d \in \{0, 1, \ldots, n-q\}$, o hiperplano canônico de ordem q associado à face F^q do W_n, denotado por $HC(F^q)_d$, é dado pela equação

$$\sum_{i \in N(F^q)^+} x_i \; - \sum_{i \in N(F^q)^-} x_i = |N(F^q)^+| - d, \tag{1.2}$$

onde $N(F^q)^+ = \{j \in \mathcal{N} \mid x_j = 1 \text{ para todo } x \in V \cap F^q\}$ e $N(F^q)^- = \{j \in \mathcal{N} \mid x_j = 0$ para todo $x \in V \cap F^q\}$.

Quando substituímos a igualdade na equação (1.2) por uma inequação na forma '$\leq$', obtemos o *Corte Canônico* associado à face F^q. Note que os vetores binários que satisfazem a este corte canônico são aqueles que diferem em d ou mais componentes dos vetores de $V \cap F^q$, isto considerando-se apenas as $n-q$ componentes fixadas pelas equações que definem F^q. Denotaremos por $HC(F^q)_d^+$ o semi-espaço definido pelos pontos do $\mathbb{R}^n$ que satisfazem esta desigualdade.

Vejamos o exemplo da figura 1.1. Neste caso, é apresentado um hiperplano canônico de ordem $q = 4$ que está associado à uma face F^4 do W_n, onde $n = 9$. Este hiperplano canônico é obtido quando fixamos as primeiras 5 componentes de x, ou seja, quando satisfazemos exatamente $n-4$ inequações de (1.1) na igualdade. Assim, utilizando a equação (1.2), obtemos a equação $x_2 + x_4 - x_1 - x_3 - x_5 = 2 - d$ que representa o hiperplano canônico $HC(F^4)_d$.

Nos próximos capítulos desta dissertação, iremos discutir e apresentar os CCs que serão aplicados na RamLoc e RINS, além de apresentarmos um método que os utilizam como critério de ramificação.

[1]Por simplicidade, ao longo do texto usaremos simplesmente o termo *face* para nos referirmos a uma *face canônica*

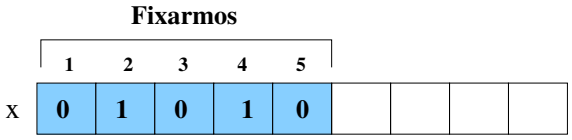

Figura 1.1: Equação do $HC(F^4)_d$ é $x_2 + x_4 - x_1 - x_3 - x_5 = 2 - d$

1.1 Objetivo e metodologia

O objetivo desta dissertação foi investigar a utilização dos CCs em heurísticas de propósito geral para MIPs 0–1 puros. Para isso, realizamos um estudo sobre a RamLoc e RINS, e definimos qual o local e o tipo de CC a ser aplicado. Também, apresentamos um método que utiliza os CCs como uma estratégia de ramificação, o qual chamamos de *Dive Branching*. O desempenho de todos os métodos implementados aqui foi medido através de experimentos práticos utilizando 25 instâncias propostas em [6]. Para compararmos os métodos em um determinado teste, utilizamos duas métricas. A primeira foi realizada discretizando o tempo de exploração das instâncias de uma hora em 36 intervalos iguais. No instante de término de cada intervalo, foi computada a razão entre o melhor valor obtido até aquele momento por cada um dos métodos empregados naquele teste e o melhor valor final encontrado dentre todos eles para cada uma das instâncias. No fim, as razões de todas as instâncias são utilizadas para o cálculo da média geométrica da razão de cada método. Um modo semelhante de avaliação foi usado em [3]. De acordo com esta metodologia, o melhor método é aquele que encontra o maior número de resultados próximos de 1 dentre os 36 intervalos onde calculou-se a média geométrica. A segunda métrica que utilizamos foi a comparação dos resultados método a método. Neste caso, avaliamos a evolução das soluções em intervalos de tempo de 10 minutos. O método que obteve o maior número de melhores soluções é considerado o superior.

Dada a importância da média geométrica nas análises que fazemos ao longo deste trabalho, apresentamos a sua definição a seguir. Assim, temos que a média geométrica de um conjunto de números positivos é o produto de todos os seus elementos elevado ao inverso da sua cardinalidade. Então, para o conjunto com n números positivos $a_1, a_2, ..., a_n$, a média geométrica é dada por $\sqrt[n]{(a_1 \times a_2 \times \ldots \times a_n)}$.

1.2 Organização do texto

Para descrever o trabalho executado e os resultados alcançados, o texto desta dissertação está organizado da seguinte forma. No capítulo 2 apresentamos a Ramificação Local,

os CCs que foram aplicados na RamLoc e os resultados obtidos pelo novo método. No capítulo 3 apresentamos a RINS, discutimos o modo como inserimos os CCs neste contexto e reportamos os experimentos computacionais que fizemos. Em seguida, no capítulo 4, apresentamos a regra que utiliza os CCs para implementar a ramificação em um algoritmo de *branch-and-bound* com intuito de abreviar o tempo de computação necessário para encontrar soluções viáveis de boa qualidade. No capítulo 5, realizamos uma comparação entre todos os métodos apresentados nesta dissertação. Finalmente, no capítulo 6, fazemos algumas considerações sobre o trabalho que foi desenvolvido e apresentamos as nossas conclusões.

Para melhor entendimento deste trabalho, espera-se que o leitor tenha conhecimento prévio sobre programação linear e programação linear inteira que podem ser encontrados em [14, 15, 9].

Capítulo 2

Ramificação Local

Introduzido por Fischetti e Lodi [6], o método de ramificação local (RamLoc) é uma heurística para MIPs 0–1 gerais que explora vizinhanças promissoras com o intuito de encontrar rapidamente soluções viáveis de boa qualidade. No espírito das meta-heurísticas de busca local, ela explora vizinhanças que são determinadas através da adição de inequações inválidas ao modelo original. Estas inequações inválidas, denominadas cortes de ramificação local, criam, através de um nível "estratégico", sub-MIPs que são explorados pelos resolvedores de MIP em um nível "tático". Os autores demonstraram, utilizando o resolvedor comercial `CPLEX` 7.0, que o método de RamLoc é bem eficiente, o que levou a sua inclusão na versão 9.1 daquele pacote. O esquema da RamLoc é discutido detalhadamente a seguir.

2.1 Estrutura da ramificação local

Consideramos um MIP genérico com variáveis 0 e 1 na forma:

$$
\begin{aligned}
(P) \qquad & min \;\; c^T x && && (2.1)\\
& Ax \geq b && && (2.2)\\
& x_j \in 0,1 && \forall j \in \mathcal{B} \neq \emptyset && (2.3)\\
& x_j \geq 0, \; inteiro && \forall j \in \mathcal{G} && (2.4)\\
& x_j \geq 0 && \forall j \in \mathcal{C} && (2.5)
\end{aligned}
$$

onde os índices das variáveis definidos em $\mathcal{N} = \{1, \ldots, n\}$ são divididas em três conjuntos $\mathcal{B}, \mathcal{G}, \mathcal{C}$ que correspondem ao conjunto de variáveis binárias, inteiras e contínuas, respectivamente, sendo que $\mathcal{B} \neq \emptyset$.

Dada uma solução $\overline{x}$ de (P) e um parâmetro inteiro k, definimos que uma vizinhança $N(\overline{x}, k)$ de $\overline{x}$ é o conjunto de possíveis soluções de (P) que satisfazem a restrição adicional

de ramificação local:

$$\Delta(x, \overline{x}) := \sum_{j \in \overline{S}} (1 - x_j) + \sum_{j \in B \setminus \overline{S}} x_j \leq k, \tag{2.6}$$

onde $\overline{S} = \{j \in \mathcal{B} : \overline{x}_j = 1\}$ e é conhecido como suporte binário de $\overline{x}$. A equação (2.6) pode ser mais facilmente compreendida utilizando a distância de Hamming entre x e $\overline{x}$, denotado por $H(x, \overline{x})$. Isto porque, para x e $\overline{x}$ binários, (2.6) é equivalente a:

$$H(x, \overline{x}) = \sum_{j \in \mathcal{B}} \mid x_j - \overline{x}_j \mid \leq k, \tag{2.7}$$

já que $H(x, \overline{x})$ corresponde ao número de variáveis binárias de x que alteraram o seu valor em relação à $\overline{x}$.

Esta restrição é utilizada como um critério de ramificação em um nível "estratégico" dentro do esquema de enumeração para (P), onde as variáveis são fixadas utilizando o esquema de *soft variable fixing*. Então, dada uma solução $\overline{x}$, o espaço de soluções associados com o nó corrente da ramificação pode ser particionado por meio de uma disjunção

$$H(x, \overline{x}) \leq k \;\; \text{(ramo esquerdo)} \;\; \text{ou} \;\; H(x, \overline{x}) \geq k + 1 \;\; \text{(ramo direito)} \tag{2.8}$$

O tamanho da vizinhança, ou seja, o parâmetro k, é escolhido como o maior número de variáveis que não serão fixadas no ramo esquerdo, onde provavelmente o modelo gerado será mais fácil de resolver do que o seu pai. A idéia é que a vizinhança $N(\overline{x}, k)$ que corresponde ao ramo esquerdo, deve ser suficientemente pequena para que possa ser otimizada em um curto tempo de computação. Porém, idealmente, ela deve ser grande o bastante para poder conter soluções melhores que $\overline{x}$. Durante a execução, o valor de k é aumentado ou diminuído automaticamente de forma adaptativa para melhorar a eficiência do método.

A primeira implementação proposta em [6] da ramificação local é ilustrado na figura 2.1, onde os triângulos marcados pela letra "T" correspondem às ramificações das sub-árvores que são exploradas pelos critérios padrão da ferramenta tática (resolvedor MIP usual). Na figura 2.1 assumimos que temos uma solução inicial $\overline{x}^1$ no nó raiz (1). O ramo esquerdo (nó 2) corresponde à otimização da vizinhança $N(\overline{x}^1, k)$, o qual é executado por um esquema de ramificação tático convergindo para uma solução ótima na vizinhança dada por $\overline{x}^2$. Supondo que $\overline{x}^2$ tenha qualidade superior àquela de $\overline{x}^1$, ela torna-se a nova solução cuja vizinhança é candidata a ser explorada. O esquema é então re-aplicado no ramo direito (nó 3), onde a exploração de $N(\overline{x}^2, k) \setminus N(\overline{x}^1, k)$ no nó 4 produz uma nova solução $\overline{x}^3$ ainda melhor do que $\overline{x}^2$. O nó 5 corresponde ao problema inicial (P) acrescido das restrições $H(x, \overline{x}^1) \geq k + 1$ e $H(x, \overline{x}^2) \geq k + 1$. O ramo esquerdo (nó 6) produz um sub-problema que não possui uma solução melhorada. Nesta situação, é adicionada a restrição $H(x, \overline{x}^3) \geq k + 1$ no ramo direito (nó 7), o qual é explorado por uma ramificação

tática. Note que a solução fracionária da programação linear (PL) do nó 1 não está necessariamente eliminada nos nós filhos (2 e 3), como é sempre o caso ao se aplicar uma ramificação padrão. O mesmo vale para os nós 3 e 5.

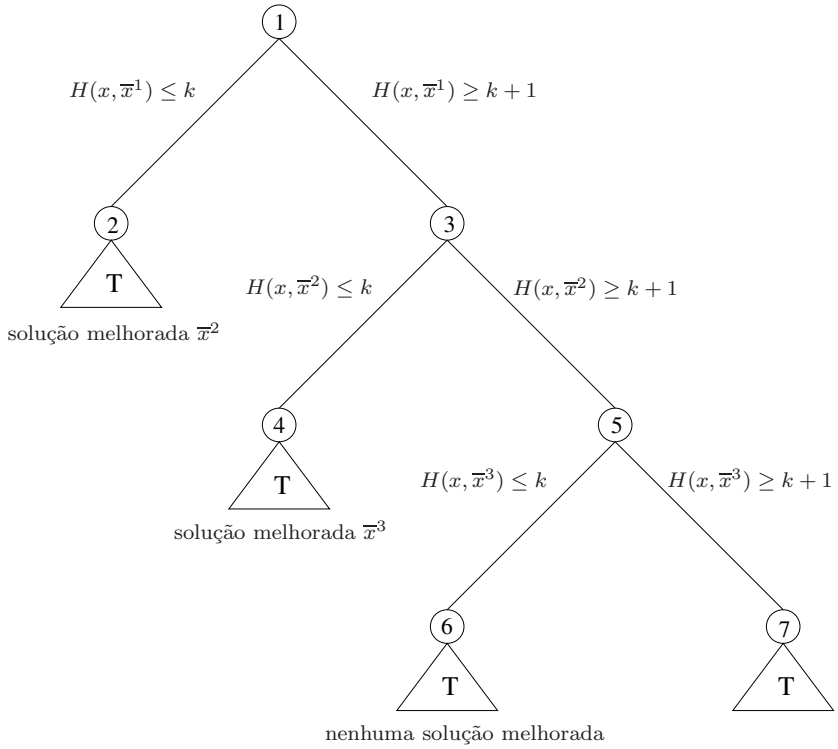

Figura 2.1: Esquema básico da ramificação local.

O que se espera do método de ramificação local é que ele atualize a melhor solução mais cedo e mais vezes do que os algoritmos convencionais implementados nos resolvedores. Isto é feito até que se chegue em um ponto onde não podemos mais aplicar a ramificação local. Neste momento, não temos nenhuma nova solução viável para gerar a próxima vizinhança (nó 6 da figura 2.1) e, então, recorremos à ramificação tática para concluir a enumeração (nó 7 da figura 2.1).

2.1.1 Impondo limite de tempo nos nós de ramificação esquerda

A solução exata do nó do ramo esquerdo pode consumir muito tempo para o valor do parâmetro k dado. Do ponto de vista de uma heurística, é razoável impor um limite de tempo de computação para o ramo esquerdo. No caso do limite ser excedido, duas situações podem ocorrer:

1. Se a solução atual $\overline{x}^i$ foi melhorada, devemos voltar ao nó pai e criar um novo ramo esquerdo associado com a solução atual $\overline{x}^{i+1}$, sem modificar o valor do parâmetro k. Como o resolvedor não conseguiu explorar todo o sub-problema, não podemos adicionar a restrição que o elimina da formulação (P), $H(x, \overline{x}^i) \geq k + 1$, pois não poderíamos garantir a otimalidade do método. Esta situação é demonstrada na figura 2.2, onde o nó 3 atualmente possui 3 filhos. Em um deles, o nó 4, o tempo limite é alcançado e uma solução $\overline{x}^3$ ainda melhor que $\overline{x}^2$ é encontrada mas sem que se tenha provado a otimalidade dentro da vizinhança de $\overline{x}^2$. Os demais filhos do nó 3 são dados pelos ramos regulares esquerdo e direito dados pelos nós $4'$ e 5. Note que, no exemplo, a vizinhança associada com o nó 4 não foi explorada completamente. Portanto, não podemos adicionar a restrição $H(x, \overline{x}^2) \geq k + 1$ aos nós $4'$ e 5, caso contrário não poderíamos garantir que o método retorna uma solução ótima ao terminar. Assim, a RamLoc adiciona em (P) a restrição $H(x, \overline{x}^2) \geq 1$ que elimina somente a solução $\overline{x}^2$.

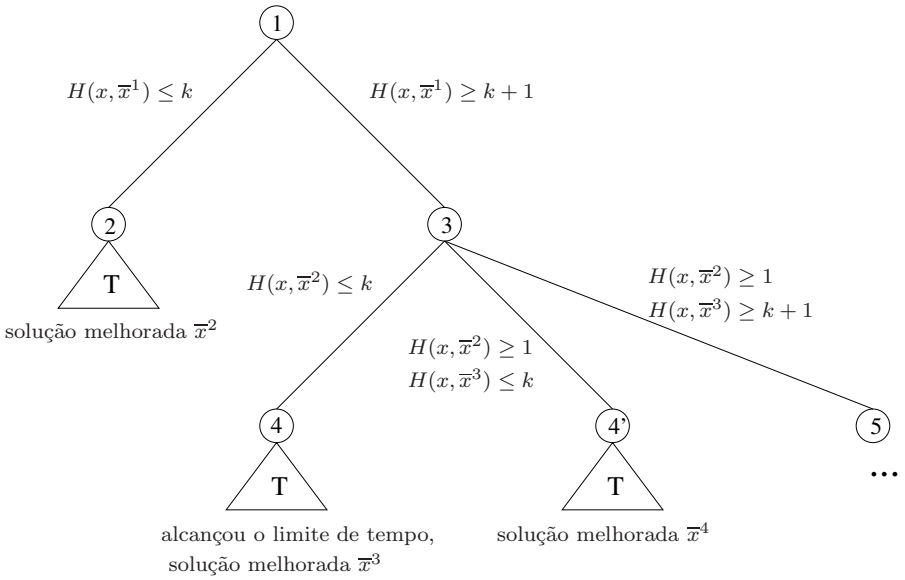

Figura 2.2: Trabalhando com limite de tempo nos nós: caso(1).

2. Se o tempo limite é alcançado sem nenhuma melhoria da solução, o método reduz o tamanho da vizinhança em uma tentativa de intensificar a exploração. Isto é obtido pela redução no ramo esquerdo do termo k para $\lfloor \frac{k}{2} \rfloor$. Esta situação é ilustrada na figura 2.3, onde temos 3 filhos para o nó 3. No nó 4, supõe-se que o tempo limite foi alcançado sem melhorias na solução. Assim, no nó $4'$, reduz-se o tamanho da vizinhança onde, vamos supor, é encontrada uma solução ótima $\overline{x}^3$ na

vizinhança reduzida. No nó 5, procede-se da maneira usual, adicionando-se a restrição $H(x, \overline{x}^2) \geq \lfloor \frac{k}{2} \rfloor + 1$ que exclui a vizinhança reduzida do conjunto de soluções do modelo original.

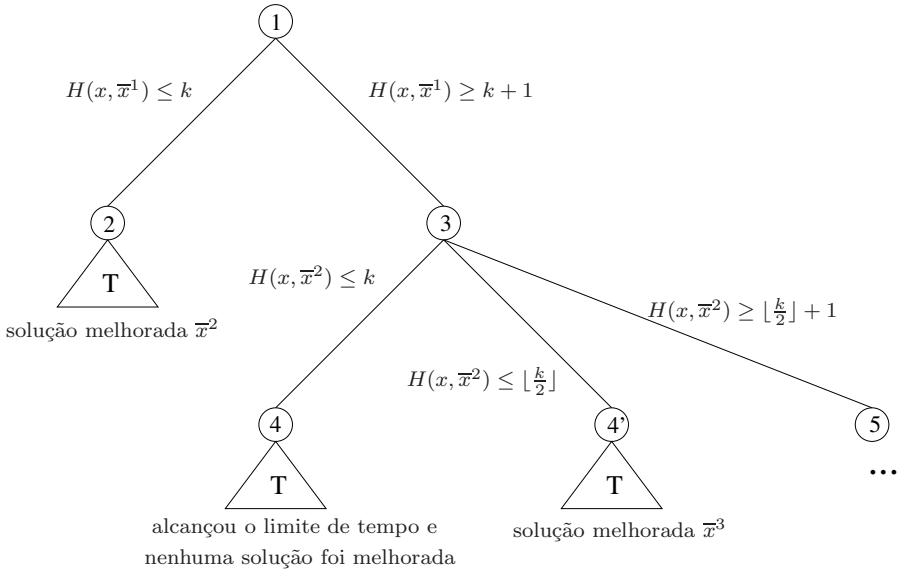

Figura 2.3: Trabalhando com limite de tempo nos nós: caso(2).

2.1.2 A diversificação

Uma alternativa para melhorar o desempenho do método explora o conhecido mecanismo de diversificação, emprestado das meta-heurística de busca local. Neste esquema, a diversificação é aplicada sempre que o nó esquerdo atual não contém nenhuma solução melhor. Este caso ocorre no nó 4 da figura 2.4, onde o esquema padrão seria utilizado para terminar a enumeração, como vimos no nó 7 da figura 2.1. Para manter um controle estratégico na enumeração são utilizados dois mecanismos diferentes de diversificação. Primeiro é aplicada uma diversificação "fraca" que consiste em aumentar a vizinhança corrente adicionando o valor $\lceil \frac{k}{2} \rceil$ ao lado direito do corte de ramificação local que a define. A diversificação então produz um nó no ramo-esquerdo o qual é processado por uma ramificação tática dentro de um determinado limite de tempo (nó $4'$ da figura 2.4). Caso não encontre nenhuma solução melhor na vizinhança estendida dentro do limite de tempo, é aplicada a diversificação "forte" no espírito das meta-heurísticas do tipo *Variable Neighborhood Search* [12] onde se procura uma solução que tipicamente é pior do que a atual mas que, idealmente, não é muito pior do que a solução corrente $\overline{x}$.

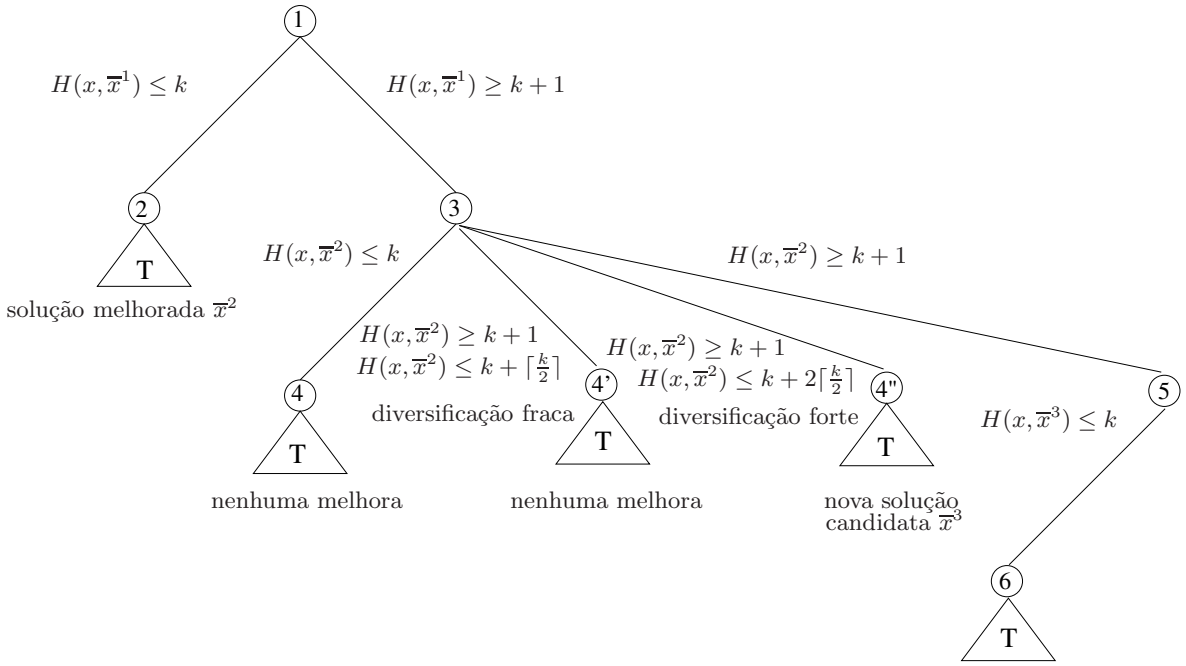

Figura 2.4: Diversificação na RamLoc.

Assim, a ramificação tática para o problema corrente é aplicada alterando a última restrição pela desigualdade $H(x,\overline{x}^2) \leq k + 2\lceil\frac{k}{2}\rceil$, mas sem impor qualquer limite superior para o valor da solução ótima (nó 4" da figura 2.4). A exploração deste nó é abortada tão logo for encontrada uma solução viável. Esta solução, tipicamente pior do que a corrente, é utilizada como a nova solução candidata, e o método é reaplicado em uma tentativa de melhorá-la iterativamente (nó 5 da figura 2.4).

2.1.3 Algoritmo LocBra

Os passos do algoritmo LocBra são descritos pelo pseudocódigo algoritmo 1. A função `LocBra` recebe como entrada o tamanho da vizinhança (k), o tempo limite para cada exploração da ramificação tática (`tempo_limite_nó`), o número máximo de diversificações permitidas (`dv_max`), e o tempo limite total (`tempo_limite_total`). Ao final, ele retorna em (x^*) a melhor solução encontrada ou a solução ótima junto com o status final da otimização (`opt`). Agora, caso não exista uma prova de inviabilidade e nenhuma solução viável tenha sido encontrada, `LocBra` retornará `opt=falso` e x^* =`indefinida`.

O método consiste principalmente na repetição do laço das linhas 4 a 43 até atingir o tempo limite total ou exceder o número de diversificações. Em cada iteração, o problema

MIP correspondente à uma vizinhança é resolvido por um resolvedor tático `MIP_SOLVE` que recebe como entrada 3 parâmetros: o tempo limite local `TL`, o limite superior `UB` usado para interromper a otimização quando o limite inferior torna-se melhor ou igual a `UB`, e a sentinela `primeiro` que, quando marcada para `verdadeiro`, aborta a computação quando a primeira solução viável é encontrada. `MIP_SOLVE` retorna na saída a melhor solução $\tilde{x}$ encontrada na vizinhança corrente juntamente com o estado final da otimização `stat`.

A função `LocBra` utiliza uma sentinela interna denotada por `diversifica` indicando o tipo da próxima diversificação requerida: "fraca" `(diversifica = falso)` ou "forte" `(diversifica = verdadeiro)`. Como regra geral, a forte é executada somente se a fraca tiver sido executada na iteração anterior. Nas iterações que não a requerem, `diversifica` recebe falso.

Quatro diferentes casos podem ocorrer após cada chamada do `MIP_SOLVE`:

1. `solução_ótima_encontrada`: o MIP corrente foi resolvido à otimalidade. Se o MIP resolvido corresponde ao problema original (o nó raiz), a função realiza um retorno imediato (linha 11 do algoritmo 1). Agora, caso o problema resolvido seja um sub-MIP do problema original (nó 2 da figura 2.1), o algoritmo aplica a restrição que o elimina da formulação. Este efeito é obtido invertendo a última restrição de ramificação local $H(x, \overline{x}) \leq rhs$ por $H(x, \overline{x}) \geq rhs+1$, onde a solução corrente $\overline{x}$ que origina o valor `UB` (e possivelmente o valor de x^* do valor `melhorUB`) é atualizado.

2. `provado_inviabilidade`: o MIP corrente não tem solução com custo estritamente melhor do que a entrada do limitante superior `UB`. Como no caso anterior, se o MIP resolvido corresponde ao problema original (o nó raiz), a função realiza um retorno imediato (linha 16 do algoritmo 1). Mas, se o problema resolvido for um sub-MIP do problema original, a restrição que o elimina da formulação é aplicada. Da mesma forma que no caso acima, o sub-MIP é eliminado invertendo a última restrição de ramificação local e é aplicado a diversificação fraca ou forte, como na figura 2.4, dependendo do valor da sentinela `diversifica`.

3. `solução_possível_encontrada`: uma solução com custo estritamente melhor que o limitante superior `UB` foi encontrado, mas o resolvedor não é capaz de provar a otimalidade devido ao limite de tempo imposto ou por ser requerido abortar na primeira solução viável. Neste caso, não podemos eliminar as soluções do sub-MIP explorado. A fim de cortar a solução corrente $\overline{x}$, o método substitui a última restrição de ramificação local $H(x, \overline{x}) \leq rhs$ por uma restrição $H(x, \overline{x}) \geq 1$. Para que possamos aplicar está restrição, devemos garantir que a solução encontrada é a melhor para o conjunto de valores assumidos pelas variáveis binárias, ou seja, quando fixamos $x_j = \overline{x}_j$, $\forall j \in \mathcal{B}$. Tal garantia sempre existe se todas as variáveis do problema são binárias ($\mathcal{G} = \mathcal{C} = \emptyset$). Em um caso geral, a exatidão do método

```
1. function LocBra(k, tempo_limite_total, tempo_limite_nó, dv_max, x*);
2.    rhs:=melhorUB :=UB :=TL :=+∞;  x*:= indefinido;
3.    opt:= primeiro := verdadeiro;  dv:= 0;  diversifica:= falso;
4.    repeat
5.       if(rhs < ∞) then adicione a restrição de ramificação local H(x, x̄) ≤ rhs endif;
6.       TL:= min { TL, tempo_limite_total − tempo_gasto };
7.       stat:= MIP_SOLVE(TL, UB, primeiro, x̃);
8.       TL:= tempo_limite_nó;
9.       if(stat = "solução_ótima_encontrada") then
10.         if(c^T x̃ < melhorUB) then melhorUB := c^T x̃;  x*:= x̃; endif;
11.         if(rhs ≥ +∞ ) return(opt);
12.         inverta a ultima restrição de ramificação local para H(x, x̄) ≥ rhs + 1;
13.         diversifica:= primeiro:= falso;  x̄:= x̃  UB:= c^T x̃;  rhs:= k
14.      endif;
15.      if(stat = "provado_inviabilidade") then
16.         if(rhs ≥ +∞ ) return(opt);
17.         inverta a ultima restrição de ramificação local para H(x, x̄) ≥ rhs + 1;
18.         if(diversifica) then UB := TL:= +∞;  dv:= dv+1;  primeiro:= verdadeiro endif;
19.         rhs:= rhs +⌈k/2⌉;  diversifica:= verdadeiro
20.      endif;
21.      if(stat = "solução_possível_encontrada") then
22.         if(rhs < ∞) then
23.            if(primeiro) then
24.               remova a ultima restrição de ramificação local H(x, x̄) ≤ rhs
25.            else
26.               substitua a ultima restrição de ramificação local H(x, x̄) ≤ rhs por H(x, x̄) ≥ 1;
27.            endif
28.         endif;
29.         REFINE(x̃);
30.         if(c^T x̃ < melhorUB) then melhorUB := c^T x̃;  x*:= x̃; endif;
31.         primeiro:= diversifica:= falso;  x̄ := x̃;  UB:= c^T x̃;  rhs:= k
32.      endif;
33.      if(stat = "nenhuma_solução_possível_encontrada") then
34.         if(diversifica) then
35.            substitua a ultima restrição de ramificação local H(x, x̄) ≤ rhs por H(x, x̄) ≥ 1;
36.            UB := TL:= +∞;  dv:= dv+1;  rhs:= rhs +⌈k/2⌉;  primeiro:= verdadeiro
37.         else
38.            remova a ultima restrição de ramificação local H(x, x̄) ≤ rhs;
39.            rhs:= rhs −⌈k/2⌉
40.         endif;
41.         diversifica:= verdadeiro
42.      endif;
43.   until(tempo_gasto > tempo_limite_total) ou (dv > dv_max);
44.   TL:= tempo_limite_total − tempo_gasto;  primeiro:= falso;
45.   stat:= MIP_SOLVE(TL, melhorUB, primeiro, x*);
46.   opt:= (stat= "solução_ótima_encontrada") ou (stat= "provado_inviabilidade");
47.   return(opt)
48. end.
```

Algoritmo 1: Função LocBra

requer o uso de um procedimento `REFINE(`$\tilde{x}$`)` que substitui a solução de entrada $\tilde{x}$ pela ótima (computada através do resolvedor MIP) na vizinhança $H(x, \tilde{x}) \leq 0$. Isto produz um certificado de otimalidade quando fixamos todas as variáveis binárias para $\tilde{x}$. Note que `REFINE` é aplicado implicitamente para cada solução $\tilde{x}$ fornecida pelo `MIP_SOLVE` embora só seja chamado no passo 3 (porque isto é claramente inútil no passo 1). O procedimento `REFINE` pode gastar um tempo excessivo no caso onde a fixação de variáveis binárias não conduz a um problema de fácil solução, devido à presença de um grande número de variáveis inteiras gerais. Nesta situação, é permitido desativar a chamada do refinamento, mas a última restrição de ramificação local $H(x, \overline{x}) \leq rhs$ não pode ser substituída pela $H(x, \overline{x}) \geq 1$ já que isto pode afetar a otimalidade do método. Agora, caso a solução tenha sido encontrada durante a aplicação da diversificação forte, a restrição $H(x, \overline{x}) \geq 1$ não é aplicada. Isto pode ser visto nas linhas 23 e 24 do algoritmo 1 onde a sentinela `primeiro` tem valor igual a `verdadeiro`. Nesta situação, a restrição de ramificação local é removida retirando o limite imposto pela diversificação forte. Como o nosso trabalho em relação à RamLoc será aplicado somente em MIPs 0–1 puros, nós desabilitamos o procedimento de `REFINE`.

4. `nenhuma_solução_possível_encontrada`: nenhuma solução viável com custo estritamente menor do que `UB` foi encontrada dentro do tempo limite do nó, mas isto não garante que a solução não exista. Neste caso, a última restrição de ramificação local é removida ou substituída pela restrição $H(x, \overline{x}) \geq 1$ dependendo do tipo de diversificação que será executada. No caso de uma diversificação fraca, diferentemente da diversificação apresentada na seção 2.1.2, uma nova vizinhança de ramificação local reduzida é considerada (a restrição anterior de ramificação local é removida, linhas 38, e a vizinhança é reduzida, linha 39 do algoritmo 1), sendo que em uma diversificação forte a restrição $H(x, \overline{x}) \geq 1$ é introduzida para escapar da solução corrente (linha 35 do algoritmo 1), a vizinhança é aumentada e o limite superior `UB` é inicializado com $+\infty$.

Na saída do laço das linha 4 a 43, o tempo restante da computação (se existir) é usado na tentativa de resolver o MIP corrente tentando provar a otimalidade. Isto corresponde ao processamento do nó 7 na figura 2.1. Conseqüentemente o esquema de ramificação local atua como um método exato quando `tempo_limite_total=`$+\infty$ e `dv_max<`$+\infty$ uma vez que para `dv_max=`$+\infty$ o esquema total pode ser visto como uma busca de heurística local.

2.2 Aplicação dos cortes canônicos mais profundos

Como discutimos na seção 2.1.1, quando a RamLoc encontra uma solução viável em uma vizinhança $N(\overline{x}, k)$ mas não consegue provar a otimalidade, ela adiciona uma restrição $H(x, \overline{x}) \geq 1$ que elimina o ponto $\overline{x}$ que gerou a vizinhança. Este processo acontece muitas vezes, segundo [6], o que pode gerar uma re-exploração de partes de vizinhanças já verificadas explicitamente ou implicitamente pelo resolvedor tático. Isto ocorre porque existe uma interseção entre a vizinhança $N(\overline{x}^1, k)$ e a nova vizinhança $N(\overline{x}^2, k)$ gerada pelo ponto $\overline{x}^2 \in N(\overline{x}^1, k)$. Além deste problema, a adição contínua de restrições da forma $H(x, \overline{x}) \geq 1$ pode gerar pontos fracionários na região viável, como será mostrado a seguir.

2.2.1 Cortes geométricos

Em um estudo recente, Souza, Macambira e Maculan [4] apresentaram uma re-interpretação dos resultados discutidos em [2], denominando alguns cortes canônicos especiais como cortes geométricos. Em seu texto eles mostram a equivalência entre as duas definições apresentando dois casos especiais de cortes chamados d-esférico e d-cilíndrico.

Corte d-esférico

Como vimos no capítulo 1, o corte canônico é uma inequação que elimina um dos subespaços definidos pelo hiperplano canônico $HC(F^q)_d$, onde F^q representa uma face de dimensão q do hipercubo n-dimensional W_n e $d \in \{0, 1, \ldots, n-q\}$. Então, considerando uma face de dimensão 0 de W_n, a face F^0 coincide com o ponto $\overline{x}$ do conjunto V de vértices de W_n. Neste caso, nós obtemos o corte canônico $HC(F^0)_d$ na forma

$$\sum_{j|\overline{x}_j=1} x_j - \sum_{j|\overline{x}_j=0} x_j \leq \sum_{j|\overline{x}_j=1} \overline{x}_j - d. \tag{2.9}$$

Essa equação é chamada de corte d-esférico, onde o parâmetro d pode ser interpretado como a distância de Hamming em relação à face F^0. Comparando este corte com o corte de ramificação local da equação $H(x, \overline{x}) \geq k + 1$, podemos ver que eles são equivalentes para $d = k + 1$. Um exemplo de corte 1-esférico para o ponto $(1, 1, 1)$ em um caso tri-dimensional pode ser visto na figura 2.5. A inequação correspondente é dada por $x_1 + x_2 + x_3 \leq 2$.

Quando acrescentamos simultaneamente vários cortes esféricos em um poliedro inteiro, o poliedro resultante pode apresentar vértices fracionários. Este fato não é desejável quando pretendemos trabalhar com vetores inteiros. Um exemplo onde tal situação ocorre pode ser visto na figura 2.6. Vamos supor que queiramos trabalhar com vértices de W_3, excluídos os pontos $\overline{x}^1 = (1, 1, 1)$ e $\overline{x}^2 = (1, 1, 0)$. Se, para isso, adicionamos os cortes

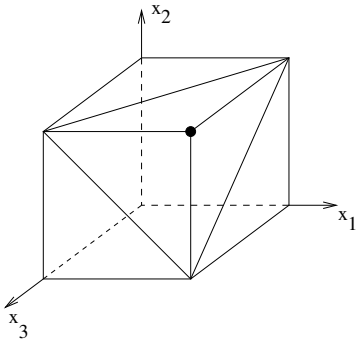

Figura 2.5: Uma inequação 1–esférica

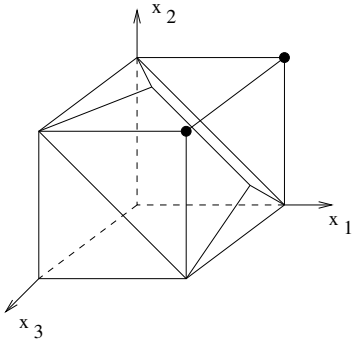

Figura 2.6: Duas inequações 1–esféricas

1-esférico $x_1 + x_2 + x_3 \leq 2$ e $x_1 + x_2 - x_3 \leq 1$ correspondentes aos pontos $\overline{x}^1$ e $\overline{x}^2$, respectivamente, não produziremos um poliedro inteiro.

Corte d-cilíndrico

Se considerarmos $q = 1$ e uma face de dimensão 1 de W_n, a face F^1 coincide com um segmento de reta, de comprimento unitário, que junta dois vértices adjacentes $\overline{x}^1$ e $\overline{x}^2$ de V. Neste caso, $\overline{x}^1$ e $\overline{x}^2$ diferem somente por uma componente e obtemos um corte canônico $HC(F^1)_d$ na forma

$$\sum_{j|\overline{x}^1_j=\overline{x}^2_j=1} x_j - \sum_{j|\overline{x}^1_j=\overline{x}^2_j=0} x_j \leq \sum_{j|\overline{x}^1_j=\overline{x}^2_j=1} \overline{x}^1_j - d. \tag{2.10}$$

Essa equação é chamada de corte d-cilíndrico, onde o parâmetro d pode ser interpretado como a distância de Hamming em relação à face F^1.

Como vimos anteriormente, quando executamos dois cortes 1-esféricos estamos gerando um poliedro fracionário. Mas, se os dois pontos $\overline{x}^1$ e $\overline{x}^2$ forem adjacentes, isto é, quando $H(\overline{x}^1, \overline{x}^2) = 1$, podemos realizar um corte 1-cilíndrico que eliminará a parte fracionária do poliedro inteiro. Este corte pode ser visto na figura 2.7, corrigindo o problema apontado na figura 2.6.

2.2.2 Corte CCF^H

Para minimizar as dificuldades mencionadas no início da seção 2.2, iremos aplicar cortes que eliminam faces de dimensão maior que zero, como é o caso dos cortes da RamLoc padrão. Isso é feito da seguinte maneira. Sejam dois pontos $\overline{x}^1$ e $\overline{x}^2$ de (P) e suponha que $\overline{x}^2$ foi encontrado quando se explorava a vizinhança $N(\overline{x}^1, k)$. Seja H a distância de Hamming entre $\overline{x}^1$ e $\overline{x}^2$. Defina F^H como sendo a face de W_n formado por todos vetores binários de x tais que $x_j = \overline{x}^1_j = \overline{x}^2_j$. O corte canônico que elimina todos vetores binários que estão à distância menor ou igual a d de F^H é dado por

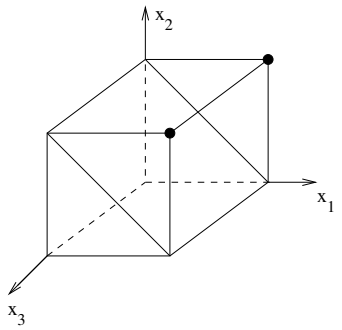

Figura 2.7: *1–cilíndrico* $x_1 + x_2 \leq 1$ para os pontos $\overline{x}^1 = (1, 1, 1)$ e $\overline{x}^2 = (1, 1, 0)$.

$$\sum_{j|\overline{x}^1_j=\overline{x}^2_j=1} x_j - \sum_{j|\overline{x}^1_j=\overline{x}^2_j=0} x_j \leq \sum_{j|\overline{x}^1_j=\overline{x}^2_j=1} \overline{x}^1_j - d, \quad \text{que definiremos como} \quad CCF^H(\overline{x}^1, \overline{x}^2) \geq d \tag{2.11}$$

Para aplicarmos o corte acima a uma formulação e garantirmos que não perdemos a solução ótima, é necessário que saibamos qual a melhor solução dentre aquelas que estão a uma distância menor ou igual a d de F^H. Como o número de pontos neste conjunto cresce exponencialmente com H e d, só aplicaremos cortes com $d = 1$, que eliminam somente os 2^H pontos inteiros pertencentes à face F^H. Assim, definiremos o *tamanho de uma face* como sendo o número de pontos inteiros que a compõem, sendo estes pontos válidos ou inválidos para o conjunto de restrições. A enumeração de maneira implícita ou explicita de todos estes pontos inteiros será chamada de *exploração da face*. Note que, se $H = 1$, a equação (2.11) equivale ao corte d-cilíndrico. Como visto antes, na discussão da figura 2.6, a aplicação deste corte apresenta vantagens em relação ao uso de dois cortes simples, um eliminando só o ponto $\overline{x}^1$ e o outro só o ponto $\overline{x}^2$.

Uma outra possível vantagem ao explorarmos a face definida pelo hiperplano canônico correspondente ao corte $CCF^H(\overline{x}^1, \overline{x}^2) \geq 1$, é que estamos realizando um processo de intensificação na busca. Podemos ver que a exploração da face F^H, definida pelo corte CCF^H, intensifica a busca em uma sub-vizinhança promissora, pois os pontos que pertencem à face possuem características semelhantes às duas melhores soluções encontradas na vizinhança $N(\overline{x}, k)$.

2.3 Implementação

Para avaliarmos a capacidade dos cortes canônicos em melhorar a eficiência da RamLoc em problemas 0–1 puros, criamos uma heurística com 3 variações. As duas primeiras variações exploram a capacidade de redução do espaço de solução através da inclusão dos cortes canônicos CCF^H, e a terceira apenas explora as faces geradas pelos cortes sem adicioná-los à formulação. A idéia da terceira variação é de que ela não interfira diretamente na busca realizada pela RamLoc.

Como discutimos anteriormente, os cortes canônicos que iremos aplicar podem possuir faces com até 2^H soluções inteiras possíveis, o que pode exigir muito tempo para serem computadas. Pensando nesta possibilidade, decidimos criar as duas primeiras variações. A primeira chamada de `DHM` (Distância de Hamming Média), explora no máximo faces de tamanho $2^{k/2}$ e a segunda denominada de `DHG` (Distância de Hamming Grande), explora no máximo faces de tamanho 2^k. Apesar de na maioria das vezes o método de ramificação local encontrar soluções a uma distância de Hamming de no máximo k, é possível que uma combinação de eventos gere um par de pontos a uma distância maior. Este é o caso por exemplo, quando explorarmos toda uma vizinhança do ponto $\overline{x}^i$ e não encontramos nenhuma solução melhor. Assim, o método de RamLoc irá eliminar toda a vizinhança adicionando a restrição $H(x, \overline{x}^i) \geq k + 1$ e estenderá a vizinhança da solução $\overline{x}^i$ em mais $k/2$, ou seja, a próxima solução estará entre $H(x, \overline{x}^i) \geq k + 1$ e $H(x, \overline{x}^i) \leq k + k/2$.

Mesmo limitando o tamanho da face, o tempo gasto para explorá-la completamente pode ser muito longo, prejudicando o desempenho da RamLoc. Logo, acrescentamos um parâmetro `tmax` que determina o tempo máximo para explorar cada face. Deste modo, quando o resolvedor não consegue explorar toda a face dentro do tempo `tmax`, o corte canônico que a elimina não é acrescentado à formulação.

Geralmente, quando adicionamos cortes canônicos originados de $\overline{x}^i$ e $\overline{x}^{i+1}$, podemos retirar o último corte adicionado pela RamLoc $H(x, \overline{x}^i) \geq 1$, pois o corte $CCF^H(\overline{x}^i, \overline{x}^{i+1}) \geq 1$ o domina. Mas, como vimos anteriormente, as variações `DHM` e `DHG` só exploram faces de tamanho máximo $2^{k/2}$ e 2^k, respectivamente. Então, nos casos onde as distâncias de Hamming entre $\overline{x}^i$ e $\overline{x}^{i+1}$ ultrapassam o máximo permitido por cada variação, ou seja, $H(\overline{x}^i, \overline{x}^{i+1}) > k/2$ para `DHM` e $H(\overline{x}^i, \overline{x}^{i+1}) > k$ para `DHG`, uma outra solução na face é gerada (ela pode ser inválida) modificando algumas variáveis de $\overline{x}^i$ para o mesmo valor de $\overline{x}^{i+1}$ até que atinja a distância máxima de Hamming permitida. Para realizar esta fixação, utilizamos a função `GeraNovoPonto` que utiliza um vetor `Hind` de tamanho $H(\overline{x}^i, \overline{x}^{i+1})$ que contém os índices das componentes nos quais os vetores $\overline{x}^i$ e $\overline{x}^{i+1}$ diferem. Utilizamos também, uma variável `PG` que é inicializada com a posição 0 do vetor `Hind` e armazena, no final da execução da função, qual foi a última posição fixada no vetor. A idéia de utilizar a variável global `PG` é para evitar que as variáveis iniciais do vetor `Hind` sejam

sempre as primeiras a serem modificadas, fazendo com que o vetor `Hind` tenha o comportamento de um vetor circular. Para melhor compreensão, veja o algoritmo 2, onde a função `GeraNovoPonto` recebe 4 parâmetros que são: o tamanho da vizinhança (k), o tipo de corte ($TipoCorte$), a solução que gerou a vizinhança ($xBarAntigo$) e a solução encontrada na vizinhança ($xBarNovo$). Ao final, a função retorna um novo ponto na face definida por $\overline{x}^i$ e $\overline{x}^{i+1}$ que possui distância máxima de $\overline{x}^{i+1}$ permitido pelas variações `DHM` e `DHG`.

```
1. function GeraNovoPonto(k, TipoCorte, xBarAntigo, xBarNovo)
2.    i = PG
3.    j = 0
4.    size=0
5.    H = DistanciaHamming(xBarAntigo, xBarNovo)
6.    Hind = IndicesDiferem(xBarAntigo, xBarNovo)
7.    case(TipoCorte)
8.       "DHM": size= k/2
9.       "DHG": size= k
10.   endcase
11.   while(j < H - size)
12.      xBarAntigo[Hind[i mod H]] = xBarNovo[Hind[i mod H]]
13.      i = i+1
14.      j = j+1
15.   endwhile
16.   PG = i mod H
17.   return xBarAntigo
18. end
```

Algoritmo 2: Algoritmo GeraNovoPonto

A terceira e última variação que chamaremos de `NC` (Não usa Corte), explora faces de tamanho 2^H e também é limitada pelo tempo máximo `tmax`, mas não adiciona nenhum corte canônico à formulação.

Vimos na seção 2.2.2 que para adicionarmos um corte CCF^H na formulação, devemos antes avaliar todo o hiperplano canônico correspondente, ou seja, explorar a face. Para isso, utilizaremos uma estratégia parecida com a RamLoc adicionando à formulação o corte $CCF^H(\overline{x}^i, \overline{x}^{i+1}) = 0$ e criando um sub-MIP que é enumerado pelo resolvedor. A aplicação deste CC garante que somente as soluções que satisfazem as restrições do problema e que estão na face serão avaliadas. Esta técnica de fixação é conhecida como *hard variable fixing* ou *diving*, onde valores são atribuídos às variáveis restringindo a formulação.

Como vimos na seção 2.1.3, quando não conseguimos provar a otimalidade de uma vizinhança, aplicamos o método `REFINE` para garantir que a solução computada é ótima, para o conjunto de valores assumidos pelas variáveis binárias. Como este trabalho está focado na resolução de MIPs 0–1 puros, ou seja, MIPs onde as únicas variáveis inteiras são binárias, decidimos não aplicar o método `REFINE` abrindo mão da otimalidade. Esta decisão foi baseada na necessidade de explorarmos o maior número de sub-espaços possíveis dentro dos problemas a serem resolvidos, já que iremos realizar testes com apenas uma hora de execução. Mas, se analisarmos a aplicação do método `REFINE` nos MIPs 0–1 puros, ela só é necessária quando o resolvedor encontra uma solução heurística que a parte contínua não foi otimizada. Assim, achamos que a não aplicação do método `REFINE` deve afetar (se afetar) bem pouco o valor das soluções obtidas pelas heurísticas.

O algoritmo 3 descreve a heurística que utiliza os corte canônicos. A heurística recebe 4 parâmetros que são o tamanho da vizinhança (k), o tipo de corte ($TipoCorte$), a solução que gerou a vizinhança ($xBarAntigo$) e a solução encontrada na vizinhança ($xBarNovo$). A chamada do algoritmo 3 é feita no algoritmo 1 na linha 29, no mesmo lugar onde era feito a chamada para o procedimento `REFINE`.

2.4 Experimentos e resultados

Todos os experimentos foram feitos utilizando o `XPRESS`, versão 16.10.3, em uma máquina *Pentium IV* com 3.2 GHz, 2 Gigabytes de RAM e rodando no sistema *Linux*. Devido às limitações do `XPRESS`, a implementação da RamLoc não utiliza o *presolver*. O *benchmark* utilizado nos testes é composto por 25 das 29 instâncias utilizadas em [6], correspondendo a problemas de minimização. O tempo máximo de execução foi limitado em uma hora. As instâncias `arki001`, `UMTS` e `roll3000` foram retiradas dos testes por terem variáveis inteiras não binárias e a instância `NSR8K` foi retirada por que o `XPRESS` não consegue encontrar uma resposta inteira válida, com o *presolver* desabilitado, dentro de uma hora. Os 5 métodos testados foram: o resolvedor de MIP `XPRESS` utilizando a sua configuração *default* o qual chamaremos simplesmente de `XPRESS`; a ramificação local de Fischetti e Lodi para a qual utilizamos a sigla `LB`; e os outros 3 métodos referem-se à RamLoc utilizando a nossa heurística com a variação `DHM`, `DHG` e `NC`. As informações das instâncias estão na tabela 2.1, onde n indica o número de colunas, b o número de variáveis binárias e m o número de linhas. Também pode ser visto na tabela 2.1 qual foi o valor da melhor solução e o melhor limitante dual (`Best Bound`) computado para cada instância após calibrarmos as heurísticas, além do(s) método(s) que obteve(obtiveram) o melhor limitante primal.

Como podemos ver na tabela 2.1, o `XPRESS` obteve o maior número de melhores soluções quando comparamos todos os métodos de uma única vez, após uma hora de processamento. Ele conseguiu 11 melhores respostas contra 8 do `DHM`, `DHG`, `NC` e 7 do `LB`.

```
1. function CorteCanonico(k, TipoCorte, xBarAntigo, xBarNovo);
2.    size=0
3.    H = DistanciaHamming(xBarAntigo, xBarNovo)
4.    case(TipoCorte)
5.       "DHM": size= k/2
6.       "DHG" : size= k
7.    endcase
8.    if(TipoCorte == "NC" )
9.       explora a face definida por xBarAntigo e xBarNovo
10.   endif
11.   else
12.      if(H ≤ size )
13.         explora a face definida por xBarAntigo e xBarNovo
14.         if(Explorou toda face)
15.            remove a última restrição de ramificação local
16.            insere o corte canônico CC(xBarAntigo,xBarNovo)
17.         endif
18.      endif
19.      else
20.         NovoPonto = GeraNovoPonto(k, TipoCorte, xBarAntigo, xBarNovo);
21.         explora a face definida por NovoPonto e xBarNovo
22.         if(Explorou toda face)
23.            insere o corte canônico CC(NovoPonto,xBarNovo)
24.         endif
25.      endelse
26.   endelse
27. end
```

Algoritmo 3: Algoritmo do Corte Canônico CCF^H

Instancias	n	b	m	Melhor Valor	Best Bound	Método
A1C1S1	3648	192	3312	11778.97	8211.47	DHM
A2C1S1	3648	192	3312	11047.01	7003.65	DHG
B1C1S1	3872	288	3904	25855.70	14942.92	XPRESS
B2C1S1	3872	288	3904	27817.06	13313.07	XPRESS
biella1	7328	6110	1203	3065005.78	3065005.78	XPRESS
danoint	521	56	664	65.67	63.54	XPRESS, LB, DHM, DHG, NC
glass4	322	302	396	1200012600.00	900004928.00	DHG
markshare1	62	50	7	4.00	0.00	LB
markshare2	74	60	8	13.00	0.00	XPRESS
mkc	5325	5323	3411	-556.61	-564.77	XPRESS
net12	14115	1603	14021	214.00	146.29	XPRESS, LB, DHM, DHG, NC
nsrand-ipx	6621	6620	735	51520.00	50708.34	XPRESS
rail2536c	15293	15284	2539	689.00	689.00	XPRESS
rail2586c	13226	13215	2589	964.00	936.18	DHM
rail4284c	21714	21705	4287	1086.00	1054.24	DHM
rail4872c	24656	24645	4875	1572.00	1509.72	LB
rail507	63019	63009	509	174.00	172.81	XPRESS, LB, DHM, DHG, NC
seymour	1372	1372	4944	424.00	411.31	LB, DHM, DHG
sp97ar	14101	14101	1761	665221144.64	654887168.00	NC
sp97ic	12497	12497	1033	430339881.92	424575936.00	NC
sp98ar	15085	15085	1435	530070538.88	526999520.00	DHG
sp98ic	10894	10894	825	450067753.44	447525696.00	NC
swath	6805	6724	884	467.41	424.10	XPRESS
tr12-30	1080	360	750	130872.00	101263.63	NC
van	12481	192	27331	5.09	3.72	LB, DHM, DHG, NC

Tabela 2.1: Informações das instâncias do *benchmark*

No decorrer desta seção iremos mostrar duas outras análises dos resultados obtidos de acordo com a metodologia discutida na seção 1.1. Na primeira análise utilizamos a média geométrica da razão, onde demonstramos que os métodos `NC` e `LB` obtiveram os melhores desempenhos, e, na segunda análise, realizamos comparações dos métodos aos pares em intervalos de 10 minutos e verificamos que o método `NC` obteve o melhor desempenho.

2.4.1 Calibrando o algoritmo

Como vimos anteriormente, o método de RamLoc possui dois parâmetros que devem ser definidos: o tamanho da vizinhança (k) e o tempo para explorar as vizinhanças (t). Como o *benchmark* possui 25 instâncias e o tempo destinado para explorar cada instância é de uma hora, escolhemos só 9 delas para calibrarmos os parâmetros que seriam utilizados nos testes com todas as demais. Restringimo-nos assim às instâncias `A1C1S1`, `B1C1S1`, `glass4`, `mkc`, `rail507`, `sp97ar`, `sp98ar`, `swath` e `tr12-30` que foram escolhidas ao acaso.

Os primeiros testes foram feitos variando o tamanho da vizinhança k em 20, 30 e 40, e com o tempo de exploração da vizinhança fixado em 300 segundos. Uma observação importante é que em todos os testes utilizamos `tmax`, que é o tempo para explorar as faces, igual a 50 segundos e o número de diversificações máximas (variável `dv_max` do

algoritmo 1) igual a infinito. Como todos os métodos que utilizam a RamLoc necessitam de uma solução inicial, decidimos computá-la utilizando a ferramenta tática no MIP inicial, como sugerido na função LocBra do algoritmo 1.

Para determinarmos qual é o melhor parâmetro para cada método, discretizamos o tempo de uma hora em 36 intervalos iguais. No instante de término de cada intervalo, computamos a razão entre o melhor valor obtido, até aquele momento, e o melhor valor final encontrado em todos os testes para a instância. As razões de todas as instâncias são utilizadas para o cálculo da média geométrica da razão de cada método. Este tipo de avaliação foi usado em [3] só que, deste teste aqui, todas as instâncias são analisadas em um único grupo. Assim, determinamos que o valor escolhido para o parâmetro seria aquele para o qual a heurística testada encontrasse o maior número de resultados próximos de 1 dentre os 36 intervalos da média geométrica. Os gráficos da média geométrica podem ser vistos nas figuras 2.8, 2.9, 2.10 e 2.11.

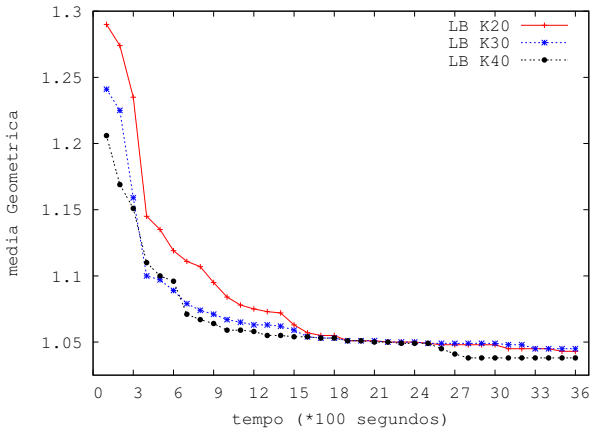

Figura 2.8: LB variando o k

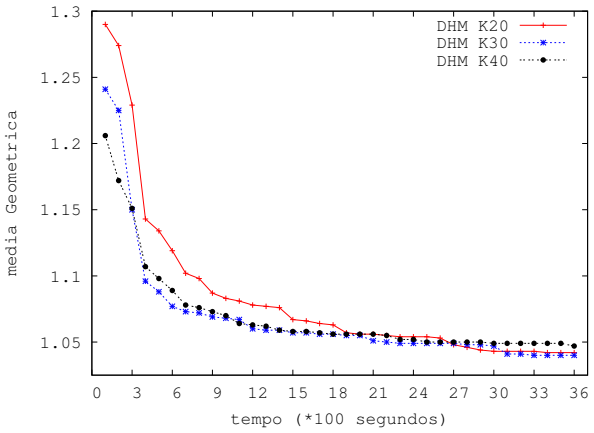

Figura 2.9: DHM variando o k

Os resultados obtidos mostram que os 4 métodos são sensíveis ao parâmetro k, sendo que o aumento deste valor proporciona uma melhora na parte inicial das 36 observações. Mesmo assim, com o critério previamente estabelecido, escolhemos $k = 20$ para os algoritmos DHG e NC, $k = 30$ para o DHM e $k = 40$ para o LB. O gráfico que mostra os melhores valores de k para cada algoritmo e o *default* do XPRESS pode ser visto na figura 2.16.

Os próximos testes foram feitos variando o tempo de exploração das vizinhanças (t) e fixando para cada método o valor de k como dito acima. Testamos os valores 40, 80, 150 e 300 segundos para t. Como podemos ver nos gráficos das figuras 2.12, 2.13, 2.14 e 2.15, os 4 métodos também são sensíveis ao parâmetro t. De um modo em geral, valores mais baixos de t proporcionam um ganho inicial maior.

Novamente, de acordo com o critério previamente fixado, escolhemos os parâmetros para rodar todas as 25 instâncias, conforme exibido na tabela 2.2.

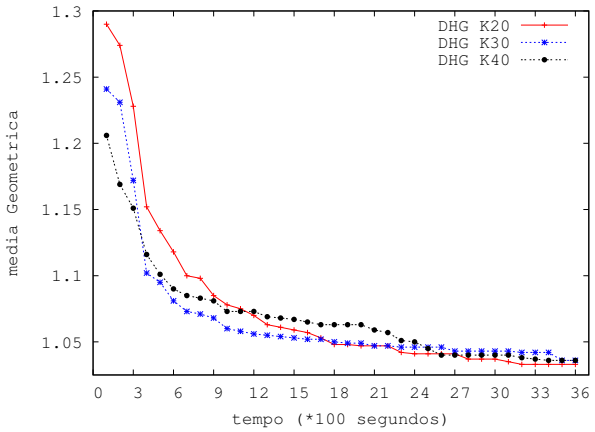

Figura 2.10: DHG variando o k

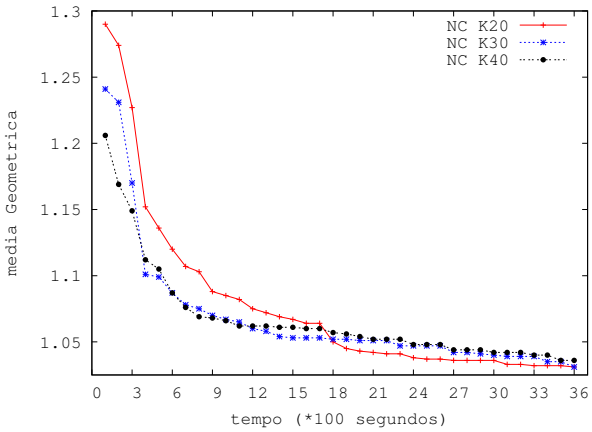

Figura 2.11: NC variando o k

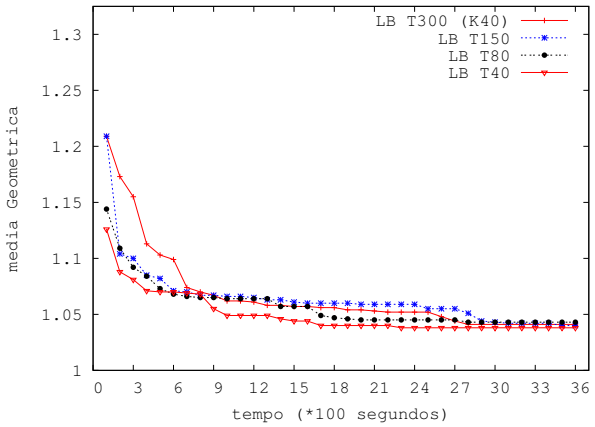

Figura 2.12: LB variando o T

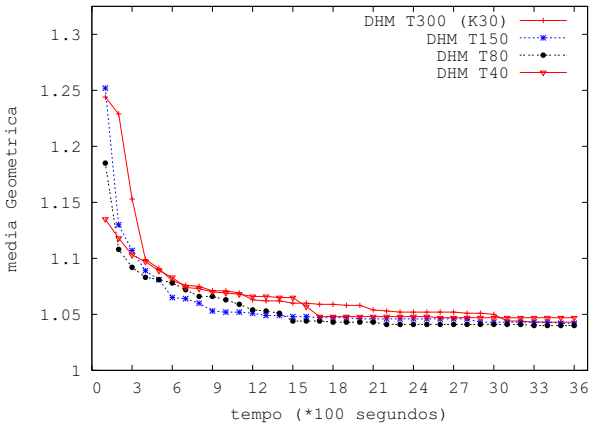

Figura 2.13: DHM variando o T

Na etapa seguinte, comparamos os resultados das diferentes implementações da RamLoc, com e sem CCs, contra o *default* do XPRESS. Verificamos que a nossa heurística e o LB demandam um tempo relativamente longo para produzirem boas soluções. Uma possível razão para tal comportamento pode ser o tempo requerido para encontrar uma primeira resposta. Outra possibilidade é a baixa qualidade da primeira resposta fornecida pelo XPRESS para iniciar os métodos. Recentemente Fischetti, Glover e Lodi [5] desenvolveram um método chamado "*feasibility pump*". O objetivo deste método é encontrar boas soluções viáveis em um baixíssimo tempo computacional. Comparando as respostas obtidas pelo XPRESS com àquelas reportadas no artigo que descreve o "*feasibility pump*", verificamos que o XPRESS possui um bom desempenho tanto na qualidade da resposta inicial, quanto no tempo para obtê-la. Assim, decidimos não implementar o "*feasibility pump*" e sim estudar a curva da média geométrica das razões do *default* do XPRESS. A

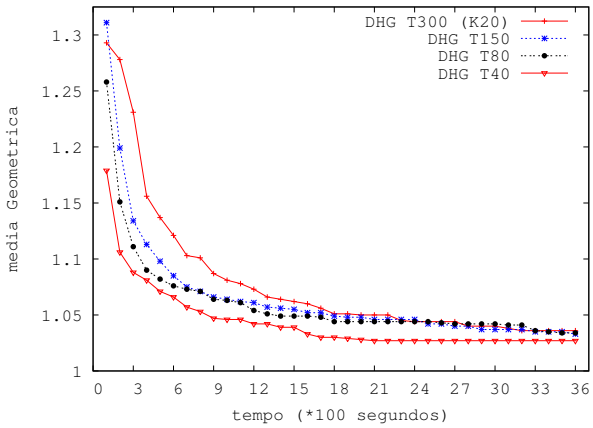

Figura 2.14: DHG variando o T

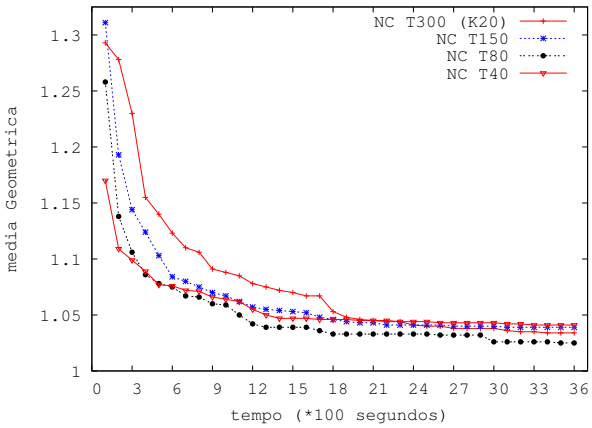

Figura 2.15: NC variando o T

Método	k	t	tmax
LB	40	40	-
DHM	30	80	50
DHG	20	40	50
NC	20	80	50

Tabela 2.2: Parâmetros

idéia foi combinar o XPRESS com os 4 métodos de ramificação local, de modo a prover estes últimos com boas soluções iniciais.

Como podemos ver na figura 2.16, o *default* do XPRESS possui uma queda considerável até os 1300 segundos, estabilizando em seguida. Então, para tentarmos melhorar o desempenho dos métodos de RamLoc, decidimos que a resposta inicial para os mesmos seria a melhor dentre aquelas alcançadas pelo XPRESS após 600 segundos de processamento. Esta escolha deveu-se à observação de que, neste tempo, o XPRESS obtinha a maior parte do ganho em relação ao custo da primeira solução. Ao mesmo tempo, como sugerido pelo gráfico, soluções alcançadas neste tempo ainda dão margem à melhorias, permitindo que as heurísticas tivessem maior facilidade de escapar de mínimos locais. Deste modo, os 4 métodos passaram a ser aplicados somente nos 3000 segundos restantes de computação. Como utilizaremos o mesmo problema carregado em memória para executar a RamLoc, não poderemos aplicar o *presolver* na parte inicial do problema, pois o XPRESS não permite adicionar, remover ou alterar restrições com *presolver* habilitado. Isto explica a possível diferença que pode ocorrer entre o *default* do XPRESS, que utiliza o *presolver*, e os demais métodos quando forem analisados os primeiros 600 segundos de execução. A tendência neste caso é que o *default* do XPRESS tenha resultados melhores.

Como podemos ver na figura 2.18, esta combinação do XPRESS com a RamLoc melho-

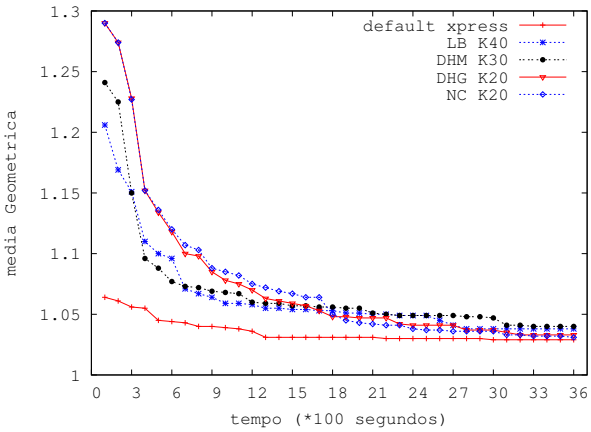

Figura 2.16: Melhores valores de k vs *default* do XPRESS

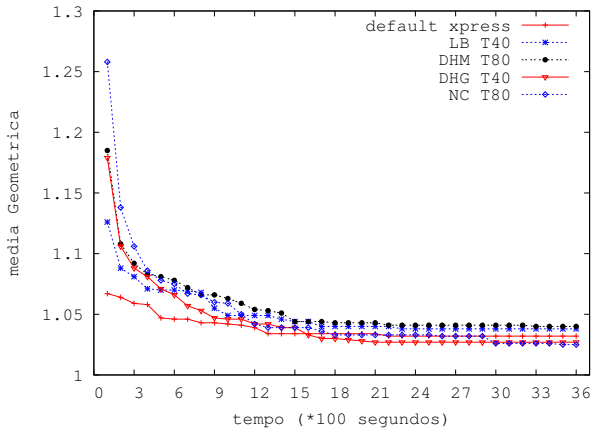

Figura 2.17: Melhores valores de T vs *default* do XPRESS

rou bastante o desempenho dos 4 métodos, sendo que os métodos DHM, DHG e NC obtiveram um resultado melhor do que o *default* do XPRESS. Vimos também que o LB obteve o pior desempenho entre os 4 métodos.

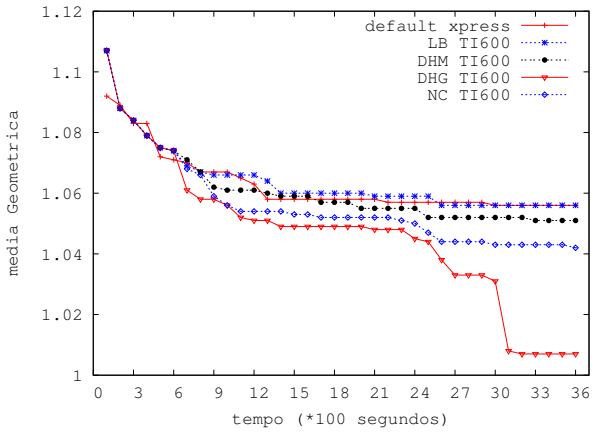

Figura 2.18: Comparando os métodos combinados com o *default* do XPRESS

Tendo combinado os métodos com o XPRESS e fixados os parâmetros escolhidos na tabela 2.2, realizamos os testes com o conjunto completo das 25 instâncias.

2.4.2 Análise dos resultados

Como descrevemos anteriormente, aplicamos o método de avaliação que foi usado em [3]. Para isso, dividimos os resultados das instâncias em dois diferentes grupos. No grupo de "propagação pequena" encontram-se as instâncias onde o *gap* da razão entre a pior e a melhor solução final obtida entre os 4 métodos e o *default* do `XPRESS` é menor do que 10%. Já no grupo de "propagação média", este *gap* é maior do que 10%. As informações destas razões computadas para as instâncias, após uma hora, podem ser consultadas na tabela 2.3, onde é possível observar qual foi o desempenho final de cada método em relação à melhor solução obtida nas 25 instâncias e quais instâncias pertencem ao grupo de "propagação pequena" e de "propagação média".

Instância	Normal	LB	DHM	DHG	NC
Propagação pequena					
A1C1S1	1.018	1.021	**1.000**	1.010	1.008
A2C1S1	1.068	1.030	1.014	**1.000**	1.004
B1C1S1	**1.000**	1.015	1.015	1.003	1.008
biella1	**1.000**	1.003	**1.000**	1.002	**1.000**
danoint	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
mkc	**1.000**	1.002	1.001	1.001	1.002
net12	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
nsrand-ipx	**1.000**	1.031	1.031	1.031	1.022
rail2536c	**1.000**	1.001	1.001	1.001	1.001
rail2586c	1.006	1.005	**1.000**	1.008	1.004
rail507	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
seymour	1.002	**1.000**	**1.000**	**1.000**	1.002
sp97ar	1.001	1.009	1.001	1.006	**1.000**
sp97ic	1.002	1.020	1.025	1.025	**1.000**
sp98ar	1.005	1.007	1.002	**1.000**	1.002
sp98ic	**1.000**	1.010	1.003	1.005	**1.000**
swath	**1.000**	1.046	1.025	1.026	1.022
tr12-30	1.030	1.004	1.002	**1.000**	**1.000**
Propagação média					
B2C1S1	**1.000**	1.101	1.101	1.035	1.008
glass4	1.528	1.458	1.472	**1.000**	1.375
markshare1	2.000	**1.000**	1.750	1.750	1.750
markshare2	**1.000**	1.615	1.615	1.615	1.615
rail4284c	1.165	1.005	**1.000**	1.027	1.008
rail4872c	1.150	**1.000**	1.005	1.001	1.008
van	1.205	**1.000**	**1.000**	**1.000**	**1.000**

Tabela 2.3: Razão, após uma hora, entre a solução do método e a melhor encontrada

Analisando o gráfico de "propagação pequena" da figura 2.19, podemos ver que as nossas heurísticas obtiveram os melhores resultados, sendo a variante `NC` a de melhor desempenho. Vimos também que o `LB` obteve um resultado geral melhor do que o *default* do `XPRESS` perdendo somente após 2900 segundos. Avaliando o gráfico de "propagação média" da figura 2.20, verificamos que o algoritmo com o melhor desempenho foi o `LB` seguido pelas 3 variações de nossa heurística.

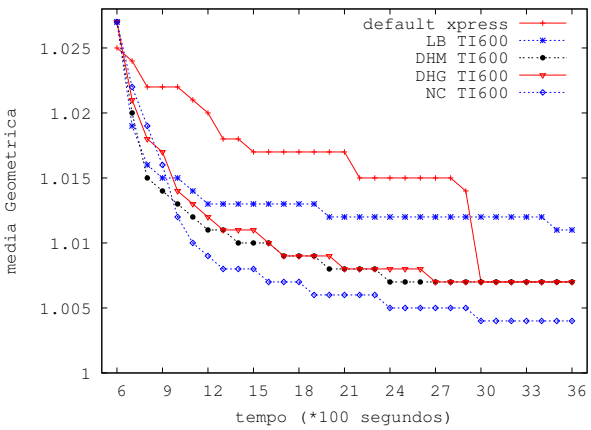

Figura 2.19: Execução das 18 instâncias do grupo de "propagação pequena"

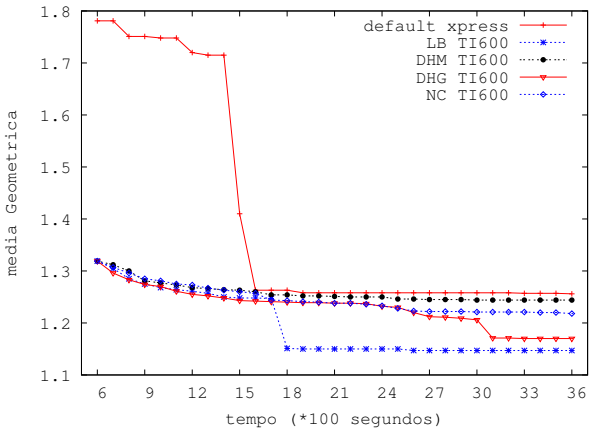

Figura 2.20: Execução das 7 instâncias do grupo de "propagação média"

Outra análise interessante é a comparação dos resultados dos métodos aos pares. Neste caso, avaliamos a evolução deles em intervalos de tempo de 10 minutos, classificando aquele que obtiver o maior número de melhores soluções como sendo o melhor. Nesta análise, consideramos os dados apresentados na tabela 2.4. A coluna `comparações` indica quais métodos serão comparados, enquanto as demais indicam o número de melhores soluções de cada método e o número de empates.

Analisando os valores da tabela 2.4 nos 10 primeiros minutos, verificamos que para este conjunto de instâncias o `XPRESS` sem o *presolver* obteve um maior número de melhores soluções do que o *default* do `XPRESS` (com o *presolver*). Esta informação pode ser retirada do `LB` e das nossas 3 variações que utilizam apenas o `XPRESS` sem *presolver* nos primeiros 10 minutos. Isto ocorre porque, conforme foi destacado anteriormente, as heurísticas de ramificação local só passam a atuar após passados 600 segundos de execução do resolvedor de MIP.

Verificando o restante da tabela, podemos ver que o `NC` sempre obteve os melhores resultados durante os 60 minutos, em relação aos demais métodos. Em relação ao *default* do `XPRESS`, ele perde para os demais métodos nos primeiros 20 minutos, recuperando-se após 30 minutos, quando consegue resultados melhores do que o `LB` e `DHG`. Uma informação importante de se enfatizar é que os nossos 3 métodos sempre obtiveram resultados melhores do que o `LB` em todos os intervalos.

Como vimos na seção 2.1.3, quando o a RamLoc chama o resolvedor para avaliar uma vizinhança (chamada `MIP_SOLVE` linha 7 do algoritmo 1), ele retorna na variável `stat` quatro possíveis casos:

- `caso1 = solução_ótima_encontrada`;

	10 minutos						20 minutos					
Comparações	Xpress	LB	DHM	DHG	NC	Empate	Xpress	LB	DHM	DHG	NC	Empate
Xpress x LB	10	12	-	-	-	3	11	11	-	-	-	3
Xpress x DHM	10	-	12	-	-	3	7	-	15	-	-	3
Xpress x DHG	10	-	-	12	-	3	8	-	-	13	-	4
Xpress x NC	10	-	-	-	12	3	5	-	-	-	16	4
LB x DHM	-	0	1	-	-	24	-	3	9	-	-	13
LB x DHG	-	0	-	1	-	24	-	5	-	11	-	9
LB x NC	-	0	-	-	1	24	-	4	-	-	12	9
DHM x DHG	-	-	0	1	-	24	-	-	8	7	-	10
DHM x NC	-	-	0	-	1	24	-	-	6	-	9	10
DHG x NC	-	-	-	0	0	25	-	-	-	7	9	9

	30 minutos						40 minutos					
Comparações	Xpress	LB	DHM	DHG	NC	Empate	Xpress	LB	DHM	DHG	NC	Empate
Xpress x LB	13	9	-	-	-	3	13	9	-	-	-	3
Xpress x DHM	10	-	12	-	-	3	10	-	12	-	-	3
Xpress x DHG	12	-	-	10	-	3	11	-	-	11	-	3
Xpress x NC	8	-	-	-	13	4	9	-	-	-	12	4
LB x DHM	-	4	9	-	-	12	-	6	9	-	-	10
LB x DHG	-	4	-	12	-	9	-	4	-	12	-	9
LB x NC	-	5	-	-	13	7	-	4	-	-	13	8
DHM x DHG	-	-	7	8	-	10	-	-	9	6	-	10
DHM x NC	-	-	6	-	11	8	-	-	7	-	11	7
DHG x NC	-	-	-	9	9	7	-	-	-	6	11	8

	50 minutos						60 minutos					
Comparações	Xpress	LB	DHM	DHG	NC	Empate	Xpress	LB	DHM	DHG	NC	Empate
Xpress x LB	14	8	-	-	-	3	13	9	-	-	-	3
Xpress x DHM	10	-	12	-	-	3	11	-	11	-	-	3
Xpress x DHG	12	-	-	10	-	3	12	-	-	10	-	3
Xpress x NC	9	-	-	-	12	4	8	-	-	-	13	4
LB x DHM	-	4	11	-	-	10	-	4	11	-	-	10
LB x DHG	-	4	-	13	-	8	-	5	-	12	-	8
LB x NC	-	4	-	-	14	7	-	4	-	-	14	7
DHM x DHG	-	-	8	7	-	10	-	-	8	7	-	10
DHM x NC	-	-	6	-	12	7	-	-	6	-	12	7
DHG x NC	-	-	-	6	12	7	-	-	-	7	11	7

Tabela 2.4: Comparação dos métodos aos pares

- `caso2 = provado_inviabilidade`;
- `caso3 = solução_possível_encontrada`;
- `caso4 = nenhuma_solução_possível_encontrada`.

A soma dos retornos feitos em todas as instâncias pelo resolvedor para cada método podem ser vistos na na tabela 2.5, que também informa o número de diversificações sofridas (`DV`).

Percorrendo a tabela 2.5, podemos ver que o número de retornos feitos pelo resolvedor no método `LB` para o `caso1` e o `caso2`, em relação aos demais, é pequeno. Isto indica que o número de cortes profundos aplicados à formulação é baixo, pois no `caso3` são

	Caso1	Caso2	Caso3	Caso4	DV	Melhorou	N.Faces	M	MF	CC
LB	5	22	1029	828	350	15	-	-	-	-
DHM	20	9	582	508	183	16	557	221	18	553
DHG	48	1608	998	952	1953	18	973	441	54	965
NC	63	28	544	468	174	19	519	224	34	-

Tabela 2.5: Estatísticas dos métodos

aplicados somentes cortes do tipo $H(x, \overline{x}) \geq 1$ e, no `caso4`, não são aplicados cortes. Esta característica, como haviamos comentado anteriormente, é a justificativa deste trabalho.

Verificando as nossas variações, podemos ver que nas colunas `caso2` e `DV` do método `DHG`, apareceram valores bem discrepantes em relação aos outros métodos. Esta anomalia aconteceu durante a execução da instância `net12`. Analisando a execução do método nesta instância, constatamos que ela estava normal até os 3089 segundos de execução. Passado este tempo, após provar a inviabilidade de uma vizinhança (retorno `caso2`), o algoritmo aplicou o critério de diversificação e entrou em *loop*, pois toda vizinhança que era selecionada pela diversificação também era resolvida em menos de um segundo retornando `caso2`. Este processo continuou até os 3600 segundos, onde foram geradas 1592 vizinhanças. Uma possível explicação para o ocorrido é a possibilidade do espaço de solução ter sido totalmente explorado, pois a RamLoc não possui um mecanismo que consiga detectar esta característica quando o número de diversificações máxima é infinito, gerando este tipo de *loop*. Uma outra característica que fortalece está hipótese é que a solução computada é a melhor solução encontrada por todos os métodos, como podemos ver na tabela 2.1. Assim, se não levarmos em conta a anomalia ocorrida na execução do método `DHG`, verificamos que todos os nossos métodos possuem o comportamento parecido com a `LB`, em relação ao retorno dos 4 casos. A diferença é que na quase totalidade das ocorrências do `caso3`, para os métodos `DHM` e `DHG`, conseguimos aplicar um corte mais profundo, como podemos ver na coluna `CC` que representa os cortes canônicos aplicados na formulação.

Como discutimos anteriormente, a aplicação dos cortes canônicos está ligado à exploração das faces que eles definem. As informações das faces exploradas também estão na tabela 2.5, onde podemos ver para cada método o número de faces exploradas (`N.Faces`), a quantidade de faces que obtiveram uma solução melhor do que a encontrada na vizinhança de RamLoc (`M`) e o número de faces onde a melhor solução encontrada era a melhor até o momento (`MF`).

Como podemos ver, em 43% das faces exploradas, o resolvedor conseguiu encontrar uma solução melhor do que a vizinhança de RamLoc que a gerou. Além disso, 11% destas soluções correspondiam à melhor solução encontrada até aquele ponto.

Por fim, analisaremos a capacidade dos métodos de melhorar a resposta inicial dada

pelo `XPRESS` após 600 segundos de execução. Esta informação está na tabela 2.5 na coluna `Melhorou`. A informação na tabela, nos mostra como o `XPRESS` é um software bem robusto pois, em 32% das instâncias os métodos não conseguiram melhorar a solução inicial. Vimos também que o método `NC` foi o que obteve o melhor desempenho, pois só não melhorou 6 das 25 respostas iniciais passadas para ele.

Capítulo 3

A Heurística RINS

Recentemente, Danna, Rothberg e Le Pape apresentaram uma heurística de busca local chamada RINS - *Relaxation Induced Neighborhood Search* [3] - que explora vizinhanças que são criadas avaliando as informações da relaxação do nó corrente da árvore de *branch-and-cut* junto com a melhor solução inteira viável encontrada até aquele momento. Uma grande vantagem da RINS é que ela pode ser inserida nos algoritmos de *branch-and-cut* dos resolvedores de programação inteira mista (MIP), sem contudo alterar o modo básico de operação da enumeração, como é o caso da Ramificação Local apresentada no capítulo 2. As vizinhanças da RINS são descritas por sub-MIPs cujas soluções inteiras são soluções viáveis para o MIP original. Assim, elas correspondem a limitantes úteis para a poda de nós na árvore de *branch-and-cut* do resolvedor, o que ajuda a acelerar o método. Para desenvolver a RINS, os autores basearam-se em algumas idéias da heurística de Ramificação Local [6], entre elas a descrição de vizinhanças por meio de sub-MIPs e a sua exploração feita por meio do resolvedor de MIPs. Experimentos computacionais relatados em [3] mostraram que a RINS teve um ótimo desempenho, quando implementada no resolvedor comercial `CPLEX`.

3.1 Detalhando a RINS

Quando exploramos uma árvore de *branch-and-cut*, duas soluções estão tipicamente à nossa disposição: o *incumbent*, que é a melhor solução inteira conhecida até aquele momento, e a solução fracionária do nó corrente. O *incumbent* é uma solução viável e, portanto, atende às restrições de integralidade. Ao fim da execução, ele equivale a uma solução ótima. Inversamente, a solução da relaxação contínua do nó corrente é, na maioria das vezes, não inteira, mas o seu valor na função objetivo é sempre melhor do que a do *incumbent*.

Comparando o valor das variáveis das duas soluções, podemos ver que algumas delas

assumem valores diferentes entre o *incumbent* e a relaxação, mas muitas assumem o mesmo valor. A RINS é baseada na intuição de que as variáveis com mesmo valor entre as duas soluções formam uma solução parcial que pode ser estendida para uma solução completa e inteira.

A definição do algoritmo da heurística RINS é bem simples. Em um nó na árvore global de *branch-and-cut*, após encontrar a primeira solução inteira, os seguintes passos são executados:

1. Crie um sub-MIP, derivado do MIP original, onde as variáveis que tenham valores iguais entre o *incumbent* e a relaxação contínua corrente são fixadas.

2. Configure o *cutoff*, ou seja, o valor de corte para os nós da árvore de *branch-and-cut*, que será explorada na resolução do sub-MIP descrito no passo anterior, para o valor do *incumbent* corrente.

3. Resolva o sub-MIP criado.

Estes sub-MIPs tiram proveito dos planos de cortes e dos limitantes globais adicionados pelo processo de *branch-and-cut*, que está resolvendo o MIP original. Porém, não se restringem à sub-árvore do nó corrente, na medida em que descartam do modelo as restrições impostas pelas ramificações (*branchings*) que definem este nó.

Uma dificuldade em relação à criação dos sub-problemas é que não é possível a *priori* determinar o tamanho da árvore de enumeração que será gerada para resolvê-los ou mesmo a dificuldade para computá-los. Assim, para evitar que o resolvedor gaste muito tempo no cálculo dos sub-MIPs, é utilizado um parâmetro nl que limita o número máximo de nós a serem explorados na sua resolução.

Um outro fator importante em uma estratégia de busca local é a diversificação. Geralmente quando o resolvedor modifica de um nó para outro, mesmo não alterando o *incumbent*, o valor da relaxação contínua modifica. Assim, se a RINS for chamada a cada nó, uma diversificação no espaço de busca estará sendo realizada. Apesar disso, tipicamente as vizinhanças induzidas pela relaxação de nós consecutivos são muito similares. Então, para melhorar a diversificação é utilizado o parâmetro $f >> 1$ que determina que a RINS seja aplicada a cada f nós explorados durante a enumeração. Note que mesmo utilizando um valor uniforme e alto para f há uma possibilidade remota de que múltiplos nós do MIP correspondam ao mesmo sub-MIP ou, similarmente, um sub-MIP possa ser idêntico a uma sub-árvore da árvore global do MIP. Porém, a RINS não possui qualquer mecanismo para evitar esta situação.

Uma grande vantagem da RINS construir suas vizinhanças como sub-MIPs, é que qualquer solução encontrada nela também é uma solução para o MIP original. Assim, quando uma exploração de um sub-MIP termina (porque é inviável ou foi provado a sua

otimalidade ou os nl nós tenham sido explorados), o *incumbent* do MIP global é atualizado pela melhor solução inteira encontrada no sub-MIP (se tiver alguma), e a exploração do MIP global é continuada. É importante notar que o único efeito da exploração de um sub-MIP RINS na execução do MIP original é a obtenção de um possível novo *incumbent* fora, evidentemente, um eventual aumento do tempo total de computação.

3.2 Cortes canônicos na RINS

Como vimos no capítulo 1, os cortes canônicos fixam variáveis binárias formando faces que são eliminadas ao aplicar os cortes. Por outro lado, na definição da RINS, vimos que ela gera sub-MIPs fixando qualquer tipo de variável, inteira ou contínua, desde que ela possua o mesmo valor na solução relaxada do nó corrente e no *incumbent*, como podemos ver na figura 3.1. Desse modo, para que possamos utilizar os cortes canônicos, devemos limitar a fixação das variáveis da RINS, no passo 1 do algoritmo, somente às variáveis binárias (figura 3.2). Esta limitação nos permite adicionar os cortes canônicos à formulação do MIP, desde que, o resolvedor consiga explorar todo o sub-MIP, pois os sub-problemas gerados representam hiperplanos canônicos.

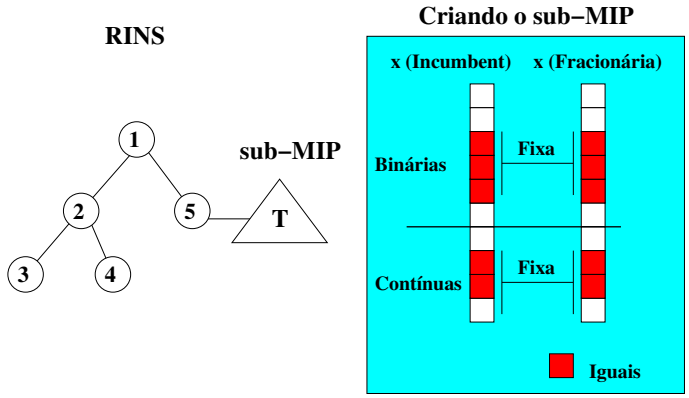

Figura 3.1: `RINS` com $f = 5$

O corte canônico que iremos aplicar é definido a seguir. Suponha que duas soluções estejam disponíveis, a relaxação do nó corrente $\tilde{x}$ e o *incumbent* $\overline{x}$. Seja D o número de variáveis binárias que possuem o mesmo valor entre $\tilde{x}$ e $\overline{x}$. Defina F^D como sendo a face de W_n formado por todos vetores binários de x tais que $x_j = \tilde{x}_j = \overline{x}_j$. O corte canônico

que elimina todos vetores binários que estão à distância $\leq d$ de F^D é dado por

$$\sum_{j|\tilde{x}_j=\overline{x}_j=1} x_j - \sum_{j|\tilde{x}_j=\overline{x}_j=0} x_j \leq \sum_{j|\tilde{x}_j=\overline{x}_j=1} \overline{x}_j - d. \tag{3.1}$$

Como estamos querendo eliminar somente as soluções viáveis do sub-problema criado pela RINS, iremos aplicar o CC com $d = 1$. Assim, toda vez que um sub-MIP for totalmente explorado, adicionamos o corte canônico que o elimina na formulação do problema.

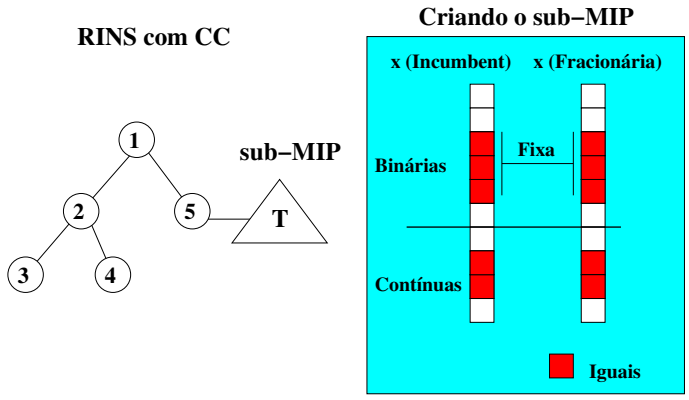

Figura 3.2: `RINS com CC` com $f = 5$

3.3 Experimentos e resultados

Para avaliarmos a capacidade da RINS com e sem os cortes canônicos, utilizamos o resolvedor comercial `XPRESS`, versão 16.10.3, em uma máquina *Pentium IV* com 3.2 GHz, 2 Gigabytes de RAM e rodando no sistema *Linux*. O *benchmark* utilizado nos testes é o mesmo da seção 2.4, onde utilizamos 25 das 29 instâncias utilizadas em [6]. Todas instâncias referem-se a problemas de minimização e o tempo máximo de execução foi limitado em uma hora. Devido ao grau de liberdade fornecido pelos parâmetros da RINS, realizamos alguns testes para calibrarmos o método. Estes testes estão descritos a seguir.

3.3.1 Calibrando o algoritmo

O algoritmo da RINS utiliza dois parâmetros: o número de nós explorados nos sub-MIPs nl e a freqüência de aplicação da heurística f. Como queremos determinar bons parâmetros para a função da RINS no `XPRESS`, decidimos realizar testes utilizando os

parâmetros usados no artigo da RINS [3] e realizando uma variação nestes valores. Como no capítulo da Ramificação Local, iremos utilizar as mesmas 9 instâncias e o mesmo modo de avaliação para calibrar a RINS. Restringimo-nos assim às instâncias `A1C1S1`, `B1C1S1`, `glass4`, `mkc`, `rail507`, `sp97ar`, `sp98ar`, `swath` e `tr12-30`. Os primeiros testes foram realizados utilizando o parâmetro de número de nós $nl = 1000$ e variando a freqüência f em 50, 100 e 150.

Para determinarmos qual é o melhor parâmetro para cada método, discretizamos o tempo de uma hora em 36 intervalos iguais. No instante de término de cada intervalo, computamos a razão entre o melhor valor obtido, até aquele momento e o melhor valor final encontrado em todos os testes para a instância. As razões de todas as instâncias são utilizadas para o cálculo da média geométrica da razão de cada método. Este método de avaliação foi usado em [3], só, que neste caso, todas as instâncias estão agrupadas no mesmo grupo. Assim, determinamos que o valor escolhido para o parâmetro seria aquele para o qual a heurística testada, encontrasse o maior número de resultados próximos de 1 dentre os 36 intervalos da média geométrica.

O gráfico da média geométrica, onde variamos o valor de f, pode ser visto na figura 3.3. Como podemos ver, a curva com o parâmetro $f = 50$ obteve o melhor resultado.

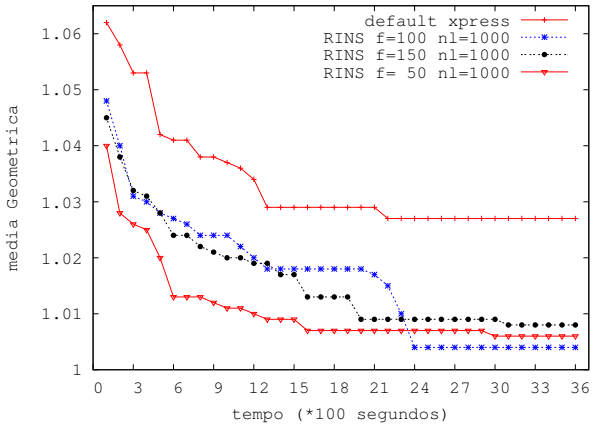

Figura 3.3: `RINS` variando o parâmetro f

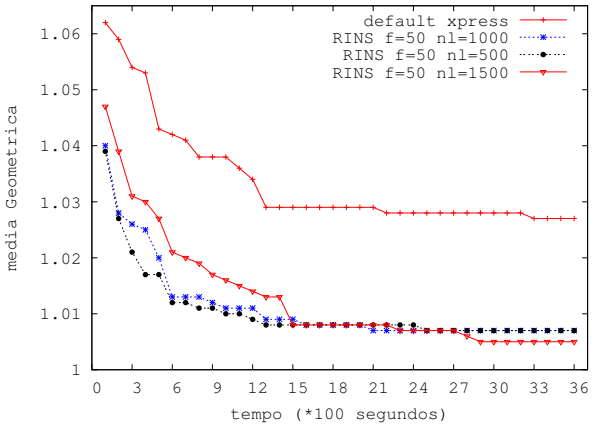

Figura 3.4: `RINS` variando o parâmetro nl

Os testes seguintes, foram feitos variando o valor de nl em 500, 1000, 1500 e fixando o valor de f em 50. O gráfico da média geométrica pode ser visto na figura 3.4. Assim, verificamos que, dentre os valores testados, o melhor conjunto de parâmetros é $f = 50$ e $nl = 500$. Interessante notar que a RINS obteve o resultado melhor do que o *default* do `XPRESS` em todos os testes.

Tendo calibrado a heurística, realizamos os testes com a totalidade das 25 instâncias do *benchmark*. Também realizamos testes aplicando os cortes canônicos na RINS, que chamaremos de RINSCC, utilizando os mesmos parâmetros da RINS e o *default* do `XPRESS`.

As informações das instâncias estão na tabela 3.1, onde n indica o número de colunas, b o número de variáveis binárias e m o número de linhas. Também pode ser visto na tabela 3.1 qual foi o valor da melhor solução e o melhor limitante dual (`Best Bound`) computado para cada instância, após a calibragem, além do(s) método(s) que a obteve(obtiveram) o melhor limitante primal.

Instancias	n	b	m	Melhor Valor	Best Bound	Método
A1C1S1	3648	192	3312	11812.13	8211.47	RINSCC
A2C1S1	3648	192	3312	11092.18	7003.65	RINSCC
B1C1S1	3872	288	3904	24916.91	14942.92	RINSCC
B2C1S1	3872	288	3904	26583.46	13169.76	RINS
biella1	7328	6110	1203	3065005.78	3065005.78	XPRESS, RINS
danoint	521	56	664	65.67	63.54	XPRESS, RINSCC
glass4	322	302	396	1640014740.00	900004928.00	RINS
markshare1	62	50	7	8.00	0.00	XPRESS, RINSCC
markshare2	74	60	8	13.00	0.00	XPRESS, RINSCC
mkc	5325	5323	3411	-563.01	-564.77	RINS
net12	14115	1603	14021	214.00	146.29	XPRESS, RINS, RINSCC
nsrand-ipx	6621	6620	735	51520.00	50708.34	XPRESS
rail2536c	15293	15284	2539	689.00	689.00	XPRESS
rail2586c	13226	13215	2589	960.00	936.18	RINS, RINSCC
rail4284c	21714	21705	4287	1083.00	1054.24	RINS, RINSCC
rail4872c	24656	24645	4875	1564.00	1510.04	RINS, RINSCC
rail507	63019	63009	509	174.00	172.81	XPRESS, RINS
seymour	1372	1372	4944	424.00	411.31	RINS, RINSCC
sp97ar	14101	14101	1761	665772913.92	654887168.00	XPRESS
sp97ic	12497	12497	1033	431024958.88	424599072.00	RINS
sp98ar	15085	15085	1435	530566673.44	526999520.00	RINS
sp98ic	10894	10894	825	450280257.28	447525696.00	XPRESS, RINS
swath	6805	6724	884	467.41	424.10	XPRESS
tr12-30	1080	360	750	130640.00	99565.79	RINS
van	12481	192	27331	5.09	3.72	RINS, RINSCC

Tabela 3.1: Informações das instâncias do *benchmark*

3.3.2 Análise dos resultados

Como podemos ver na tabela 3.1, a RINS obteve, após uma hora de execução, 15 melhores respostas contra 12 da RINSCC e 11 do *default* do `XPRESS`. Esta análise só avalia o estado final da execução. Para avaliarmos o comportamento dos métodos durante todo o processo, utilizamos duas outras avaliações que é a média geométrica das razões e a comparação dos métodos aos pares.

Da mesma forma que o artigo da RINS [3], utilizamos a média geométrica das razões na avaliação de desempenho dos métodos. Para isso, dividimos os resultados da instâncias em dois diferentes grupos. O de "propagação pequena" correspondentes às instâncias onde o *gap* da razão entre a pior e a melhor solução final obtida entre os 3 métodos é menor do que 10% e a de "propagação média" onde o *gap* é maior do que 10%. As informações

das razões das 25 instâncias após uma hora de processamento podem ser consultadas na tabela 3.2, onde é possível observar qual foi o desempenho final de cada método em relação à melhor solução obtida entre eles e quais instâncias pertencem ao grupo de "propagação pequena" e "propagação média".

Instâncias	**XPRESS**	**RINS**	**RINSCC**
Propagação pequena			
A1C1S1	1.015	1.001	**1.000**
A2C1S1	1.064	1.007	**1.000**
B1C1S1	1.038	1.016	**1.000**
B2C1S1	1.046	**1.000**	1.008
biella1	**1.000**	**1.000**	1.008
danoint	**1.000**	1.013	**1.000**
mkc	1.011	**1.000**	1.010
net12	**1.000**	**1.000**	**1.000**
nsrand_ipx	**1.000**	1.012	1.047
rail2586c	**1.000**	1.001	1.003
rail2586c	1.010	**1.000**	**1.000**
rail507	**1.000**	**1.000**	1.006
seymour	1.002	**1.000**	**1.000**
sp97ar	**1.000**	1.003	1.005
sp97ic	**1.000**	**1.000**	1.017
sp98ar	1.005	**1.000**	1.005
sp98ic	**1.000**	**1.000**	1.014
tr12-30	1.032	**1.000**	**1.000**
Propagação média			
glass4	1.118	**1.000**	1.211
markshare1	**1.000**	2.000	**1.000**
markshare2	**1.000**	1.615	**1.000**
rail4284c	1.168	**1.000**	**1.000**
rail4872c	1.156	**1.000**	**1.000**
swath	**1.000**	1.008	1.105
van	1.205	**1.000**	**1.000**

Tabela 3.2: Razão, após uma hora, entre a solução do método e a melhor encontrada

A análise do gráfico de "propagação pequena" figura 3.5, mostra que a RINS obteve o melhor desempenho ficando o RINSCC em segundo lugar. Agora, quando avaliamos o gráfico de "propagação média" figura 3.6, verificamos que o algoritmo da RINSCC obteve os melhores resultados, sendo que a RINS obteve o pior desempenho na última meia hora.

Analisamos também os 3 métodos avaliando qual deles obteve o maior número de melhores soluções a cada 10 minutos de processamento. Para isso, avaliamos os métodos aos pares onde classificamos aquele que obtiver o maior número de melhores soluções como sendo o melhor. Os resultados obtidos podem ser vistos na tabela 3.3, que possui o campo `comparações` indicando quais os dois métodos serão comparados, o número de melhores soluções que cada método obteve e o número de empates.

Como podemos ver, o método RINS obteve durante todas as 6 parciais o maior número de melhores soluções quando comparado com os demais métodos. Em relação aos métodos

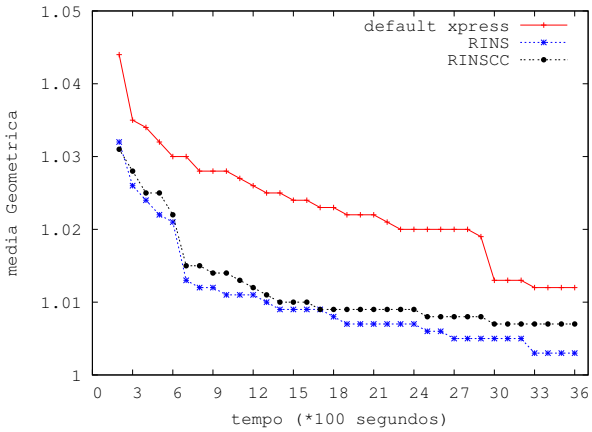

Figura 3.5: Execução das 18 instâncias do grupo de "propagação pequena"

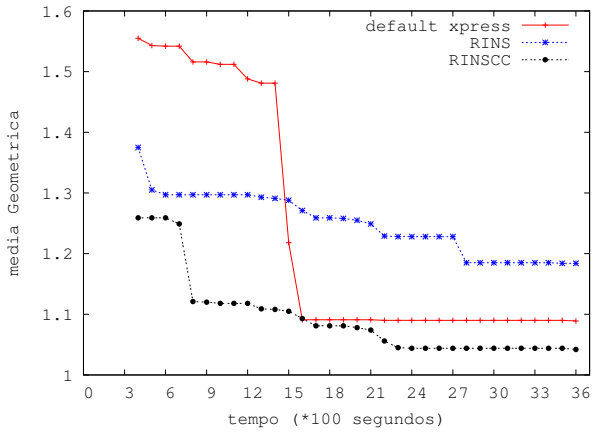

Figura 3.6: Execução das 7 instâncias do grupo de "propagação média"

RINSCC e o *default* do `XPRESS`, o RINSCC obteve o melhor desempenho nos primeiros 10 minutos e na meia hora final.

Algumas estatísticas importantes em relação à RINS e RINSCC, podem ser vistas na tabela 3.4, onde é apresentado número de sub-problemas criados (`Sub-prob`), o número de sub-problemas onde foram encontrados alguma solução viável (`viáveis`) e o número de cortes canônicos adicionados na formulação (`CC`).

Como podemos ver, o número de sub-problemas diminuiu na RINSCC em relação à RINS. Isto pode ocorrer devido a alguns fatores. O primeiro refere-se à fixação de somente variáveis binárias na definição dos sub-MIPs associados às vizinhanças e o segundo refere-se ao número de cortes canônicos adicionados. No primeiro caso, ao fixarmos somente as variáveis binárias podemos estar criando sub-MIPs muito difíceis, pois o número de variáveis fixadas se reduz em relação à RINS original. Podemos ver isto em instâncias onde o número de variáveis binárias são bem menores do que as contínuas, Alguns exemplos são as instância `danoint` onde na RINS foram analisados 1457 sub-problemas contra 188 na RINSCC e a instância `tr12-30` com 850 na RINS contra 152 na RINSCC. Em relação ao segundo caso, os cortes canônicos adicionados são restrições densas. Como mostra a literatura, os problemas lineares tornam-se usualmente mais difíceis de serem resolvidos quando as desigualdades que os definem são densas. O aumento do tempo necessário para computar os programas lineares durante a execução dos algoritmos de *branch-and-bound* e *branch-and-cut* acaba por impactar o desempenho destes procedimentos.

	10 minutos				20 minutos			
Comparações	Xpress	RINS	RINSCC	EMPATE	Xpress	RINS	RINSCC	EMPATE
Xpress x RINS	10	13	-	2	9	13	-	3
Xpress x RINSCC	10	-	12	3	12	-	10	3
RINS x RINSCC	-	9	9	7	-	11	8	6

	30 minutos				40 minutos			
Comparações	Xpress	RINS	RINSCC	EMPATE	Xpress	RINS	RINSCC	EMPATE
Xpress x RINS	9	14	-	2	8	14	-	3
Xpress x RINSCC	11	-	10	4	10	-	11	4
RINS x RINSCC	-	11	8	6	-	11	8	6

	50 minutos				60 minutos			
Comparações	Xpress	RINS	RINSCC	EMPATE	Xpress	RINS	RINSCC	EMPATE
Xpress x RINS	7	14	-	4	7	14	-	4
Xpress x RINSCC	10	-	11	4	10	-	11	4
RINS x RINSCC	-	13	6	6	-	13	6	6

Tabela 3.3: Comparação dos métodos aos pares

	RINS		**RINSCC**		
Instâncias	**Sub-prob**	**Viáveis**	**Sub-prob**	**Viáveis**	**CC**
A1C1S1	453	39	360	8	0
A2C1S1	455	35	380	12	0
B1C1S1	289	17	89	9	1
B2C1S1	239	22	130	8	0
biella1	91	4	4	3	2
danoint	1457	0	188	1	65
glass4	4265	13	204	15	147
markshare1	17317	9	2868	9	774
markshare2	15187	3	7888	7	50
mkc	3718	24	33	14	33
net12	41	0	17	0	14
nsrand_ipx	567	15	15	9	14
rail2536c	29	2	1	1	1
rail2586c	6	4	6	4	0
rail4284c	1	1	1	1	0
rail4872c	2	2	2	2	0
rail507	17	1	3	0	3
seymour	48	4	26	4	4
sp97ar	136	15	12	7	8
sp97ic	511	14	19	8	17
sp98ar	230	20	17	10	13
sp98ic	612	12	21	5	21
swath	3147	5	53	2	51
tr12-30	850	29	152	23	68
van	5	0	1	0	0

Tabela 3.4: Estatísticas dos métodos

Capítulo 4

Utilização dos Cortes Canônicos como Ramificação

Durante o desenvolvimento desta dissertação, utilizamos os CCs como planos de corte nos métodos de RamLoc e RINS. Uma outra possibilidade de aplicação dos CCs, é utilizá-los em uma estratégia de ramificação do método de *branch-and-cut* de um resolvedor. A idéia é introduzir alguns "mergulhos" no modo de exploração da árvore de enumeração, caracterizado por um procedimento agressivo de fixação de variáveis binárias do modelo que está sendo tratado. Denominaremos esta estratégia de *dive branching* (DB), já que a fixação de variáveis imposta pelos CCs assemelha-se à forma como são fixadas variáveis na heurística de *dive* (cf., [15]).

Para um nó na árvore de *branch-and-cut*, realizaremos uma ramificação no sub-espaço definido por este nó aplicando um CC conforme definido a seguir. Dada a solução relaxada $\tilde{x}$ do nó corrente, as variáveis binárias desta solução se dividem em três grupos: o grupo das variáveis que foram fixadas pelas ramificações do *branch-and-cut*, o grupo das variáveis com valor 0 ou 1 mas que não foram fixadas durante a enumeração, que chamaremos de G, e o grupo das variáveis com valor fracionário. Agora, seja g o número de variáveis binárias em G. Defina F^g como sendo a face de W_n formada por todos vetores binários de x tais que $x_j = \tilde{x}_j$ para todo $\tilde{x}_j \in G$. Além disso, defina G^+ como sendo o subconjunto de G onde todos $\tilde{x}_j = 1$ e G^- como sendo o subconjunto onde todos $\tilde{x}_j = 0$. O corte canônico que elimina todos vetores binários que estão à distância $\leq d$ de F^g é dado por

$$\sum_{x_i \in G^+} x_i \; - \sum_{x_i \in G^-} x_i \leq |G^+| - d \text{ que denotaremos por } \; CC(x, \tilde{x}) \geq d \qquad (4.1)$$

Tomando-se $d = 1$, dada uma solução relaxada $\tilde{x_j}$, o espaço de soluções associado com o nó corrente é particionado por uma disjunção da forma $CC(x, \tilde{x}) \leq 0$ (ramo esquerdo) e $CC(x, \tilde{x}) \geq 1$ (ramo direito). Note que $CC(x, \tilde{x})$ é não negativo e, portanto, o ramo

esquerdo corresponde às soluções do modelo que estão sobre a face F^g. Evidentemente, o ramo direito corresponderá às soluções que estão fora desta face. O intuito desta estratégia de ramificação é permitir que soluções viáveis sejam encontradas rapidamente em nós a pouca distância do nó raiz da enumeração. Percebe-se que um único CC correspondente a uma ramificação nesta estratégia equivale a várias ramificações padrão em que uma variável binária é fixada no seu valor máximo ou mínimo. Por isso, associamos esta ramificação com CCs à idéia de um "mergulho" rápido na enumeração.

4.1 Implementação

Como queremos controlar a quantidade de "mergulhos" na árvore de *branch-and-cut*, iremos adicionar ao método um parâmetro f que indicará a freqüência de aplicação do DB, ou seja, o número de nós explorados entre duas aplicações sucessivas da ramificação DB. Decidimos também aplicar CCs associados a faces de dimensão não muito elevadas, restringido o número máximo de variáveis que são fixadas no grupo G. Isto é feito por meio do parâmetro m.

Esta última restrição nos leva a um questionamento: quais variáveis de G devem ser fixadas? Para responder a esta pergunta tentamos utilizar os custos reduzidos das variáveis não básicas do grupo G. A idéia era fixar as m variáveis não básicas cujos valores absolutos dos custos reduzidos fossem os maiores. Contudo, experimentos computacionais que realizamos não permitiram comprovar que esta escolha é melhor do que uma simples escolha aleatória. Assim, decidimos fixar sempre as m primeiras variáveis.

A descrição da ramificação DB em um nó de enumeração é dada a seguir, onde supõe-se que $\tilde{x}$ é a solução da relaxação linear corrente.

1. Aplicamos o corte $CC(x, \tilde{x}) \leq 0$ do lado esquerdo. Para evitar sobrecarregar a formulação acrescentando restrições, fixamos as primeiras m variáveis de G alterando seus limites inferiores ou superiores de modo equivalente ao que seria feito pelo corte.

2. Aplicamos o corte $CC(x, \tilde{x}) \geq 1$ do lado direito, determinando um sub-espaço onde foi eliminado a face canônica criada no lado esquerdo.

Caso o nó onde será aplicado o DB possua o grupo G vazio, aplicaremos a ramificação padrão sobre variáveis binárias, ou seja, escolhendo-se a variável com a parte fracionária mais próxima de 0.5.

Uma escolha importante refere-se a qual estratégia adotar para explorar a árvore de enumeração. As opções usuais são *best bound*, busca em profundidade ou busca em largura. O resolvedor que estamos utilizando, o `XPRESS`, nos permite selecionar todas estas opções. Mas como ele possui algumas rotinas para selecionar automaticamente uma

"melhor" estratégia baseado nas características da matriz, e também, como não tinhamos muito mais tempo para realizarmos testes com diferentes estratégias, decidimos optar pela seleção automática fornecida pelo XPRESS.

4.2 Experimentos e resultados

Para avaliarmos a capacidade do DB em melhorar os resultados de uma ramificação padrão, utilizamos o resolvedor comercial XPRESS, versão 16.10.3, em uma máquina *Pentium IV* com 3.2 GHz, 2 Gigabytes de RAM e rodando no sistema *Linux*. O *benchmark* utilizado nos testes é o mesmo da seção 2.4 e o tempo máximo de execução foi limitado em uma hora.

Diferentemente das seções 2 e 3, e por estarmos na fase final da dissertação, não pudemos realizar testes intensivos para calibrar a estratégia. Para o nosso primeiro teste, utilizamos para f, a freqüência de aplicação do DB, os valores 100 e 50 e fixamos o valor de m, o número de variáveis que serão fixadas de G, em 10. Estes testes foram executados sobre o conjunto de todas as instâncias do *benchmark*.

Para definirmos qual é o melhor valor para um dado parâmetro, utilizamos também a média geométrica da razão computada em cada intervalo de 100 segundos. A diferença desta avaliação em relação àquelas das seções 2.4 e 3.3 é que não iremos separar as instâncias nos grupos de propagação pequena e média.

Os resultados dos testes para a escolha dos parâmetros f e m estão ilustrados nos gráficos das figuras 4.1 e 4.2. Podemos ver na figura 4.1 que o DB com $f = 100$ obteve o melhor desempenho. Com o parâmetro f fixado neste valor, criamos duas variantes da ramificação DB, que chamaremos de DB1 e DB2. Em DB1 faz-se $m = 10$, enquanto em DB2 faz-se $m = 20$. O gráfico da média geométrica das razões pode ser visto na figura 4.2. Como podemos perceber, o DB2 obteve um desempenho melhor do que o DB1 e que o *default* do XPRESS.

Os melhores valores calculados durante esta bateria de testes e o(s) método(s) que a obteve(obtiveram) podem ser vistos na tabela 4.1. A tabela também contém as informações das instâncias, onde n indica o número de colunas, b o número de variáveis binárias e m o número de linhas. Nota-se que o resolvedor com a estratégia DB2 alcançou a melhor solução em 14 das 25 instâncias contra 12 do DB1 e 11 do *default* do XPRESS.

Também realizamos a comparação dos métodos aos pares, como feito em seções anteriores. As estatísticas levantadas para esta análise são exibidas na tabela 4.2.

Analisando os dados da tabela, podemos ver que os métodos DB1 e DB2 conseguem obter uma maior quantidade de melhores respostas do que o *default* do XPRESS nos primeiros 10 minutos. Vimos também que o DB2 obteve os melhores resultados nos últimos 20 minutos, além de ter sido sempre superior ao DB1.

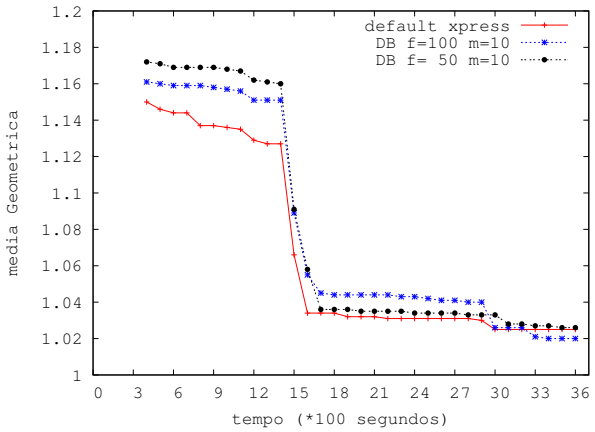

Figura 4.1: DB variando o f

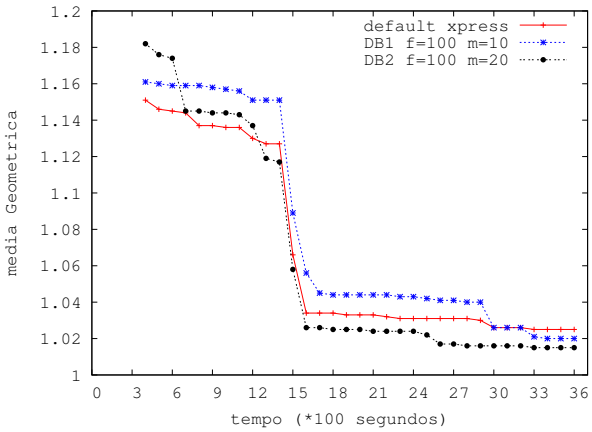

Figura 4.2: DB variando o m

O número de nós e CCs aplicados em cada instância são mostrados na tabela 4.3. Verificando esta tabela, podemos ver que em algumas instâncias o número de CCs aplicados é bem pequeno. Isto ocorre por duas razões. A primeira acontece em instâncias onde na maioria das vezes o grupo G, nos nós da árvore de *branch-and-cut* onde seriam aplicados os CCs, é vazio. Como discutimos anteriormente, neste caso, aplicamos a ramificação padrão e não o corte canônico. A outra razão é devida a escolha dos parâmetros f e m. Como utilizamos os mesmos parâmetros para todas as instâncias, em algumas situações o número de nós explorados é bem baixo, devido à dificuldade intrínseca dos modelos a serem resolvidos. Com isto, devido à escolha do valor de f, são poucas as oportunidades da aplicação dos CCs. Uma maneira de gerarmos mais CCs neste tipo de instância seria adotarmos parâmetros diferentes em cada caso baseando-nos em algum conhecimento prévio sobre a instância de entrada. Contudo, não foi possível investigar esta questão neste trabalho.

Instancias	n	b	m	Melhor Valor	Best Bound	Método
A1C1S1	3648	192	3312	11839.98	8245.80	DB1
A2C1S1	3648	192	3312	11492.26	7023.22	DB2
B1C1S1	3872	288	3904	25659.00	14942.92	DB1
B2C1S1	3872	288	3904	27691.88	13301.05	DB2
biella1	7328	6110	1203	3065005.78	3065005.78	XPRESS, DB1, DB2
danoint	521	56	664	65.67	63.59	XPRESS, DB1, DB2
glass4	322	302	396	1700016275.00	900006464.00	DB1
markshare1	62	50	7	5.00	0.00	DB1
markshare2	74	60	8	13.00	0.00	XPRESS
mkc	5325	5323	3411	-559.09	-564.77	DB2
net12	14115	1603	14021	214.00	148.31	XPRESS, DB1, DB2
nsrand-ipx	6621	6620	735	51360.00	50719.97	DB2
rail2536c	15293	15284	2539	689.00	689.00	XPRESS, DB1, DB2
rail2586c	13226	13215	2589	967.00	936.18	DB2
rail4284c	21714	21705	4287	1264.00	1054.24	DB2
rail4872c	24656	24645	4875	1789.00	1510.04	DB1
rail507	63019	63009	509	174.00	172.81	XPRESS, DB1, DB2
seymour	1372	1372	4944	424.00	411.39	DB2
sp97ar	14101	14101	1761	665772913.92	654912128.00	XPRESS
sp97ic	12497	12497	1033	429999311.04	424640928.00	DB2
sp98ar	15085	15085	1435	532763180.64	527032512.00	DB1
sp98ic	10894	10894	825	450280257.28	447947136.00	XPRESS
swath	6805	6724	884	467.41	424.18	XPRESS
tr12-30	1080	360	750	134789.00	75683.81	XPRESS
van	12481	192	27331	6.13	3.72	XPRESS, DB1, DB2

Tabela 4.1: Informações das instâncias do *benchmark*

	10 minutos				20 minutos			
Comparações	Xpress	DB1	DB2	EMPATE	Xpress	DB1	DB2	EMPATE
Xpress x DB1	5	11	-	9	12	6	-	7
Xpress x DB2	7	-	9	9	9	-	9	7
DB1 x DB2	-	7	8	10	-	6	10	9

	30 minutos				40 minutos			
Comparações	Xpress	DB1	DB2	EMPATE	Xpress	DB1	DB2	EMPATE
Xpress x DB1	11	6	-	8	11	6	-	8
Xpress x DB2	10	-	8	7	9	-	9	7
DB1 x DB2	-	5	13	7	-	5	13	7

	50 minutos				60 minutos			
Comparações	Xpress	DB1	DB2	EMPATE	Xpress	DB1	DB2	EMPATE
Xpress x DB1	10	7	-	8	8	8	-	9
Xpress x DB2	8	-	10	7	7	-	12	6
DB1 x DB2	-	8	10	7	-	8	11	6

Tabela 4.2: Comparação dos métodos aos pares

	DB1		DB2	
Instância	Nós	CC	Nós	CC
A1C1S1	637913	6169	638508	6179
A2C1S1	662159	6250	664393	6194
B1C1S1	264730	2272	260747	2343
B2C1S1	151048	1323	149409	1309
biella1	4908	23	3806	21
danoint	168452	1045	168150	1042
glass4	3445096	28984	3750540	31513
markshare1	11175980	69666	11080452	65960
markshare2	9977735	78602	9596333	73049
mkc	559998	4187	612615	4602
net12	3035	23	3344	29
nsrand_ipx	136187	1115	138218	1052
rail2536c	1164	5	1137	4
rail2586c	619	6	646	6
rail4284c	146	1	145	1
rail4872c	216	2	209	2
rail507	1442	11	1405	8
seymour	17431	168	17389	158
sp97ar	13415	107	15814	125
sp97ic	54449	418	60293	405
sp98ar	24290	195	24020	197
sp98ic	51860	377	61076	454
swath	426357	3021	375002	2653
tr12-30	2296337	22739	2312181	22883
van	78	0	78	0

Tabela 4.3: Estatísticas dos métodos

Capítulo 5

Comparação entre todos os métodos

Neste capítulo, comparamos diversos métodos que implementamos neste trabalho. Esta análise incluirá o método de RamLoc proposto por Fischetti e Lodi (`LB`), a nossa melhor variação do RamLoc com o uso de CCs (`NC`), a `RINS`, a `RINSCC`, a melhor heurística de *dive branching* (`DB2`) e o *default* do `XPRESS`.

Instancias	`n`	`b`	`m`	Melhor Valor	Best Bound	Método
A1C1S1	3648	192	3312	11812.13	8245.80	RINSCC
A2C1S1	3648	192	3312	11090.47	7153.25	NC
B1C1S1	3872	288	3904	24916.91	14942.92	RINSCC
B2C1S1	3872	288	3904	26583.46	13318.65	RINS
biella1	7328	6110	1203	3065005.78	3065005.78	XPRESS, RINS, DB2
danoint	521	56	664	65.67	63.59	XPRESS, LB, NC, RINSCC, DB2
glass4	322	302	396	1640014740.00	900004928.00	RINS
markshare1	62	50	7	4.00	0.00	LB
markshare2	74	60	8	13.00	0.00	XPRESS, RINSCC
mkc	5325	5323	3411	-563.01	-564.77	RINS
net12	14115	1603	14021	214.00	148.31	XPRESS, LB, NC, RINS, RINSCC, DB2
nsrand-ipx	6621	6620	735	51360.00	50719.97	DB2
rail2536c	15293	15284	2539	689.00	689.00	XPRESS, DB2
rail2586c	13226	13215	2589	960.00	936.18	RINS, RINSCC
rail4284c	21714	21705	4287	1083.00	1054.24	RINS, RINSCC
rail4872c	24656	24645	4875	1564.00	1510.04	RINS, RINSCC
rail507	63019	63009	509	174.00	172.81	XPRESS, LB, NC, RINS, DB2
seymour	1372	1372	4944	424.00	411.37	LB, RINS, RINSCC, DB2
sp97ar	14101	14101	1761	665221144.64	654887168.00	NC
sp97ic	12497	12497	1033	429999311.04	424640928.00	DB2
sp98ar	15085	15085	1435	530566673.44	526999520.00	RINS
sp98ic	10894	10894	825	450067753.44	447658272.00	NC
swath	6805	6724	884	467.41	424.10	XPRESS
tr12-30	1080	360	750	130640.00	101263.63	RINS
van	12481	192	27331	5.09	3.72	LB, NC, RINS, RINSCC

Tabela 5.1: Informações das instâncias do *benchmark*

Se nos restringimos aos dados da tabela 5.1, referentes aos resultados alcançados ao término de 1 hora de computação, observamos que os métodos tiveram desempenhos

relativamente próximos. De fato, a RINS foi aquele que mais vezes chegou à melhor solução, 13, porém o LB que teve o pior desempenho segundo esta estatística atingiu 6 vezes a melhor solução. Os métodos NC e XPRESS atingiram 7 melhores soluções cada, o DB2 8 melhores soluções e o RINSCC 10 melhores soluções. De qualquer modo, vale notar que só o LB foi inferior ao *default* do XPRESS nesta análise.

Consideremos agora a média geométrica das razões da mesma maneira que nas seções anteriores. Novamente dividimos os resultados das instâncias em dois grupos diferentes. Lembrando, o grupo de "propagação pequena" corresponde às instâncias onde o *gap* da razão entre a pior e a melhor solução final obtida entre os 6 métodos é menor do que 10%. Já no grupo de "propagação média", este *gap* é maior do que 10%. As informações das razões das instâncias, após uma hora de computação, podem ser consultadas na tabela 5.2, onde é possível observar qual foi o desempenho final de cada método em relação à melhor solução obtida nas 25 instâncias e quais instâncias pertencem ao grupo de propagação pequena e média.

Instâncias	XPRESS	LB	NC	RINS	RINSCC	DB2
Propagação pequena						
A1C1S1	1.015	1.018	1.005	1.001	**1.000**	1.009
A2C1S1	1.064	1.026	**1.000**	1.007	**1.000**	1.036
B1C1S1	1.038	1.053	1.046	1.016	**1.000**	1.048
biella1	**1.000**	1.003	**1.000**	**1.000**	1.008	**1.000**
danoint	**1.000**	**1.000**	**1.000**	1.013	**1.000**	**1.000**
mkc	1.011	1.013	1.013	**1.000**	1.010	1.007
net12	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
nsrand_ipx	1.003	1.034	1.025	1.016	1.050	**1.000**
rail2536c	**1.000**	1.001	1.001	1.001	1.003	**1.000**
rail2586c	1.010	1.009	1.008	**1.000**	**1.000**	1.007
rail507	**1.000**	**1.000**	**1.000**	**1.000**	1.006	**1.000**
seymour	1.002	**1.000**	1.002	**1.000**	**1.000**	**1.000**
sp97ar	1.001	1.009	**1.000**	1.004	1.006	1.004
sp97ic	1.003	1.021	1.001	1.002	1.020	**1.000**
sp98ar	1.005	1.006	1.001	**1.000**	1.005	1.005
sp98ic	**1.000**	1.010	**1.000**	**1.000**	1.015	1.003
tr12-30	1.032	1.006	1.002	**1.000**	**1.000**	1.032
Propagação média						
B2C1S1	1.046	1.152	1.055	**1.000**	1.008	1.042
glass4	1.118	1.067	1.006	**1.000**	1.211	1.098
markshare1	2.000	**1.000**	1.750	4.000	2.000	1.500
markshare2	**1.000**	1.615	1.615	1.615	**1.000**	1.077
rail4284c	1.168	1.007	1.011	**1.000**	**1.000**	1.167
rail4872c	1.156	1.005	1.013	**1.000**	**1.000**	1.152
swath	**1.000**	1.046	1.022	1.008	1.105	1.020
van	1.205	**1.000**	**1.000**	**1.000**	**1.000**	1.205

Tabela 5.2: Razão, após uma hora, entre a solução do método e a melhor encontrada

Analisando a tabela 5.2, onde a razão é representada com apenas 3 dígitos de precisão, notamos que o número de melhores respostas em relação à tabela 5.1 varia. Isto ocorre

pois em algumas instâncias como a `biella1` o valor ótimo encontrado pelo `XPRESS`, `RINS` e `DV` foi 3065005.78 enquanto o `NC` encontrou 3065065.13. Mas se analisarmos a razão com apenas 3 dígitos de precisão, o método `NC` também terá na tabela 5.2 o valor 1.000 pois, a sua razão é aproximadamente 1.000019.

Os gráficos das médias geométricas computadas a cada 100 segundos de computação para as instâncias de propagação pequena e de propagação média podem ser vistos nas figuras 5.1 e 5.2 respectivamente. Como podemos ver, em relação ao grupo de propagação

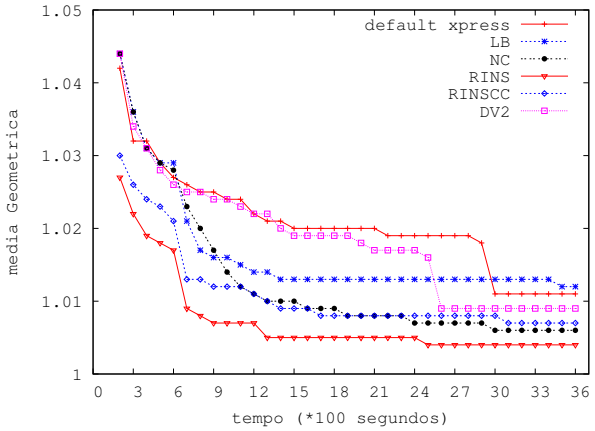

Figura 5.1: Execução das 17 instâncias do grupo de "propagação pequena"

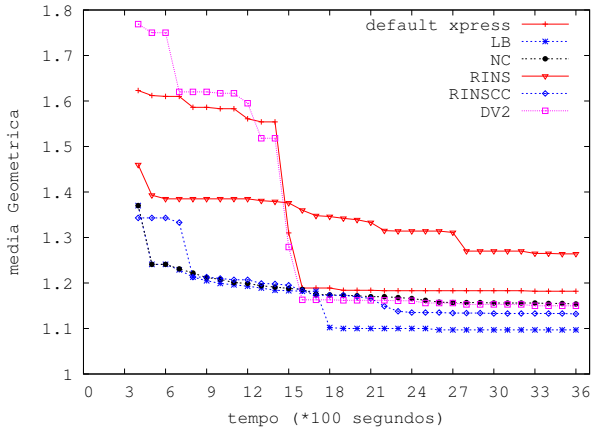

Figura 5.2: Execução das 8 instâncias do grupo de "propagação média"

pequena, o `RINS` obteve o melhor resultado. Podemos também concluir, que todos os métodos avaliados nesta seção melhoraram o desempenho das instâncias de propagação pequena em relação ao *default* do `XPRESS`. Porém, analisando o grupo de propagação média, notamos que o `LB` obteve o melhor desempenho. podemos ver ainda que a `RINS` foi o único método que não obteve desempenho melhor do que o *default* do `XPRESS` neste grupo.

Finalmente, realizamos foi a comparação dos métodos aos pares, onde avaliamos a evolução da computação em intervalos de tempo de 10 minutos. Como de hábito, fixaremos que o método que obtiver o maior número de melhores soluções será considerado o melhor. A tabela 5.3, exibe na coluna `comparações` os nomes dos métodos que estão sendo comparados, além do número de melhores soluções que cada método obteve e o número de empates (`E`). Nota-se que, em todas as comparações, a RINS obteve o melhor desempenho em todos os intervalos.

A idéia original deste trabalho era investigar a inclusão de cortes canônicos no método de Ramificação Local de Fischetti e Lodi [6]. Se nos concentramos apenas na comparação de `LB` e `NC` na tabela 5.3, veremos que, neste sentido, o trabalho foi bem sucedido já que o `LB` sempre é superado pelo `NC` (exceto, é claro, nos 10 minutos iniciais onde o

XPRESS *default* roda sozinho, conforme explicado no capítulo 2).

Outra forma de aplicação dos CC que exploramos foi o seu uso como parte do processo global de ramificação do resolvedor MIP. Restringindo-nos às comparações XPRESS x DB2 na tabela 5.3, percebemos que esta estratégia mostrou-se particularmente interessante no início e no final do processamento, sendo os dois métodos praticamente equivalentes nos instantes intermediários.

Por outro lado, a obtenção de heurísticas para modelos gerais de programação inteira mista tornou-se um tópico de pesquisa bastante ativo durante o período de desenvolvimento desta dissertação. Em decorrência disso, novas técnicas heurísticas surgiram num pouco espaço de tempo. Em particular, destacamos a RINS que, de tão bem sucedida, foi imediatamente incorporada ao resolvedor CPLEX, tido como uma das melhores ferramentas comerciais de MIP.

Conseguimos implementar a RINS neste trabalho e ainda encontramos um modo de incorporar CCs a esta heurística. Mesmo sem termos realizado muitos testes de ajuste de parâmetros da RINSCC, notamos na tabela 5.3 que a inclusão de CCs na RINS, se não melhorou esta última, pelo menos não deteriorou seu desempenho. Contudo, uma melhor comparação da RINS e da RINSCC pode ser feita nos gráficos das figuras 5.1 e 5.2. Embora a RINS supere a RINSCC no grupo de propagação pequena, nestas instâncias a RINSCC está freqüentemente entre os 3 melhores métodos disponíveis e supera o *default* do XPRESS. Além disso, para o grupo de propagação média, a RINSCC só é superada por LB, enquanto a RINS foi o método que apresentou o pior desempenho entre todos aqueles testados perdendo inclusive para o XPRESS. Isto sugere que a inclusão dos CCs na RINS precisa ser melhor investigada.

A experiência que adquirimos através dos intensivos testes computacionais que fizemos, mostra que todos os métodos são muito sensíveis ao ajuste de parâmetros. Também não é possível estabelecer uma clara dominação de um método sobre um outro. Contudo, como evidenciamos pelas discussões acima, conseguimos mostrar a utilidade dos CCs para a obtenção de soluções viáveis de boa qualidade em um tempo razoável de computação.

	10 minutos							20 minutos						
Comparações	Xpress	LB	NC	RINS	RINSCC	DB2	E	Xpress	LB	NC	RINS	RINSCC	DB2	E
Xpress x LB	10	12	-	-	-	-	3	11	11	-	-	-	-	3
Xpress x NC	10	-	12	-	-	-	3	5	-	16	-	-	-	4
Xpress x RINS	10	-	-	13	-	-	2	9	-	-	13	-	-	3
Xpress x RINSCC	10	-	-	-	12	-	3	12	-	-	-	10	-	3
Xpress x DB2	7	-	-	-	-	9	9	9	-	-	-	-	9	7
LB x NC	-	0	1	-	-	-	24	-	4	12	-	-	-	9
LB x RINS	-	4	-	15	-	-	6	-	6	-	14	-	-	5
LB x RINSCC	-	9	-	-	11	-	5	-	12	-	-	9	-	4
LB x DB2	-	12	-	-	-	10	3	-	11	-	-	-	10	4
NC x RINS	-	-	4	15	-	-	6	-	-	8	13	-	-	4
NC x RINSCC	-	-	9	-	11	-	5	-	-	13	-	9	-	3
NC x DB2	-	-	12	-	-	10	3	-	-	14	-	-	7	4
RINS x RINSCC	-	-	-	9	9	-	7	-	-	-	11	8	-	6
RINS x DB2	-	-	-	14	-	9	2	-	-	-	13	-	9	3
RINSCC x DB2	-	-	-	-	11	11	3	-	-	-	-	11	12	2

	30 minutos							40 minutos						
Comparações	Xpress	LB	NC	RINS	RINSCC	DB2	E	Xpress	LB	NC	RINS	RINSCC	DB2	E
Xpress x LB	13	9	-	-	-	-	3	13	9	-	-	-	-	3
Xpress x NC	8	-	13	-	-	-	4	9	-	12	-	-	-	4
Xpress x RINS	9	-	-	14	-	-	2	8	-	-	14	-	-	3
Xpress x RINSCC	11	-	-	-	10	-	4	10	-	-	-	11	-	4
Xpress x DB2	10	-	-	-	-	8	7	9	-	-	-	-	9	7
LB x NC	-	5	13	-	-	-	7	-	4	13	-	-	-	8
LB x RINS	-	5	-	15	-	-	5	-	4	-	16	-	-	5
LB x RINSCC	-	10	-	-	11	-	4	-	10	-	-	11	-	4
LB x DB2	-	9	-	-	-	12	4	-	8	-	-	-	13	4
NC x RINS	-	-	9	12	-	-	4	-	-	8	13	-	-	4
NC x RINSCC	-	-	12	-	10	-	3	-	-	12	-	10	-	3
NC x DB2	-	-	14	-	-	8	3	-	-	13	-	-	9	3
RINS x RINSCC	-	-	-	11	8	-	6	-	-	-	11	8	-	6
RINS x DB2	-	-	-	13	-	9	3	-	-	-	14	-	8	3
RINSCC x DB2	-	-	-	-	10	12	3	-	-	-	-	11	11	3

	50 minutos							60 minutos						
Comparações	Xpress	LB	NC	RINS	RINSCC	DB2	E	Xpress	LB	NC	RINS	RINSCC	DB2	E
Xpress x LB	14	8	-	-	-	-	3	13	9	-	-	-	-	3
Xpress x NC	9	-	12	-	-	-	4	8	-	13	-	-	-	4
Xpress x RINS	7	-	-	14	-	-	4	7	-	-	14	-	-	4
Xpress x RINSCC	10	-	-	-	11	-	4	10	-	-	-	11	-	4
Xpress x DB2	8	-	-	-	-	10	7	7	-	-	-	-	12	6
LB x NC	-	4	14	-	-	-	7	-	4	14	-	-	-	7
LB x RINS	-	3	-	16	-	-	6	-	2	-	17	-	-	6
LB x RINSCC	-	9	-	-	12	-	4	-	8	-	-	13	-	4
LB x DB2	-	7	-	-	-	14	4	-	7	-	-	-	14	4
NC x RINS	-	-	5	15	-	-	5	-	-	6	14	-	-	5
NC x RINSCC	-	-	12	-	10	-	3	-	-	12	-	10	-	3
NC x DB2	-	-	11	-	-	11	3	-	-	11	-	-	11	3
RINS x RINSCC	-	-	-	13	6	-	6	-	-	-	13	6	-	6
RINS x DB2	-	-	-	15	-	6	4	-	-	-	15	-	6	4
RINSCC x DB2	-	-	-	-	10	12	3	-	-	-	-	10	12	3

Tabela 5.3: Comparação dos métodos aos pares

Capítulo 6

Conclusões e Perspectivas Futuras

Nos últimos anos, temos visto alguns grupos ligados à área de MIP diversificarem a sua pesquisa, antes quase que totalmente centrada na prova de otimalidade, para tentar obter soluções viáveis para MIPs gerais a um baixo tempo computacional. Esta mudança foi provocada pela necessidade dos usuários de MIP nas empresas, onde percebeu-se que a busca pela otimalidade nem sempre se justifica, especialmente em cenários reais em que, usualmente, há incerteza nos valores dos dados de entrada.

Para tentar atender a esta demanda, vimos surgirem várias heurísticas e meta-heurísticas de propósito geral para MIPs (cf., [6], [3], [1], [13], [7], [8], [11]). Nesta dissertação direcionamos os nossos esforços em tentar melhorar o desempenho de duas delas, a RamLoc [6] e a RINS [3], através da aplicação de cortes canônicos introduzidos por Balas e Jeroslow [2].

Efetuamos intensos experimentos computacionais e conseguimos mostrar que a aplicação de CCs pode melhorar o desempenho do método de RamLoc. Em testes com instâncias de *benchmarks* de domínio público, observamos que em 43% das faces canônicas exploradas, conseguimos encontrar uma solução melhor do que a computada na vizinhança da Ramificação Local que a gerou. Além disso, 11% destas soluções eram as melhores encontradas até aquele momento. Assim, concluímos que a aplicação dos CCs gera uma intensificação na busca da vizinhança gerada pela RamLoc, possibilitando o resolvedor focar-se em regiões mais promissoras do espaço de soluções, melhorando o desempenho global do método proposto por Fischetti e Lodi.

Em relação à aplicação dos CCs na heurística RINS, os resultados foram interessantes porém os CCs não foram tão eficazes quanto no caso anterior, pelo menos nas instâncias chamadas de "propagação pequena". Como pudemos ver na tabela 3.4, o número de sub-problemas reduziu muito em relação à RINS sem os cortes canônicos. Isto ocorreu devido à restrição que impusemos à heurística de somente fixar as variáveis binárias. Esta decissão aumentou o tamanho das vizinhanças a serem exploradas, diminuindo as

chances de obter boas soluções dentro do limite de nós fixados para a resolução do sub-MIP correspondente. Portanto, uma possível melhora no resultado do RINSCC poderia ter sido alcançada caso tivéssemos calibrado o código para que aumentasse a freqüência com que os sub-problemas são resolvidos, ao invés de utilizarmos os parâmetros que foram calibrados para o RINS. Mesmo assim, apesar de não ter conseguido um resultado geral melhor do que o RINS, vimos que em algumas instâncias o RINSCC melhorou bastante o resultado da heurística original (ver tabela 3.2). Nestas instâncias denominadas de propagação média, a heurística RINS original é até pior do que o *default* do `XPRESS`. Isso sugere que há margem para melhorias na inclusão dos CCs no RINS.

Além de tentar melhorar o desempenho da RamLoc e da RINS, nesta dissertação apresentamos um novo método que utiliza os CCs em uma estratégia de ramificação, que denominamos de *Dive Branching*. Apesar dos poucos experimentos que conseguimos fazer, mostramos que este procedimento melhorou o desempenho do *default* do `XPRESS` em várias instâncias de domínio público. Acreditamos, portanto, que está idéia é promissora e deve ser explorada futuramente.

Mesmo com os bons resultados alcançados nesta dissertação, vislumbramos algumas possibilidades de melhoria no desempenho dos métodos de RamLoc, RINS e DB. Nos três casos existem vários parâmetros a serem calibrados, o que pode ser um problema para um usuário final. Um possível trabalho futuro seria tentar calibrar estes parâmetros de maneira automática, utilizando as informações da instância que será resolvida tais como estatísticas sobre o percentual de variáveis binárias, densidade das restrições do modelo, tempo de resolução da relaxação linear, etc.

Não houve tempo para testar mais intensamente a aplicação dos CCs na RINS e no DB. Achamos que um estudo mais aprofundado sobre a calibragem dos parâmetros de execução pode melhorar ainda mais o desempenho destes dois métodos.

Finalmente, não devemos excluir a possibilidade de combinação das demais heurísticas com a estratégia de *dive branching*. Isso faz sentido especialmente se recordarmos que em várias destas heurísticas que usam os CCs, o *default* do `XPRESS` roda sozinho nos primeiros 10 minutos de execução. Mas, pela tabela 5.3, vê-se que o DB2 foi mais efetivo que o `XPRESS`, exatamente neste período de execução.

Referências Bibliográficas

[1] E. Balas, S. Ceria, M. Dawande, F. Margot, e G.Pataki. Octane: A new heuristic for pure 0–1 programs. *Operations Research*, 49 (2):207–225, 2001.

[2] E. Balas e R. Jeroslow. Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23 (1):61–69, 1972.

[3] Emilie Danna, Edward Rothberg, e Claude Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102:71–90, 2005.

[4] Cid Carvalho de Souza, Elder M. Macambira, e Nelson Maculan. A note on characterizing canonical cuts using geometry. *International Transactions in Operational Research*, 12:581–593, 2006.

[5] Matteo Fischetti, Fred Glover, e Andrea Lodi. The feasibility pump. *Mathematical Programming*, 104:91–104, 2005.

[6] Matteo Fischetti e Andrea Lodi. Local branching. *Mathematical Programming, Series B*, 98:23–47, 2003.

[7] Fred Glover e Manuel Laguna. General purpose heuristics for integer programming part i. *Journal of Heuristics*, 2:343–358, 1997.

[8] Fred Glover e Manuel Laguna. General purpose heuristics for integer programming part ii. *Journal of Heuristics*, 3(2):161–179, 1997.

[9] Marco Cesar Goldbarg e Henrique Pacca L. Luna. *Otimização Combinatória e Programação Linear: Modelos e Algoritmos*. Elsevier, 2000.

[10] Pierre Hansen, Nenad Mladenović, e Dragan Urošević. Variable neighborhood search and local branching. *Computers & Operations Research*, 33:3034–3045, 2006.

[11] A. Løkketangen e Fred Glover. Solving zero/one mixed integer programming problems using tabu search. *European Journal of Operation Research*, 106:624–658, 1998.

[12] N. Mladenović e P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

[13] M. Nediak e J. Eckstein. Pivot, cut and dive: A heuristic for 0–1 mixed integer programming. Technical report, Rutgers Center for Operations Research, RRR 53-2001, 2001.

[14] G. L. Nemhauser e L. Wolsey. *Integer and Combinatorial Optimization*. Wiley & Sons, 1988.

[15] Laurence A. Wolsey. *Integer Programming*. Wiley & Sons, 1998.