UNIVERSIDADE FEDERAL DE OURO PRETO

Um algoritmo híbrido para resolução de problemas binários

Josiane da Costa Vieira Rezende
Universidade Federal de Ouro Preto

Orientador: Marcone Jamilson Freitas Souza

Coorientador: Rone Ilídio da Silva

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto, como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação.

Ouro Preto, 04 de Setembro de 2015

Um algoritmo híbrido para resolução de problemas binários

Josiane da Costa Vieira Rezende
Universidade Federal de Ouro Preto

Orientador: Marcone Jamilson Freitas Souza

Coorientador: Rone Ilídio da Silva

R467a Rezende, Josiane da Costa Vieira.
Um algoritmo híbrido para resolução de problemas binários [manuscrito] / Josiane da Costa Vieira Rezende. - 2015.
70f.: il.: tabs.

Orientador: Prof. Dr. Marcone Jamilson Freitas Souza.

Dissertação (Mestrado) - Universidade Federal de Ouro Preto. Instituto de Ciências Extas e Biológicas. Computação. Ciência da Computação.

1. Programação linear. 2. Heurística. 3. Processo estocástico. I. Souza, Marcone Jamilson Freitas. II. Universidade Federal de Ouro Preto. III. Titulo.

CDU: 004.94:519.21

Catalogação: www.sisbin.ufop.br

Ata da Defesa Pública de Dissertação de Mestrado

Aos 04 dias do mês de setembro de 2015, às 15 horas na Sala de Seminários do DECOM no Instituto de Ciências Exatas e Biológicas (ICEB), reuniram-se os membros da banca examinadora composta pelos professores: **Prof. Dr. Marcone Jamilson Freitas Souza (presidente e orientador), Prof. Dr. Alexandre Xavier Martins e Prof. Dr. Sérgio Ricardo de Souza**, aprovada pelo Colegiado do Programa de Pós-Graduação em Ciência da Computação, a fim de arguirem a mestranda **Josiane da Costa Vieira Rezende**, com o título **"Um algoritmo híbrido para resolução de problemas binários"**. Aberta a sessão pelo presidente, coube ao candidato, na forma regimental, expor o tema de sua dissertação, dentro do tempo regulamentar, sendo em seguida questionado pelos membros da banca examinadora, tendo dado as explicações que foram necessárias.

Recomendações da Banca:

(X) Aprovada sem recomendações

( ) Reprovada

( ) Aprovada com recomendações: _____

Banca Examinadora:

Prof. Dr. Marcone Jamilson Freitas Souza

Prof. Dr. Alexandre Xavier Martins

Prof. Dr. Sérgio Ricardo de Souza

Prof. Dr. Luiz Henrique de Campos Merschmann
Coordenador do Programa de Pós-Graduação em Ciência da Computação
DECOM/ICEB/UFOP

Ouro Preto, 04 de setembro de 2015.

Dedico este trabalho a Deus, ao meu marido Lourenço, meus pais Raimundo e Efigênia, meu irmão Lucas, minha avó Maria de Lourdes e meu filho que nascerá em outubro Lorenzo Gabriel.

Resumo

Este trabalho apresenta um método híbrido, denominado HGVPRLB, para resolver problemas lineares binários. O método combina os procedimentos *Greedy Randomized Adaptive Search Procedures* – GRASP, *Variable Neighborhood Descent* – VND, propagação de restrições, e cortes *Local branching*. Como em todo algoritmo GRASP, o método HGVPRLB apresenta duas fases, que interagem entre si até que o tempo limite seja atingido. A primeira fase visa a construção de uma solução inicial; a segunda, por sua vez, visa ao refinamento dessa solução construída. Na fase de construção, é utilizado o resolvedor *CBC* e um procedimento de propagação de restrições, de forma a gerar uma solução inicial para o problema. O resolvedor *CBC* relaxa as variáveis binárias, isto é, atribui o valor de cada variável no intervalo real [0,1]. O procedimento propagação de restrições possui a finalidade de verificar se existe solução viável ao se fixar uma determinada variável no valor 1. Se esta solução existir, ele poderá retornar, ainda, um conjunto de possíveis fixações das demais variáveis. Na fase de refinamento são utilizados cortes *Local branching* controlados pelo procedimento VND até que um tempo previamente definido seja atingido. Os cortes *Local Branching* utilizam um resolvedor de programação linear inteira como uma ferramenta caixa-preta para explorar eficientemente subespaços das soluções do problema. O método desenvolvido foi aplicado a um conjunto de problemas binários da biblioteca MIPLIB 2010 com o intuito de verificar sua capacidade de obter soluções viáveis de qualidade variando-se o tempo de processamento. Os experimentos computacionais realizados mostraram que, quando o tempo de processamento aumenta, o método consegue aumentar tanto o número de soluções viáveis quanto a qualidade delas. Além disso, o método desenvolvido se mostrou superior a outro método da literatura, bem como a dois outros resolvedores de código aberto nesses dois indicadores de avaliação.

Palavras-chave: Programação Linear Binária, Heurística, GRASP, *Variable Neighborhood Descent*, *Local Branching*, Propagação de Restrições.

Abstract

This paper presents a hybrid method, called HGVPRLB, in order to solve binary linear problems. The method combines the procedures Greedy Randomized Adaptive Search Procedures – GRASP, Variable Neighborhood Descent – VND, Constraint Propagation, and Local Branching cuts. As in all GRASP, HGVPRLB has two phases that interact with each other until the time limit is reached. The first phase aims to construct an initial solution and in the second, this constructed solution is refined. In the construction phase, *CBC* solver is used the and the constraint propagation procedure in order to obtain an initial solution. The *CBC* solver relaxes the binary variables, that is, assigns the value of each variable in the real interval [0,1]. The constraint propagation procedure has the purpose of verifying if there is feasible solution when a particular variable is set to 1. If this solution exist, it can still return a set of possible fixations to the other variables. In the refinement phase are used Local branching cuts controlled by a VND procedure until a predetermined time is reached. The Local Branching cuts uses an integer linear programming solver as a black box tool to efficiently explore solution subspaces of the problem. The proposed method was applied to a set of binary problems from MIPLIB 2010 library in order to verify its ability to get good quality feasible solutions varying the processing time. Computational experiments showed that when the processing time increases, the method can increase the number of feasible solutions as well as the quality of these solutions. Besides it, the proposed method outperforms another method of literature, as well as two other open source solvers in relation to these evaluation metrics.

Keywords: Linear Binary Programming, Heuristics, GRASP, *Variable Neighborhood Descent*, *Local Branching*, Constraint Propagation.

Declaração

Esta dissertação é resultado de meu próprio trabalho, exceto onde referência explícita é feita ao trabalho de outros, e não foi submetida para outra qualificação nesta nem em outra universidade.

Josiane da Costa Vieira Rezende

Agradecimentos

Primeiramente agradeço a Deus pela força e proteção durante esta caminhada.

Ao meu marido Lourenço, que mesmo nos momentos mais difíceis, foi minha grande fonte de incentivo e perseverança.

Ao meus pais Raimundo e Efigênia e minha avó Maria de Lourdes pelas orações nos momentos que mais precisei, sempre apoiando e acreditando no meu potencial.

Ao meu irmão Lucas, que sempre esteve presente em uma conversa de amizade e incentivo.

Ao meu filho Lorenzo Gabriel que irá nascer em outubro deste ano e nos últimos meses participa de todos os momentos na escrita desta dissertação, e que mesmo sem saber ainda, se tornou minha grande fonte de inspiração e dedicação.

A minha sogra Ângela, meu sogro Miguel, minhas cunhadas Jessica, Iara e Tatiana pelo apoio.

À todos os demais familiares pela confiança.

Aos amigos, que sempre estiveram dispostos a ajudar.

Aos professores do Programa de Pós-Graduação em Ciência da Computação da UFOP.

Ao meu coorientador Rone, que em momentos onde me encontrava perdida teve paciência e dedicação para ajudar, ensinar e apoiar.

Ao meu orientador Marcone, que me recebeu como orientanda no momento mais complicado do meu mestrado. Não obstante à sua ocupação de Reitor, esteve sempre disponível a me auxiliar com total dedicação, paciência e motivação.

Obrigada a todos, sem vocês este sonho não seria realidade.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Problemas de Programação Linear (PL) são problemas de otimização nos quais a função objetivo e as restrições são expressas por funções lineares. A função objetivo é uma função matemática que define a qualidade da solução em função das variáveis de decisão. As restrições definem quais as atividades que consomem recursos e em que proporções eles devem ser consumidos. Essas informações são apresentadas em forma de equações ou inequações lineares, uma para cada recurso. O conjunto dessas equações e/ou inequações, denomina-se "restrições do modelo". Em qualquer modelo de PL, as variáveis de decisão descrevem as decisões a serem feitas.

Problemas de Programação Linear Binária – PLB (Garfinkel e Nemhauser, 1972), por sua vez, podem ser entendidos como casos específicos de Programação Linear, em que todas as variáveis de decisão são binárias, ou seja, assumem valor 1 quando a característica de interesse está presente e 0, caso contrário.

Podem ser encontrados na literatura dois tipos de métodos para a solução de problemas de PLB: exatos e heurísticos. Os métodos exatos sempre encontram a melhor solução para o problema, quando ela existe, satisfazendo a todas as restrições impostas. Porém, para muitos problemas, tal tipo de método pode requerer um tempo computacional proibitivo para a tomada de decisão, ao ponto de tornar inviável a execução do algoritmo, se o problema for da classe NP-difícil. Por outro lado, o segundo tipo de método procura uma solução por meio de um procedimento empírico, sem a garantia de que ela seja ótima; entretanto, a solução pode ser encontrada em tempo viável para a tomada de decisão, e estar muito próxima da ótima.

Na literatura, encontram-se diversos softwares que utilizam uma dessas duas es-

tratégias para solucionar os problemas de programação linear. Tais softwares são chamados resolvedores. Eles podem ser tanto de código aberto quanto comerciais. Dentre eles, podem ser citados: CPLEX (IBM, 2015), COIN-OR-CBC (Foundation, 2015), Symphony (SYMPHONY, 2015), MINTO (MINTO, 2015), SCIP (SCIP, 2015), GLPK (GLPK, 2015), Lp Solve (lp solve, 2015), LocalSolver (Benoist et al., 2011) e *Local Branching* (Fischetti e Lodi, 2003).

A literatura também apresenta diversas estratégias para a solução de problemas de PLB. Dentre elas, destacam-se: Relaxamento LP (Geoffrion, 1974) e Programação por Restrições (Apt, 1999).

Relaxamentos LP consistem em relaxar as condições de integralidade das variáveis de decisão, isto é, os valores das variáveis passam a ser números reais compreendidos entre 0 e 1. Em programação por restrições é definido um espaço de busca em um conjunto de restrições e deseja-se encontrar soluções viáveis, isto é, pontos que pertencem ao espaço de busca e que satisfazem as restrições.

Existe uma grande necessidade de programas resolvedores de alta qualidade, e preferencialmente livres, de modo que o usuário possa modificá-los de acordo com suas necessidades a fim de obter melhores resultados de forma mais rápida.

Este trabalho vem ao encontro desse objetivo, ao propor um algoritmo híbrido para resolver problemas de PLB. O algoritmo proposto, nomeado HGVPRLB, é baseado na metaheurística *Greedy Randomized Adaptive Search Procedures* – GRASP (Feo e Resende, 1995). Em sua fase construtiva, ele mescla relaxamentos LP e programação por restrições para a obtenção de uma solução inicial. Em seguida, é realizada uma busca local, guiada por uma heurística *Variable Neighborhood Descent* – VND (Mladenović e Hansen, 1997), a qual, por sua vez, faz uso de cortes *Local Branching* (Fischetti e Lodi, 2003).

Experimentos computacionais realizados com problemas binários da biblioteca MIPLIB 2010 (Koch et al., 2011) mostram que o algoritmo proposto é capaz de produzir, em tempo reduzido, um número maior de soluções viáveis nessa biblioteca quando comparado com outro trabalho da literatura e com dois resolvedores de código aberto. Além disso, o algoritmo proposto apresenta soluções de qualidade superior aos mesmos com que foi comparado.

1.1 Objetivos

Nesta seção são apresentados os objetivos deste trabalho.

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver um algoritmo híbrido que seja capaz de resolver eficientemente problemas lineares binários.

1.1.2 Objetivos Específicos

Para alcançar o objetivo geral, se faz necessário a obtenção dos seguintes objetivos específicos:

- Analisar e estudar trabalhos da literatura que tratam o problema abordado e relacionados;
- Analisar e estudar técnicas de solução de problemas de otimização combinatória;
- Desenvolver um algoritmo híbrido, que combine procedimentos heurísticos com um resolvedor de programação linear inteira;
- Testar a eficiência do algoritmo desenvolvido a partir de experimentos computacionais realizados em problemas-teste da literatura;
- Divulgar os resultados obtidos em eventos científicos e periódicos da área.

1.2 Motivação

Diversas situações práticas podem ser modeladas como problemas de Programação Binária, como: Programação de horários - *timetabling* (de Werra, 1985), Alocação de Aulas a Salas - *Classroom Assignment Problem* (Carter e Tovey, 1992), *Bin Packing* (Weisstein, 2000) e Mochila Binária (Fréville, 2004), dentre muitos outros problemas existentes na literatura.

A resolução desta classe de problemas em um tempo aceitável para a tomada de decisão tem atraído a atenção dos pesquisadores como alternativa e complemento das

abordagens clássicas, uma vez que, mesmo com as sucessivas melhorias nos métodos exatos, tais como *branch-and-bound* e *branch-and-cut*, muitos desses problemas não são resolvidos em um tempo aceitável. Surge então a importância do desenvolvimento de algoritmos eficientes, combinando heurísticas com métodos exatos, para a solução do problema, de modo que mesmo que a solução ótima não seja encontrada, seja possível obter soluções viáveis de boa qualidade em um curto período de tempo.

1.3 Estrutura do Trabalho

O restante deste trabalho está estruturado como segue.

No Capítulo 2 é caracterizado o problema objeto deste trabalho. A revisão bibliográfica é apresentada no Capítulo 3. No Capítulo 4, o método proposto é apresentado em detalhes. Já no Capítulo 5 são apresentados os experimentos computacionais realizados com o método desenvolvido. O Capítulo 6 conclui esta dissertação e apresenta perspectivas de trabalhos futuros.

Capítulo 2

Caracterização do Problema

Problemas de Programação Linear Binária (PBL) podem ser entendidos como um caso particular de Problemas de Programação Linear Inteira (PLI), nos quais todas as variáveis de decisão assumem somente valores binários, isto é, 0 ou 1.

Um modelo de PLB é composto pela Função Objetivo (FO) e um conjunto de restrições do problema. A função objetivo é uma função matemática que determina o valor-alvo que se pretende alcançar ou a qualidade da solução em função das variáveis de decisão. A FO pode ser uma função de maximização ou minimização. As restrições podem ser definidas como um conjunto de equações ou inequações do qual as variáveis de decisão do modelo devem satisfazer. As restrições são adicionadas ao modelo de modo a satisfazer as limitações problema, e afetam diretamente os valores das variáveis de decisão. O modelo visa a determinar o valor ótimo de uma Função Objetivo, dado um conjunto de restrições.

Pode-se expressar uma modelo de Programação Linear Binária como um vetor $X = \{x_1, x_2, \cdots, x_n\}$ de n variáveis binárias, isto é, $x_j \in \{0, 1\}\ \forall j = 1, \cdots, n$, estando a cada qual associado um custo $c_j \in \mathcal{R}$. Sejam, também, b um vetor m-dimensional com elementos $b_i \in \mathcal{R}$ e A uma matriz de dimensões $m \times n$, com elementos $a_{ij} \in \mathcal{R}$, sendo $1 \leq i \leq m$ e $1 \leq j \leq n$. A formulação matemática do problema de PLB pode, então, ser expressa pelas equações (2.1) a (2.3):

$$\max\,(ou\ \min)\ \sum_{j=1}^{n} c_j x_j \tag{2.1}$$

Sujeito a:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \ \forall i = 1, \cdots, m \tag{2.2}$$

$$x_j \in \{0, 1\} \ \ \forall j = 1, \cdots, n \tag{2.3}$$

Assim, a equação (2.1) representa a função objetivo, expressa pela soma das n variáveis de decisão associadas a seus respectivos custos. A equação (2.2) representa as m restrições do problema. A equação (2.3) expressa que o domínio das variáveis de decisão é binário.

Para exemplificar o modelo, consideremos o seguinte problema:

$$max \ Z = 4x_1 + x_2 \tag{2.4}$$

Sujeito a:

$$9x_1 + x_2 \leq 18 \tag{2.5}$$

$$3x_1 + x_2 \leq 12 \tag{2.6}$$

$$x_1, x_2 \in \{0, 1\} \tag{2.7}$$

A equação (2.4) representa a função objetivo do problema, que busca maximizar Z, atendendo às restrições do problema, representadas pelas equações (2.5) a (2.7).

Neste exemplo, ao ativar as variáveis de decisão x_1 e x_2 com o valor 1, observa-se que as restrições impostas pelas equações (2.5) e (2.6) são satisfeitas, e que a função objetivo assume o valor $Z = 5$. Neste caso, esta solução é ótima, visto que o gradiente da função objetivo é (4, 1) e o maior valor que cada variável de decisão pode assumir é 1.

Capítulo 3

Revisão Bibliográfica

Neste capítulo é feita uma breve revisão bibliográfica de alguns trabalhos relevantes relacionados ao tema.

3.1 Simplex

Os primeiros estudos para a resolução de funções lineares sujeitas a restrições foram desenvolvidos por Fourier (Fourier, 1890), em seu trabalho sobre sistemas lineares de inequações. Entretanto, somente em 1947 foi desenvolvido por George Dantzig (Dantzig, 1951) o método SIMPLEX, que foi o primeiro algoritmo para solução de problemas de programação linear.

O SIMPLEX explora o fato de que a solução ótima do problema ser uma solução básica viável, o que significa, do ponto de vista geométrico, que ela é um ponto na extremidade de um politopo gerado pelas restrições do problema. Assim, a partir de um vértice inicial, o método move-se para um vértice adjacente a este enquanto forem encontradas soluções de melhora.

3.2 Local Branching

Em Fischetti e Lodi (2003) os autores propõem o uso de um resolvedor de programação linear inteira como uma ferramenta caixa-preta para explorar eficientemente subespaços das soluções (vizinhanças) definidas e controladas por um algoritmo de *branching* externo

simples. Esta estratégia busca explorar as vizinhanças das soluções factíveis obtidas, utilizando-se, para isso, a ideia de busca local. Na busca local proposta pelos autores, as vizinhanças são obtidas introduzindo-se diversos cortes no modelo inteiro, chamados *local branching cuts*.

Seja um modelo P genérico com o conjunto indexado de variáveis $N = 1, \cdots, n$ divididas nos subconjuntos (B, G e C), sendo B o conjunto de índices das variáveis binárias, G o conjunto de índices das variáveis inteiras, e C o conjunto de índices das variáveis contínuas. O método *Local branching* será explicado com base no modelo de programação inteira mista (PIM) descrito a seguir:

$$(P)\ min\ c^T\ x \tag{3.1}$$

$$s.a:\ Ax \geq b \tag{3.2}$$

$$x_j \in \{0,1\} \quad \forall j \in B \neq 0 \tag{3.3}$$

$$x_j \geq 0,\ inteiro\ \forall j \in G\ \ x_j \in \mathbb{N} \tag{3.4}$$

$$x_j \geq 0\ \ \forall j \in C \tag{3.5}$$

Desta forma, dada uma solução de referência $\bar{x}$ válida em P, define-se um conjunto $\bar{S}$, chamado de suporte binário de $\bar{x}$, tal que:

$$\bar{S} = \{j \in B \,|\, \bar{x} = 1\}$$

ou seja, o conjunto $\bar{S}$ contém os índices de todas as variáveis binárias que possuem o valor 1. Também é introduzido um parâmetro k, a partir do qual é definida a vizinhança k-OPT *N($\bar{x}$,k)* de $\bar{x}$. A função *N($\bar{x}$,k)* representa todas as soluções factíveis do problema que satisfazem a restrição de *local branching*, dada por:

$$\Delta(x, \bar{x}) = \sum_{j \in \bar{S}} (1 - x_j) + \sum_{j \in B \setminus \bar{S}} x_j \leqslant k \tag{3.6}$$

Na restrição (3.6), os termos à esquerda representam as variáveis binárias que terão seus valores trocados de 1 para 0 ou de 0 para 1, respectivamente, ou seja, o número total de variáveis binárias as quais terão seu valor alterado. Quando esta restrição é acrescentada ao modelo, o espaço de soluções do problema original é reduzido, e pode-se buscar a melhor solução vizinha de $\bar{x}$. Por outro lado, é possível buscar a solução ótima que seja vizinha de $\bar{x}$ na vizinhança complementar a (3.6), definida por:

$$\Delta(x, \bar{x}) \geqslant k + 1 \tag{3.7}$$

O parâmetro k define o tamanho da vizinhança a ser alcançada na ramificação à esquerda e é um valor de extrema importância. Os autores identificaram que uma boa faixa de valores para k seria um número no intervalo [10, 20].

A idéia geral do *Local Branching* pode se descrita como se segue: a princípio é introduzida no modelo a restrição de *local branching* (3.6), de forma a definir uma vizinhança, onde posteriormente o novo problema é resolvido, na busca de uma solução de melhor qualidade. Caso esta solução seja encontrada, ela passa a ser a solução corrente, a restrição de *local branching* é removida e a restrição 3.7 é adicionada ao modelo. Em seguida, a restrição de *local branching* é novamente adicionada ao modelo, só que, desta vez, partindo da nova solução. O procedimento é repetido até que seja encontrado um subproblema cuja solução não seja melhor do que a solução corrente.

A aplicação do *local branching* é ilustrada na Figura 3.1. O nó inicial $\bar{x}^1$ é uma solução factível do problema. O lado esquerdo contendo o nó 2 corresponde à otimização dentro da *vizinhança k-OPT* $N(\bar{x}, k)$, na qual é obtida uma solução z chamada $\bar{x}^2$. Caso esta solução seja de melhor qualidade do que a corrente, $\bar{x}^2$ se torna a nova solução. O proce-

dimento é repetido com o nó da direita 3. Desta forma, o subespaço de $N(\bar{x}^2, k) \backslash N(\bar{x}^1, k)$ no nó 4 produz uma nova solução corrente $\bar{x}^3$, solução esta que será utilizada para definir o subespaço a ser explorado no ramo esquerdo do nó 5. No exemplo, o nó 6 produz um subproblema cuja solução não é melhor do que a solução corrente. Nesta situação, a adição da constante $\Delta(x, \bar{x}^3) \geq k + 1$ leva ao nó 7, da direita.

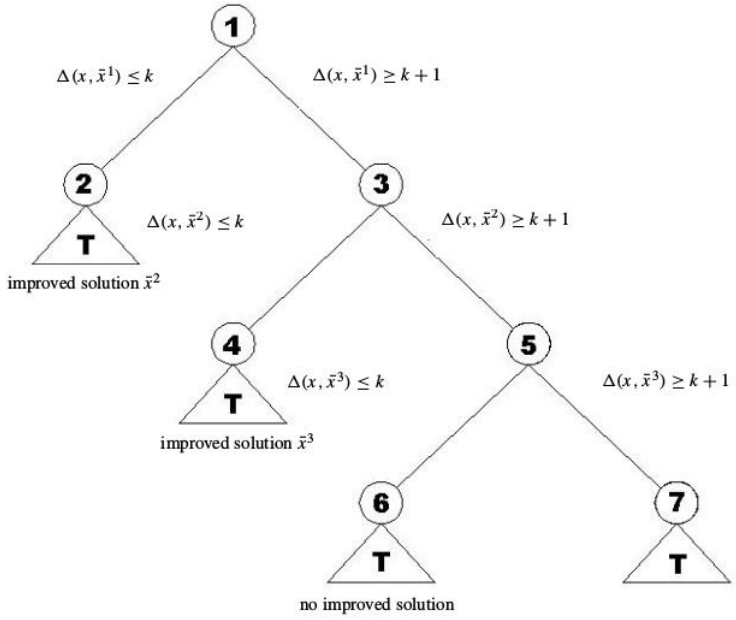

Figura 3.1: Exemplo do funcionamento do *Local Branching*. Adaptado de Fischetti e Lodi (2003).

O ponto crítico em métodos de fixação de variáveis está relacionado à escolha das variáveis a serem fixadas em cada passo para a geração dos cortes. Para problemas mais complexos, só se pode resolvê-los após vários ciclos de fixação.

3.3 Nogood Learning

Em Sandholm e Shields (2006) é sugerida a técnica *Nogood*, baseada em aprendizagem, para uso em programação inteira e programação inteira mista. Esta técnica gera planos de corte globalmente válidos para algoritmos de busca binária, a partir de informações aprendidas por meio de propagação de restrições.

Os cortes são gerados não somente quando há inviabilidade, mas também quando existe a necessidade de fixar uma variável em um valor 0 ou 1. Porém, os experimentos

realizados pelos autores mostraram que isoladamente tais cortes não ajudam a PI, pois alguns dos planos de corte gerados são fracos.

3.4 LocalSolver 1.x

Em Benoist et al. (2011) é apresentado um resolvedor comercial que utiliza uma heurística de busca local para otimizar problemas binários lineares e não-lineares.

O software *LocalSolver* permite ao usuário concentrar-se na modelagem do problema usando um formalismo simples, porém com técnicas eficientes e confiáveis.

Basicamente o software trata todas as variáveis de decisão como booleanas. As expressões são construídas sobre estas variáveis utilizando lógica, aritmética, ou operadores relacionais. Em resumo, *LocalSolver* utiliza uma sintaxe funcional, sem limitação no aninhamento de expressões. O software permite também que variáveis auxiliares (de folga) sejam introduzidas, assim como operadores relacionais como o $\leq$, dando mais expressividade e eficiência ao modelo, o que representa uma vantagem importante em relação ao *Local Branching* (Fischetti e Lodi, 2003).

Entretanto, devido à natureza comercial do produto, os algoritmos implementados são apenas superficialmente descritos pelos autores. Segundo os autores, o software alcança boas soluções em alguns problemas difíceis da biblioteca MIPLIB 2010.

3.5 pRINS

Em Gomes (2014) é apresentado um algoritmo, denominado pRINS, que explora técnicas de pré-processamento, procurando por um número ideal de fixações, visando a produzir sub-problemas de tamanho controlado. O método parte de duas soluções, sendo uma inteira e outra fracionária. Posteriormente, considerando estas soluções, as variáveis fixadas são organizadas por um vetor de prioridades. Assim, dado um tamanho desejado de problema (dimensão máxima), o método procura um número de fixações que encontre este tamanho. Posteriormente, os problemas são criados e resolvidos de modo semelhante ao método *Variable Neighborhood Descent*. O método é executado enquanto o tamanho máximo do problema não é atingido ou o tempo ainda não estiver esgotado.

3.6 O método BPLS

No trabalho Brito et al. (2014), os autores propõem um método de busca local, nomeado BPLS, para resolver problemas binários. O método desenvolvido considera a estrutura do problema como um grafo de conflitos e utiliza um procedimento de geração de vizinhos para percorrer as soluções usando cadeias de movimentos. O método apresenta uma heurística híbrida caracterizada por duas fases: uma fase construtiva e uma fase de busca local. Ambas as fases trabalham com informações fornecidas por um grafo de conflitos, criado a partir da análise das restrições impostas pela entrada do problema. O método visa a produção rápida de soluções viáveis, porém quando o tempo de processamento aumenta, o método não consegue ampliar significativamente o número de soluções viáveis encontradas durante a busca.

3.7 COIN-OR-CBC

O projeto COIN-OR surgiu a partir de um consórcio de pesquisadores da indústria e academia dedicados ao desenvolvimento da pesquisa operacional, e hoje pertence a uma corporação não lucrativa: COIN-OR Foundation.

O COIN-OR-CBC (Foundation, 2015) é um resolvedor de código aberto voltado à resolução de problemas lineares e inteiros escrito em C++. O pacote inclui recursos como pré-processamento, planos de corte, heurísticas e estratégias de *branching*.

Ele pode ser usado como um resolvedor independente chamado pela linha de comando, possuindo também funcionalidades que permitem execução em modo paralelo para o aproveitamento de computadores com vários núcleos de processamento. As instâncias podem ser informadas ao resolvedor nos formatos de arquivo .lp e .mps.

O resolvedor pode também ser usado como uma biblioteca em demais códigos, permitindo aos desenvolvedores sua utilização em aplicação de algoritmos *branch-and-cut* utilizando as bibliotecas do CBC.

Quando utilizado como resolvedor independente, ele apresenta bons resultados, quando comparado aos demais de sua categoria; entretanto, essas soluções podem ser melhoradas quando ele é utilizado como biblioteca para demais aplicações.

Capítulo 4

Metodologia

Neste capítulo é descrito o método híbrido proposto para resolver problemas lineares binários. Nomeado HGVPRLB, ele combina várias técnicas da literatura que são usadas para resolver problemas de otimização combinatória.

A Figura 4.1 apresenta um esquema geral do método, sendo ele apresentado em detalhes na Seção 4.2.

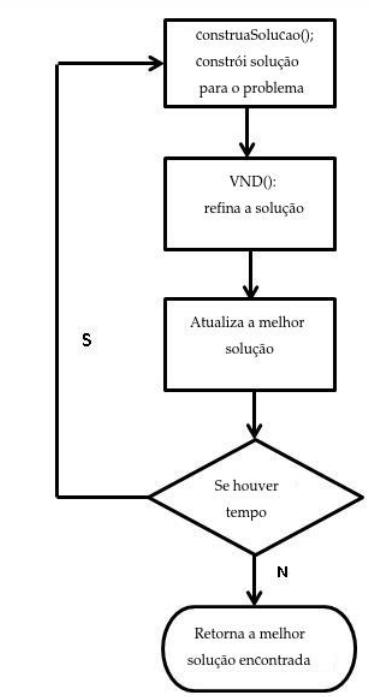

Figura 4.1: Esquema geral do método HGVPRLB.

O método HGVPRLB combina as técnicas GRASP, propagação de restrições, VND

e cortes *local branching*. A ideia ao se fazer essa combinação é explorar as características positivas de cada uma dessas técnicas, gerando um método mais eficiente.

Tal como um algoritmo GRASP clássico, há duas fases: uma construtiva, mostrada no esquema da Figura 4.2 e outra de refinamento da solução construída, mostrada no esquema da Figura 4.3. A fase construtiva é descrita em detalhes na Seção 4.2.1 e na Seção 4.2.2 a fase de refinamento é apresentada.

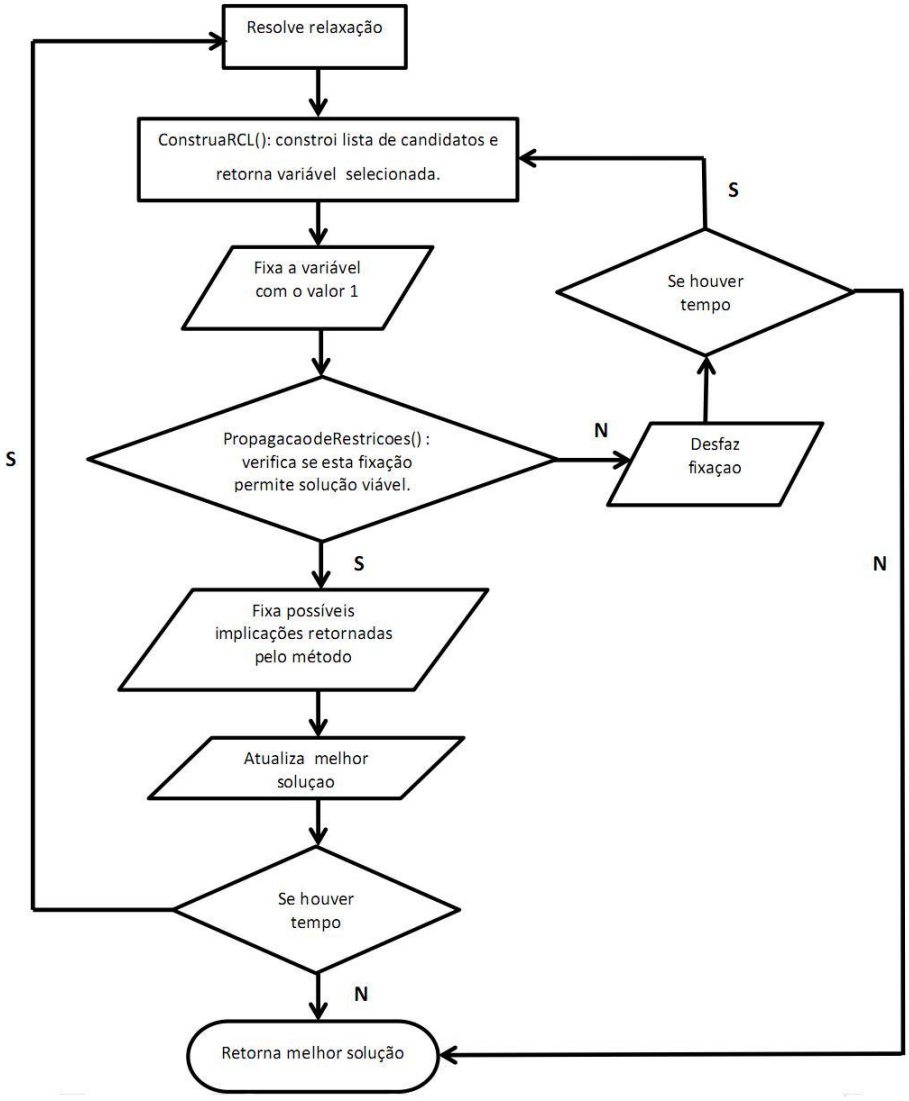

Figura 4.2: Esquema geral da fase construtiva.

A fase contrutiva utiliza o resolvedor CBC para obter uma solução relaxada do problema binário, criar uma lista restrita de variáveis candidatas a assumir o valor binário 1 e o método de propagação de restrições para fixar o valor das demais variáveis que foram

afetadas com a fixação do valor da variável candidata. O funcionamento da lista restrita de candidatos e o método de propagação de restrições são explicados em detalhes nas seções 4.2.1.1 e 4.2.1.2, respectivamente.

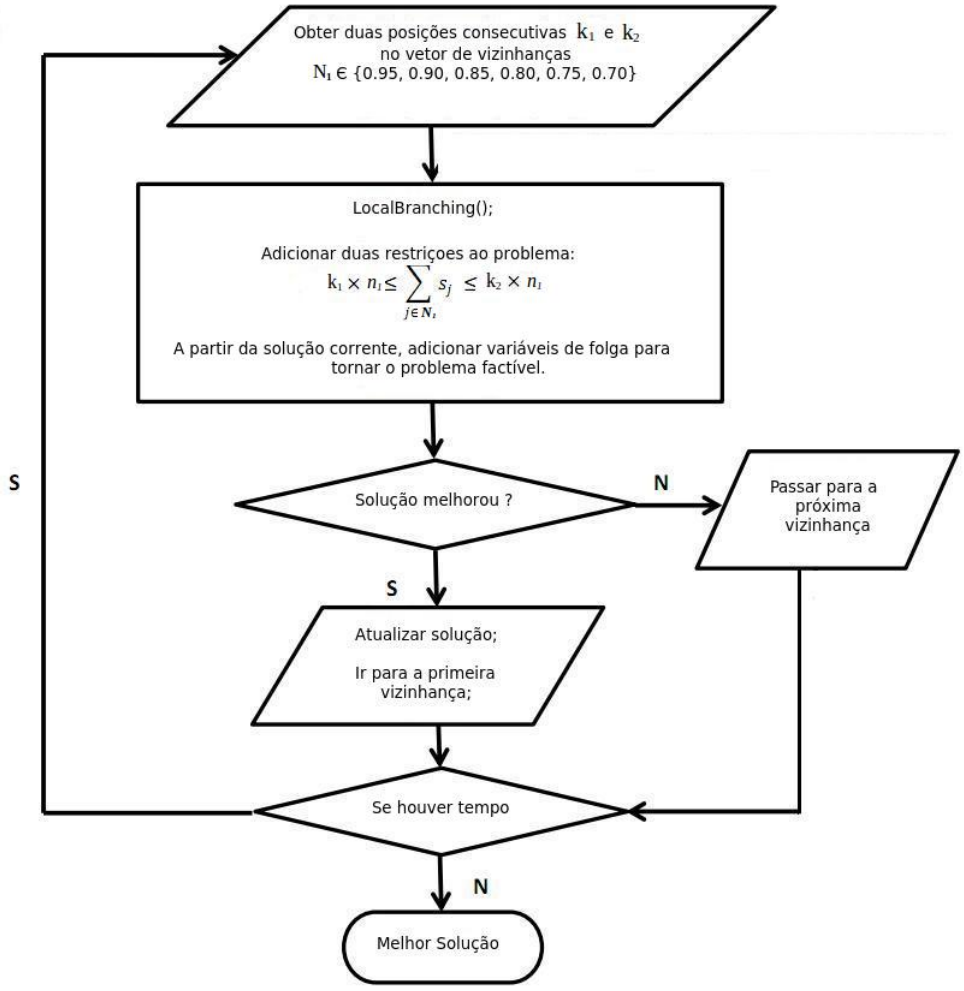

Figura 4.3: Esquema geral da fase de refinamento.

A fase de busca local, mostrada pela Figura 4.3, é baseada na heurística VND e são utilizados os cortes *local branching*, descritos na Seção 4.2.2.1. Nessa Figura, k_1 e k_2 são dois valores consecutivos do conjunto de vizinhanças $N = \{0.95, 0.90, \cdots, 0.75\}$

4.1 Representação da Solução

A Figura 4.4 apresenta um exemplo de problema binário com o objetivo de minimizar o valor da função objetivo.

$$
\begin{array}{lllllll}
min & x_1 + 1.1x_2 + 1.2\,x_3 + x_4 + x_5 + x_6 & & & & & \\
s.t. & -x_1 & -x_2 & +x_3 & & & = -1 \\
 & & & -x_3 + x_4 & & & \geq 0 \\
 & & & -x_3 & +2x_5 & & \geq 1 \\
 & & & & -x_4 & +x_6 & \geq 0 \\
 & & & & & -x_5 - x_6 & \geq -1 \\
 & & & & & & x_j \in \{0,1\}
\end{array}
$$

Figura 4.4: Exemplo de um problema binário.

Uma solução s para este problema binário é representada por um vetor de números binários com n posições, sendo que cada posição representa uma variável do problema. Se a variável estiver ativa, ela assume o valor 1; caso contrário, é representada pelo valor binário 0.

Considerando o exemplo de problema binário da Figura 4.4, uma solução para o problema é representado pela Figura 4.5. Nesta solução, as variáveis x_1 e x_5 assumem o valor binário 1, e as variáveis x_2, x_3, x_4 e x_6 assumem o valor 0. O valor da função objetivo, de acordo com a solução apresentada, é igual a 2.

$$
s = [\underbrace{1\ 0\ 0\ 0\ 1\ 0}_{0 \ldots n}]
$$

Figura 4.5: Exemplo de solução para o problema 4.4.

4.2 O Algoritmo HGVPRLB proposto

O método HGVPRLB é um algoritmo **H**íbrido baseado em **G**RASP, **V**ND, **P**ropagação de **R**estrições e ***L**ocal **B**ranching*, destinado a resolver Problemas Lineares Binários. Seu pseudocódigo é apresentado no Algoritmo 4.1.

Algoritmo 4.1: HGVPRLB

Entrada: *LP, tempoLimite,* β*,* γ*, max_iter, t_vnd,* θ
Saída: $s^\star$

```
1  f* ← +∞;
2  t ← 0;
3  enquanto t < tempoLimite faça
4  |  s ← construaSolucao(LP, β, γ, max_iter, θ);
5  |  tempoVND ← min{tempoRestante, t_vnd};
6  |  s ← VND(LP, s, tempoVND);
7  |  se f(s) < f* então
8  |  |  s* ← s;
9  |  |  f* ← f(s*);
10 |  Atualize t;
11 Retorne s*;
```

No Algoritmo 4.1, tem-se, como dados de entrada, o programa linear *LP*, o tempo limite *tempoLimite* de execução, o valor $\beta \in [0, 1]$, o valor de γ definido em 0.01, o número máximo de iterações *max_iter* a ser passado para o Algoritmo 4.2, o tempo de execução *t_vnd* utilizado no Algoritmo 4.6, o valor θ utilizado para liberar as variáveis fixas com valores 0 e 1 quando o resolvedor encontrar dificuldade na relaxação das variáveis.

As fases contrutiva e de refinamento, que compreendem a heurística GRASP, são aplicadas iterativamente até que o tempo limite seja atingido.

Na linha 4 do Algoritmo 4.1, uma solução é construída por meio do procedimento *construaSolucao(LP,*β, γ*,max_iter,*θ*)*, definido na Seção 4.2.1. Na linha 5 é calculado o menor valor entre o tempo restante e o tempo *t_vnd*. Este último é o tempo de execução do procedimento *VND(LP,s,tempoVND)*, acionado na linha 6 e descrito na Seção 4.2.2, em que a solução contruída é refinada. Na linha 7 é verificado se essa solução refinada é de qualidade superior à melhor solução $s^\star$ encontrada até então. Se a resposta for positiva, $s^\star$ é atualizada. Na linha 10, o tempo é atualizado.

Nas subseções seguintes o Algoritmo 4.1 é detalhado.

4.2.1 Fase Construtiva

O procedimento *construaSolucao(LP,*β, γ*,max_iter,*θ*)* tem como objetivo construir uma solução inicial para o Algoritmo HGVPRLB. Seu pseudocódigo é definido pelo Algoritmo 4.2.

Este procedimento recebe como parâmetros: o problema linear LP; o valor β necessário para calcular a lista restrita de candidatos; o valor γ, que define como candidatas somente variáveis cujo valor é maior ou igual a γ; o valor *max_iter*, que define o número máximo de iterações na fase de construção; e o valor θ, que define a porcentagem de variáveis que serão liberadas quando o resolvedor encontrar solução viável na relaxação.

Na linha 3 do Algoritmo 4.2 é acionado o resolvedor *CBC* aplicado ao problema LP relaxando as variáveis binárias, isto é, considerando-as no intervalo $[0, 1]$.

Nas linhas 4 a 7 é verificado se o resolvedor não encontrou uma solução viável para o problema. Se isto ocorrer, são liberadas, tanto do problema LP, quanto da solução s, as θ variáveis anteriormente fixadas. Esta verificação se repete até se obter uma solução viável relaxada para o problema.

A seguir, na linha 8, se foi encontrada uma solução viável, esta é atribuída ao vetor v. Na linha 9 é chamado o procedimento *ConstruaRCL(LP,v,*γ, β*)*, definido na Subseção 4.2.1.1, cujo objetivo é retornar o índice de uma variável que será fixada no valor 1, tanto no problema LP quanto na solução s do problema. Fixada essa variável $s_{j^\star}$, é chamado o procedimento *PropagacaodeRestricoes(LP,l,u,C)*, descrito na Seção 4.2.1.2. O objetivo desse procedimento é verificar inicialmente se existe solução viável, fixando-se a variável $s_{j^\star}$ no valor 1. Se esta solução não existir, na linha 13 a fixação da variável $s_{j^\star}$ é liberada tanto no problema LP quanto na solução s. Caso ela exista, ele retornará também um conjunto I de variáveis $s_j \neq s_{j^\star}$ que devem ser fixadas em valores 0 ou 1 em função da fixação da variável $s_{j^\star}$. Essas fixações são realizadas na linha 17.

Na linha 18 é verificado se a solução corrente tem qualidade superior à da melhor solução encontrada até o momento. Na determinação da solução corrente s, as variáveis ainda não fixadas recebem o valor 0. As linhas 3 a 19 são executadas até o número máximo de iterações *max_iter*. Na linha 20 a melhor solução encontrada é retornada.

4.2.1.1 Lista Restrita de Candidatos (RCL)

O procedimento *ConstruaRCL(LP,v,*β, γ*)* tem por objetivo selecionar o índice de uma variável a ser fixada no valor 1. Ele é acionado após a chamada ao procedimento *CBCRelax(LP,v)*, o qual gera uma solução relaxada v para o problema LP. Este procedimento é descrito no Algoritmo 4.3 e recebe como dados de entrada o problema LP, o vetor fracionário v que possui o valor da relaxação das variáveis do problema, o valor de β e o valor γ.

Algoritmo 4.2: construaSolucao

Entrada: *LP, β, γ, max_iter, θ*
Saída: $s^\star$

```
1  s ← s* ← ∅;
2  enquanto −−max_iter ≥ 0 faça
3      result ← CBCRelax(LP,v);
4      enquanto result ≠ LP_factivel faça
5          Selecione aleatoriamente θ% das variáveis fixas em LP e s;
6          Libere de LP e s as variáveis selecionadas;
7          result ← CBCRelax(LP,v);
8      v ← solução do problema LP relaxado;
9      j* ← ConstruaRCL(LP, v, β, γ);
10     s_j* ← 1;
11     Fixe s_j* = 1 em LP;
12     I ← PropagacaodeRestricoes(LP,l,u,C);
13     se I = inviável então
14         Libere s_j* em s;
15         Libere s_j* em LP;
16     senão
17         Fixe implicações I em s e em LP;
18     se f(s) for melhor que f(s*) então
19         s* ← s;
20 Retorne s*;
```

Algoritmo 4.3: ConstruaRCL

Entrada: *LP, v, β, γ*
Saída: $j^\star$

```
1  A ← {(j, s_j) : s_j ≥ γ e j não fixado em LP};
2  max ← max_{∀j∈A}{s_j};
3  min ← min_{∀j∈A}{s_j};
4  RCL ← {j : j ∈ A e s_j ≥ max - β × (max - min)};
5  j* ← Elemento j selecionado aleatoriamente da RCL;
6  Retorne j*;
```

Na linha 1 do Algoritmo 4.3 A recebe um conjunto de índices de variáveis não fixadas em *LP*. Destas, são selecionadas as variáveis que possuem valor de relaxação maior ou igual a γ. Essas variáveis formam uma lista de variáveis candidatas a serem incorporadas a uma lista restrita de candidatos, chamada RCL (das iniciais em inglês de *Restricted Candidate List*), escolhidas de forma a se obter uma lista com diversidade e de boa qua-

lidade. Uma vez que a escolha de β influencia o tamanho da RCL e, consequentemente, a qualidade das soluções obtidas nesta fase, à medida que o tamanho da RCL aumenta, a qualidade da solução piora; entretanto, em contrapartida, é aumentada a diversidade das soluções geradas. A lista restrita de candidatos *RCL* é formada pelos índices j das variáveis s_j que satisfazem a Equação (4.1).

$$s_j \geq max - \beta \times (max \text{ - } min) \tag{4.1}$$

sendo *max* e *min*, o maior e menor valor da relaxação das variáveis, respectivamente, e obtidos nas linhas 2 e 3. Na linha 4 o vetor RCL recebe os índices das variáveis que satisfazem a Equação 4.1. A partir desta lista de candidatos, na linha 5 é selecionado de forma aleatória um índice de uma variável que será o retorno do algoritmo.

4.2.1.2 Programação por Restrições

O procedimento *PropagacaodeRestricoes(LP,l,u,C)* possui a finalidade de verificar se existe solução viável fixando uma variável $s_{j^\star}$ no valor 1. Se essa solução existir, ele retornará um conjunto I de fixações das demais variáveis $s_j \neq s_{j^\star}$ nos valores 0 ou 1. Caso contrário, ele retornará que, fixando-se a variável $s_{j^\star}$ no valor 1, a solução s será inviável.

Para a adequação a este modelo de propagação de restrições, cada restrição i é reescrita na forma da Eq. (4.2):

$$\sum_{j \in N} a_{ij} s_j \leq b_i \tag{4.2}$$

em que N é o conjunto de variáveis.

Assim, seja a medida de inviabilidade U_{ij} determinada pela Eq. (4.3):

$$U_{ij^\star} = b_i - \left(\sum_{j \in N_i^+, j \neq j^\star} |a_{ij}|\, l_j + \sum_{j \in N_i^-, j \neq j^\star} |a_{ij}|\, u_j \right) \tag{4.3}$$

em que $N_i^+ = \{j \in N : a_{ij} \geqslant 0\}$ e $N_i^- = \{j \in N : a_{ij} < 0\}$.

Em cada restrição i, as variáveis s_j a ela pertencentes possuem limites superior e inferior, respectivamente, l_j e u_j, definidos como parâmetros. Se a j-ésima variável é fixada, $l_j = u_j = s_{j^\star}$. Caso contrário, $l_j = 0$ e $u_j = 1$.

O procedimento 4.5, *avaliaInviabilidade(LP,l,u,c,j*)*, usa a medida U_{ij} para verificar se a variável s_j pode ou não ser fixada nos valores 0 ou 1. Se não puder ser fixada por existência de inviabilidade, ele retorna a existência de conflito. Caso contrário, ele pode retornar: i) O valor binário que a variável s_j deve ser fixada, ou então, ii) a indefinição do valor a ser fixado para essa variável.

Assim, este procedimento retorna o par *status,limite*, em que *status* assume o valor CONFLITO e *limite* assume o valor NULO quando s_j não puder assumir um valor binário. Por outro lado, *status* assume o valor FIXA quando s_j pode assumir um valor binário e *limite* assume o valor 0 ou 1. Finalmente, *status* assume o valor SEM_IMPLICACOES e *limite* assume o valor NULO quando há indefinição do valor a ser fixado para essa variável.

Os Algoritmos 4.4 e 4.5 apresentam os pseudocódigos baseados na implementação do algoritmo de Sandholm e Shields (2006) para detectar conflitos e implicações.

No Algoritmo 4.4 são dados de entrada o problema *LP*, os limites superior e inferior definidos anteriormente como l e u, e um conjunto de restrições C. Em cada uma delas aparece o índice j da variável s_j que foi fixada. As linhas 1 e 2 recebem, a princípio, um conjunto vazio de fixações, e o *status* definido como SEM_IMPLICACOES. Na linha 4 é atribuído o valor *falso* a uma variável, chamada de *novas_implicacoes*. C é o conjunto das restrições em que a $j^\star$-ésima variável fixada aparece.

Nas linhas 5 a 12 é realizado um laço iterativo em cada uma dessas restrições.

Nas linhas 6 a 12, para cada uma das variáveis desta restrição, é avaliado se ela pode ser fixada em algum valor de acordo com as fixações anteriores. Para isso é chamado o método *avaliaInviabilidade(LP,l,u,c,j*)* apresentado no Algoritmo 4.5. Caso o resultado retornado seja igual a FIXA, esta variável e seu limite são adicionados ao conjunto I, e a variável *novas_implicacoes* recebe *verdadeiro*. Se o *status* for igual a CONFLITO, este valor é retornado pela função.

Na linha 13, se não houve fixações, a função retorna o *status* SEM_IMPLICACOES.

Nas linhas 15 a 18 para toda fixação realizada anteriormente adicionada ao conjunto

Algoritmo 4.4: PropagacaodeRestricoes

```
   Entrada: LP, l, u, C
   Saída: (status, I)
 1 status ⟵ SEM_IMPLICACOES;
 2 I ⟵ ∅;
 3 repita
 4     novas_implicacoes ← falso;
 5     para todo restrição c em C faça
 6         para todo índice j* de variável não fixada em c faça
 7             (limite,status) ⟵ avaliaInviabilidade(LP,l,u,c,j*);
 8             se status = FIXA então
 9                 I ⟵ I ∪ {(j*, limite)};
10                 novas_implicacoes ← verdadeiro;
11             senão se status = CONFLITO então
12                 return (CONFLITO, I);
13     se |I| = ∅ então
14         return (SEM_IMPLICACOES, I);
15     para todo (j, limite) ∈ I faça
16         l_j ← u_j ← limite;
17         para todo c em C que contém uma ou mais variáveis fixadas faça
18             C ⟵ C ∪ {c};
19     para todo c em C faça
20         se todas as variáveis da restrição c estão fixadas então
21             C ⟵ C \ {c};
22 até (C ≠ 0) e novas_implicacoes = verdadeiro ;
23 Retorne (status, I);
```

I, os limites l_j e l_u dessas variáveis são atualizados. Cada restrição em que essas variáveis aparecem são adicionadas ao conjunto C.

Nas linhas 19 a 21 são excluídas as restrições pertencentes a C em que todas as variáveis já foram fixadas. Esta função é repetida até que o conjunto de restrições C seja igual a vazio, ou não houver novas implicações.

No final, linha 23 do Algoritmo 4.4, o procedimento *PropagacaodeRestricoes(LP,l,u,C)* retorna o par *status* e o conjunto I de fixações realizadas.

O Algoritmo 4.5 tem como dados de entrada o problema binário LP, os limites superior e inferior l e u de cada variável, a restrição i em que está contida a variável $s_{j^\star}$, e o índice $j^\star$ dessa variável a ser verificada.

Algoritmo 4.5: avaliaInviabilidade

Entrada: *LP, l, u, i, $j^\star$*
Saída: (*status,limite*)

```
1  U_ij* ⟵ Calculo_U;
2  se j* ∈ N_i^+ então
3      se U_ij* < 0 então
4          limite ⟵ NULO;
5          status ⟵ CONFLITO;
6      senão se a_ij* > U_ij* então
7          limite ⟵ 0;
8          status ⟵ FIXA;
9      senão
10         limite ⟵ NULO;
11         status ⟵ SEM_IMPLICACOES;
12 se U_ij* ∈ N_i^- então
13     se a_ij* > U_ij* então
14         limite ⟵ NULO;
15         status ⟵ CONFLITO;
16     senão se U_ij* < 0 então
17         limite ⟵ 1;
18         status ⟵ FIXA;
19     senão
20         limite ⟵ NULO;
21         status ⟵ SEM_IMPLICACOES;
22 Retorne (status,limite);
```

Na linha 1 do Algoritmo 4.5 a inviabilidade $U_{ij^\star}$ é calculada, de acordo com a Equação (4.3).

Nas linhas 2 a 11, se a variável $s_{j^\star}$ possuir um coeficiente positivo, ela é analisada da seguinte forma: se a inviabilidade $U_{ij^\star}$ for menor que zero, o *status* recebe a definição CONFLITO e o *limite* recebe NULO. Se o coeficiente da variável $s_{j^\star}$ for maior que o valor $U_{ij^\star}$, o *status* é definido como FIXA, e o *limite* será 0. Se nenhuma das condições anteriores forem válidas, o *status* será definido como SEM_IMPLICACOES e o *limite* será NULO.

Nas linhas 12 a 21, se a variável analisada possuir um coeficiente negativo, ela é analisada de forma análoga ao caso anterior, sendo que: se o coeficiente da variável $s_{j^\star}$ for maior que o valor $U_{ij^\star}$, o *status* é definido como CONFLITO, e o *limite* como NULO. Se o valor $U_{ij^\star}$ for menor que zero, o *status* recebe a definição FIXA e o *limite* recebe o

valor 1. Se nenhuma das condições anteriores forem válidas, o *status* será definido como SEM_IMPLICACOES e o *limite* será NULO.

Na linha 22 é retornado o par *status* e *limite*.

4.2.1.2.1 Exemplo De forma a exemplificar o procedimento de Programação de Restrições, consideremos o problema da Figura 4.4 anteriormente apresentado:

No procedimento *PropagaçãodeRestrições* vamos considerar duas situações:

Primeiramente, consideremos que temos duas variáveis fixadas, sendo elas: x_1 com valor igual a 1 e x_2 também com valor igual a 1. Assim, por propagação e de acordo com a restrição 1, a variável x_3 assumiria o valor 1 e, consequentemente, de modo a atender as restrições 2 e 3, as variáveis x_4 e x_5 assumiriam também o valor 1. Conforme ilustrado na Figura 4.6, para atender à restrição 4, a variável x_6 assumiria o valor 1. Porém, para satisfazer a restrição 5, a mesma variável x_6 deveria assumir o valor 0. Isto é, foi detectado um conflito na variável x_6.

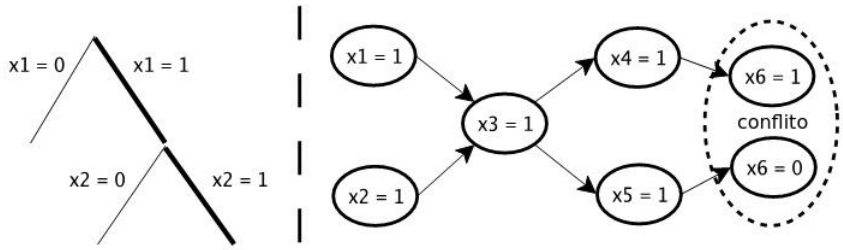

Figura 4.6: Exemplo de detecção de conflitos.

No segundo caso, podemos considerar que, a princípio, temos as variáveis x_1 com valor 1 e x_2 com valor 0 conforme mostrado na Figura 4.7. Assim, consequentemente, de modo a atender à restrição 1, a variável x_3 assumiria o valor 0. Para satisfazer às restrições 2 e 3, as variáveis x_4 e x_5 devem assumir os valores 1 e 0, respectivamante. Desta forma, as restrições 4 e 5 também seriam satisfeitas com a variável x_6 assumindo o valor 1.

4.2.2 Refinamento - VND

Nesta fase, uma solução inicial é refinada diversas vezes durante um tempo determinado. Tal fase é controlada pelo procedimento *Variable Neighborhood Descent* – VND

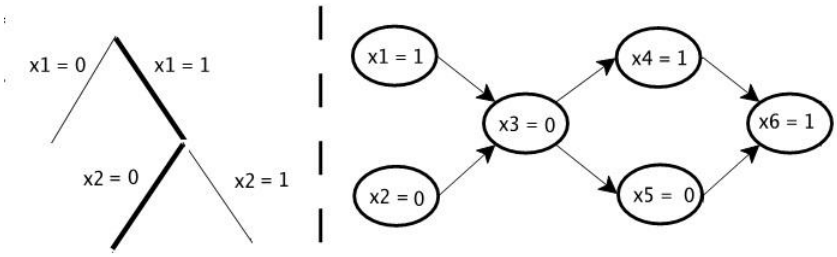

Figura 4.7: Exemplo de solução viável encontrado a partir de implicações de variáveis.

Mladenović e Hansen (1997). VND é um método de busca local que consiste em explorar o espaço de soluções por meio de trocas sistemáticas de estruturas de vizinhança previamente ordenadas.

O procedimento utiliza a primeira estrutura de vizinhança visando a melhora da solução corrente. Quando não é mais possível essa melhora na vizinhança corrente, o método inicia sua busca na próxima vizinhança. O procedimento retorna à primeira vizinhança quando uma melhor solução é encontrada. O método se encerra após aplicar todas as estruturas de vizinhança sem conseguir melhorar a solução corrente.

Neste trabalho consideramos um conjunto de estruturas de vizinhança, cada qual formada pela fixação de um certo percentual de variáveis da solução oriunda da fase de construção que possuem valor igual a 1, variando-se em 5% o número de variáveis a serem fixadas no intervalo 95% a 70%. Assim, a primeira vizinhança contém 95% das variáveis fixadas no valor 1, a segunda contém 90% e assim por diante. Para exemplificar, sejam $N1 = \{j : \ s_j = 1\}$ na fase de construção e $n_1 = |N1|$. Se estivermos na primeira vizinhança, isto é, com 95% das variáveis fixadas no valor 1, então serão adicionadas duas restrições ao problema LP, dadas pelas equações (4.4):

$$\lceil 0.90 \times n_1 \rceil \leq \sum_{j \in N1} s_j \leq \lceil 0.95 \times n_1 \rceil \tag{4.4}$$

Desta maneira, a princípio se busca apertar a solução, fixando um número maior de variáveis, e, à medida que o resolvedor encontra dificuldade em encontrar uma solução a partir desses limites, eles são relaxados e se inicia uma nova busca com um menor número de variáveis fixadas. Assim, procuramos fixar um número grande de variáveis,

mas com um grau de liberdade razoável, possibilitando ao resolvedor encontrar melhores soluções.

O Algoritmo 4.6 mostra o pseudocódigo do procedimento VND usado na fase de busca local do algoritmo HGVPRLB implementado para gerar os cortes *Local Branching*.

Algoritmo 4.6: VND

Entrada: *LP, s, tempoVND*
Saída: $s^\star$

```
1  nLvizinhanca[] ← {0.95, 0.90, 0.85, 0.80, 0.75, 0.70};
2  t ← 0;
3  k ← 0;
4  s* ← s;
5  repita
6      n1 ← número de variáveis de s fixadas no valor 1;
7      αl ← nLvizinhanca[k] ;
8      αu ← nLvizinhanca[k + 1];
9      coef_sup ← ⌈n1 × αl⌉;
10     coef_inf ← ⌈n1 × αu⌉;
11     tempoLB ← tempoVND − t;
12     s ← LocalBranching (LP, tempoLB, coef_sup, coef_inf, s) ;
13     se f (s) é melhor que f (s*) então
14         s* ← s;
15         k ← 0;
16     senão
17         k ← k + 1;
18 até t > tempoVND ;
19 Retorne s*;
```

No Algoritmo 4.6 são dados de entrada o problema linear *LP*, a solução s, e o tempo de execução *tempoVND*.

Na linha 1 é definido o vetor que contém os intervalos da estrutura de vizinhança, definida como *nLvizinhanca*[].

Nas linhas 2 e 3 são inicializados o tempo de execução do procedimento e o valor de k é setado para iniciar na primeira estrutura de vizinhança.

Na linha 4 a melhor solução é atualizada de acordo com a solução corrente.

Na linha 6, a variável n_1 recebe o número de variáveis que, na solução, possuem valor igual a 1. Estas serão as variáveis utilizadas para os cortes *Local Branching*. Estes cortes possuem limites inferior e superior que serão definidos a partir de valores da estrutura de vizinhança corrente.

Na linha 7, α_l recebe o valor da posição do vetor *nLvizinhanca*[], referente à posição atual de k.

Na linha 8 α_u, recebe o valor da posição do vetor *nLvizinhanca*[], referente à posição $k+1$.

Nas linhas 9 e 10 são calculados os valores de $coef_{sup}$ e $coef_{inf}$, de acordo com o valor de n_1, α_l e α_u, respectivamente. Os referidos coeficientes assumem o valor da soma de todas as variáveis que possuem valor 1 na solução, multiplicados por α_l e α_u, respectivamente.

Na linha 11 é calculado o tempo disponível para a execução do procedimento *LocalBranching (LP,tempoLB,*$coef_{sup}$*,*$coef_{inf}$*,s)*, apresentado na Seção 4.2.2.1. Assim, a variável *tempoLB* recebe o tempo de entrada *tempoVND* diminuído do tempo corrente *t*. Este tempo pode ser gasto totalmente ou parcialmente pelo procedimento *LocalBranching (LP,tempoLB,*$coef_{sup}$*,*$coef_{inf}$*,s)*.

Na linha 13 é verificado se a solução melhorou, e caso afirmativo, a nova solução é armazenada, e a variável k recebe o valor 0. Se isto não ocorrer, na linha 17 o valor de k é incrementado em uma unidade.

As linhas 5 a 17 são repetidas enquanto houver tempo. Na linha 19 é retornada a melhor solução encontrada.

4.2.2.1 Local Branching

O procedimento aqui proposto é baseado no trabalho Fischetti e Lodi (2003), que propõe o uso do resolvedor de programação linear inteira como uma ferramenta caixa-preta para explorar eficientemente subespaços das soluções (vizinhanças). Neste procedimento foi utilizado como resolvedor o CBC.

Para mostrar o funcionamento dos cortes *local branching*, seja a solução corrente s com valores binários já definidos para suas variáveis s_j. Então, para cada restrição i não satisfeita, é introduzida uma variável, com coeficiente adequado, de sorte a torná-la satisfeita. Por exemplo, se a restrição for $2s_1 + s_2 \leq 1$, e s_1 e s_2 estão assumindo valor 1 na solução corrente, então à esta restrição será adicionada a variável s_3 com coeficiente -2, de modo a torná-la satisfeita. Assim, ela passa a ser escrita como $2s_1 + s_2 - 2s_3 \leq 1$.

Genericamente, é subtraído do lado esquerdo de cada restrição i não satisfeita, do tipo $\leq$, dada por $\sum_{j=1}^{n} a_{ij}s_j \leq b_i$, uma variável s_{n+i} cujo coeficiente $a_{i,n+i}$ é definido pela

Equação (4.5):

$$a_{i,n+i} = b_i - \sum_{j=1}^{n} a_{ij} s_j \tag{4.5}$$

resultando, assim, na nova restrição i válida, dada pela Equação (4.6):

$$\sum_{j=1}^{n} a_{ij} s_j - a_{i,n+i} s_{n+i} \leq b_i \tag{4.6}$$

De forma análoga, a cada restrição i não satisfeita do tipo $\geq$ é adicionada, ao lado esquerdo da restrição, uma variável s_{n+i}, cujo coeficiente $a_{i,n+i}$ é dado pela Equação (4.7):

$$a_{i,n+i} = \sum_{j=1}^{n} a_{ij} s_j - b_i \tag{4.7}$$

No caso de a restrição i ser do tipo $=$, então é adicionada ou subtraída uma variável s_{n+i}, conforme o caso, de sorte a transformá-la em uma restrição satisfeita.

Considerando o problema como sendo de minimização tais variáveis também são acrescentadas à função objetivo do problema e recebem coeficientes positivos altos em relação aos demais, caso o objetivo do problema seja maximizar as variávies acrescidas recebem recebem coeficientes baixos em relação aos demais.

Tornadas satisfeitas todas as restrições, a seguir são criadas duas novas restrições, definidas pela equações (4.8). Essas restrições são adicionadas ao modelo original de programação linear inteira e cumprem a função do *local branching*.

$$\lceil \alpha_l \times n_1 \rceil \leq \sum_{j \in N1} s_j \leq \lceil \alpha_u \times n_1 \rceil \tag{4.8}$$

Nas equações (4.8), α_l e α_l são valores contíguos pertencentes ao conjunto de vi-

zinhanças $nLvizinhanca = \{0,95; 0,90; \cdots ; 0,75; 0,70\}$, $N1 = \{j : s_j = 1\}$ e n_1 é o número de variáveis da solução corrente que assumem o valor binário 1, incluindo as variáveis que foram adicionadas às restrições anteriormente não satisfeitas.

A ideia do método é que a k-ésima vizinhança *nLvizinhanca* de uma solução s deve ser "suficientemente pequena" para ser otimizada em um curto tempo de processamento, mas "grande o bastante" para conter soluções melhores do que s, sendo os valores de k controlados pela heurística VND, com $k \in nLvizinhanca$.

O Algoritmo 4.7 apresenta o pseudocódigo do procedimento de busca local *Local-Branching* utilizado para resolver o problema binário descrito.

Algoritmo 4.7: *LocalBranching*

Entrada: *LP, tempoLB,* $coef_{sup}$, $coef_{inf}$, s
Saída: $s^\star$

```
1 s* ← s;
2 se solução é inviável então
3 |  LP ← variáveis de folga;
4 LP ← adiciona cortes local branching considerando coef_inf ;
5 LP ← adiciona cortes local branching considerando coef_sup ;
6 LP ← adiciona variáveis de folga com valor alto de coeficiente;
7 define limite de tempo t para otimização;
8 s* ← otimiza();
9 Retorne s*;
```

O Algoritmo 4.7 recebe, como dados de entrada, o problema linear LP e um tempo *tempoLB* de execução. A melhor solução é atulizada na linha 1. No que se refere à linha 2, é verificada se a solução é inviável, e, em caso afirmativo, na linha 3 são acrescidas as variáveis de folga às restrições do problema que não foram obedecidas, de forma que todas as restrições passem a ser respeitadas.

A partir das variáveis de folga, são acrescidas ao problema duas novas restrições nas linhas 4 e 5, os chamados cortes *Local Branching* de Fischetti e Lodi (2003). Por se tratarem de problemas de minimização, com o objetivo de as variáveis de folga possuírem valor igual a 0, na linha 6 estas são adicionadas à função objetivo do problema com valores de coeficientes altos.

Na linha 7 é definido o tempo de otimização, ou seja o tempo que o resolvedor *CBC* será executado na busca de uma solução de melhor qualidade. Na linha 8, o problema é otimizado, e finalmente, na linha 9, a solução modificada é passada ao resolvedor *CBC*

para que ele retorne uma solução binária. Posteriormente, se uma nova solução não for encontrada, o procedimento retorna a melhor solução obtida até o momento.

Capítulo 5

Experimentos e Resultados Computacionais

Neste capítulo são apresentados o ambiente computacional usado para desenvolver e testar o método proposto, assim como os resultados dos experimentos computacionais.

Na Seção 5.1 são apresentados os problemas-teste utilizados para realizar os experimentos computacionais com o algoritmo proposto, enquanto na Seção 5.2 o ambiente computacional de desenvolvimento e testes é apresentado. Na Seção 5.3 são relatados os valores adotados para os parâmetros do algoritmo descrito no Capítulo 4. Finalmente, na Seção 5.4, os resultados e experimentos são detalhados.

5.1 Problemas-teste

Para os experimentos foram utilizados 32 problemas binários da biblioteca MIPLIB 2010 (Koch et al., 2011). Essa biblioteca foi criada em 1992 e sua última atualização data do ano 2010. Ela foi montada a partir da coletânea de problemas reais obtidos da indústria ou da comunidade acadêmica. Os problemas-teste nela contidos têm sido largamente usados para comparar o desempenho de resolvedores de programação inteira.

Uma descrição detalhada dos problemas-teste contendo o número de restrições, o número de variáveis binárias, o número de coeficientes não-nulos em restrições e o valor ótimo, são apresentados na Tabela 5.1.

Tabela 5.1: Problemas-teste MIPLIB 2010

Prob.-Teste	Restrições	Variáveis	Não-zeros	Valor Ótimo
acc-tight5	3052	1339	16134	0
air04	823	8904	72965	56137
bab5	8964	21600	155520	-106412
bley xl1	175620	5831	869391	190
bnatt350	4923	3150	19061	0
cov1075	637	120	14280	20
eil33-2	32	4516	44243	934,01
eilB101	100	2818	24120	1216,92
ex9	40962	10404	517112	81
iis-100-0-cov	3831	100	22986	29
iis-bupa-cov	4803	345	38392	36
iis-pima-cov	7201	768	71941	33
m100n500k4r1	100	500	2000	-25
macrophage	3164	2260	9492	374
mine-166-5	8429	830	19412	-5,66E+008
mine-90-10	6270	900	15407	-7,84E+008
mspp16	561657	29280	27678735	363
n3div36	4484	22120	340740	130800
n3seq24	6044	119856	3232340	52200
neos-1109824	28979	1520	89528	378
neos-1337307	5687	2840	30799	-202319
neos18	11402	3312	24614	16
neos-849702	1041	1737	19308	0
netdiversion	119589	129180	615282	242
ns1688347	4191	2685	66908	27
opm2-z7-s2	31798	2023	79762	-10280
reblock67	2523	670	7495	-3,46E+007
rmine6	7078	1096	18084	-457,19
sp98ic	825	10894	316317	4,49E+008
tanglegram1	68342	34759	205026	5182
tanglegram2	8980	4714	26940	443
vpphard	47280	51471	372305	5

5.2 Ambiente de Desenvolvimento

O algoritmo HGVPRLB proposto, descrito pelo Algoritmo 4.1, foi escrito na linguagem C, utilizando-se para a leitura de problemas-teste, o resolvedor de código aberto COIN-OR.

O código foi compilado no GCC, versão 4.6.3. Os experimentos foram realizados em um computador Intel Core i7-3770R, com clock de 3,4 GHz, 16 GB de RAM, em um sistema operacional Ubuntu 14.04.3 LTS Linux.

5.3 Definição dos parâmetros dos algoritmos

Após uma bateria preliminar de testes, os parâmetros do algoritmo HGVPRLB foram fixados nos seguintes valores: $\beta = 0,3$, $\gamma = 0,01$ e $\theta = 0,3$. O parâmetro t_vnd assumiu como valor a metade do tempo total de execução, isto é, $t\_vnd = tempoLimite/2$. Assim, quando o critério de parada é fixado em 60 segundos, então t_vnd assume o valor 30 segundos, e quando o tempo total de execução do algoritmo é fixado em 300 segundos, $t\_vnd = 150$ segundos. O valor max_iter inicia com 10 quando o tempo limite é 60 segundos e com 20 quando o tempo limite é 300 segundos, sendo que a cada iteração de busca local ele é multiplicado por 4.

5.4 Resultados Obtidos

O algoritmo HGVPRLB foi testado com relação à sua capacidade de encontrar uma solução viável de qualidade variando-se o tempo de processamento. De forma a poder compará-lo com outros métodos da literatura, que especificam os resultados alcançados em 60 e 300 segundos, foram utilizados esses dois valores de tempo do parâmetro *tempoLimite* como referência de comparação.

Os resultados dos experimentos realizados foram comparados com aqueles de dois dos melhores resolvedores de código aberto para programação inteira: CBC e GLPK, e também com o método BPLS de Brito et al. (2014). Em todas as comparações o critério de parada foi o mesmo.

Nas tabelas 5.2 e 5.3 são apresentados os valores da função objetivo alcançados por cada método de solução em 60 e 300 segundos, respectivamente. A penúltima linha de cada tabela indica o número de soluções viáveis encontradas por cada método, enquanto a última linha indica a quantidade de melhores soluções produzidas por cada método no conjunto de problemas-teste. As linhas em branco indicam que o método correspondente não encontrou solução viável no tempo estipulado. Em cada uma dessas tabelas, a coluna "Prob.-teste" indica o problema-teste em análise, as colunas "CBC" e "GLPK" os resolvedores homônimos, a coluna "BPLS" o algoritmo homônimo de Brito et al. (2014). A coluna "HGVPRLB" é subdividida em duas colunas, sendo que na primeira é reportado o melhor resultado encontrado em 30 execuções desse método, enquanto na segunda consta a mediana das amostras. Como em Brito et al. (2014) não é

Tabela 5.2: Valores das melhores soluções encontradas em 60 segundos

Prob.-Teste	CBC	GLPK	BPLS	HGVPRLB	
				Melhor	Mediana
acc-tight5					
air04	**56137**	57830	x	56138	56352,5
bab5					
bley xl1					
bnatt350					
cov1075	**20**	**20**	x	**20**	20,03
eil33.2	993,61	**934**	x	987,67	987,674
eilB101	1529,89	1374,96	x	**1326,05**	**1326,05**
ex9					
iis-100-0-cov	**29**	30	x	**29**	**29**
iis-bupa-cov	40	40	x	**36**	**36**
iis-pima-cov	36	36	x	**33**	34
m100n500k4r1	**-24**	**-24**	x	**-24**	**-24**
macrophage	392	495	x	**386**	407
mine-166-5	**-553252300**	-531532749,40	x	-3459148,24	873019,12
mine-90-10			x	**-13321729,51**	2598830,70
mspp16					
n3div36	**135000**		x	165200	175200
n3seq24			x	**63000**	68600
neos-1109824	**378**	**378**		**378**	**378**
neos-1337307	**-202191**	-201708		-202038	-202038
neos18	**16**	22	x	**16**	**16**
neos-849702					
netdiversion					
ns1688347					
opm2-z7-s2	-8239	-9430	x	**-9841**	-736
reblock67	**-34388335**		x	-2423910,13	-378519,56
rmine6	**-456,94**	-455	x	-454,99	-66,03
sp98ic	451003580	520753842,70	x	**450568218,08**	452374021,04
tanglegram1			x		
tanglegram2	515	**443**	x	**443**	**443**
vpphard					
#Sols Viáveis	19	17	20	21	21
#Melh. Sols	11	5		14	7

relatado o valor da função objetivo, e tão somente a existência ou não de solução viável, então, em cada célula da coluna relativa a esse método, é assinalado com a letra 'x' a existência de uma solução viável, deixando a célula em branco caso o método não consiga alcançar uma solução viável no tempo estipulado. Na penúltima linha de cada tabela é totalizado o número de soluções viáveis obtidas por cada método, enquanto na última linha é contabilizado o número de melhores soluções alcançadas por cada método.

Pela Tabela 5.2 observa-se que o algoritmo proposto encontrou o maior número de soluções viáveis, no caso, 21, o que representa uma solução a mais que o BPLS, duas

soluções a mais que o método CBC e 4 soluções a mais que o GLPK. Em relação à qualidade das soluções, observa-se que o CBC detém 11 melhores resultados, enquanto o GLPK detém 5 e o algoritmo proposto, 14.

Tabela 5.3: Valores das melhores soluções encontradas em 300 segundos

Prob.-Teste	CBC	GLPK	BPLS	HGVPRLB	
				Melhor	Mediana
acc-tight5					
air04	**56137**	56143	x	**56137**	56175
bab5	-103368,24			**-105984,96**	-102854,62
bley xl1					
bnatt350					
cov1075	**20**	**20**	x	**20**	**20**
eil33.2	**934**	**934**	x	**934**	**934**
eilB101	1227,71	1310,27	x	**1216,92**	1240
ex9				**81**	**81**
iis-100-0-cov	**29**	**29**	x	**29**	**29**
iis-bupa-cov	37	39	x	**36**	**36**
iis-pima-cov	34	36	x	**33**	34
m100n500k4r1	**-24**	**-24**	x	**-24**	**-24**
macrophage	**378**	481	x	386	393
mine-166-5	**-566395707,87**	-531532749,40	x	-3459148,24	1193317,05
mine-90-10	-783742624,05		x	**-13321729,51**	-287284,39
mspp16					
n3div36	**131400**	179600	x	140400	165200
n3seq24			x	**63000**	68600
neos-1109824	**378**	**378**	x	**378**	**378**
neos-1337307	**-202319**	-201708		-202038	-202038
neos18	**16**	20	x	**16**	**16**
neos-849702					
netdiversion					
ns1688347					
opm2-z7-s2	-10145	-10205	x	**-10257**	-10000
reblock67	-34630648	**-15541889,16**	x	-2423910,13	-378519,56
rmine6	**-457,01**	-455	x	-454,99	-66,03
sp98ic	451968910	**487333102,70**	x	449213315,20	450100069,76
tanglegram1			x		
tanglegram2	**443**	**443**	x	**443**	**443**
vpphard	**66**				
#Sols Viáveis	22	19	21	23	23
#Melh. Sols	14	8		16	9

Por outro lado, pela Tabela 5.3, observa-se que o algoritmo proposto foi capaz de melhorar a qualidade das soluções obtidas em relação à Tabela 5.2, alcançando o maior número de soluções viáveis, no caso, 23. Esse valor representa duas soluções a mais que o método BPLS, 4 a mais que o GLPK e uma a mais que o CBC. Em relação à qualidade

das soluções, observa-se que o CBC detém 14 melhores resultados, enquanto o GLPK detém 8 e o método proposto, 16.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste capítulo são apresentadas as conclusões na Seção 6.1, assim como os trabalhos futuros, na Seção 6.2.

6.1 Conclusões

Neste trabalho foi proposto um algoritmo híbrido, denominado HGVPRLB, para resolver problemas de programação linear binária. O HGVPRLB baseia-se nos procedimentos GRASP, VND, Propagação de restrições e *Local Branching*.

O algoritmo foi avaliado com relação à capacidade de alcançar uma solução viável de qualidade. Experimentos computacionais realizados em um conjunto de problemas-teste da biblioteca MIPLIB 2010 mostraram que o algoritmo proposto é superior a outros métodos de solução de problemas binários da literatura.

Quando o tempo de execução é limitado a 60 segundos, o algoritmo proposto encontra um maior número de soluções viáveis do que o algoritmo BPLS de Brito et al. (2014) e do que dois dos melhores resolvedores de código aberto de programação inteira disponíveis: CBC e GLPK. Além disso, o algoritmo HGVPRLB produz soluções de melhor qualidade em relação a esses métodos.

Por outro lado, quando o tempo de processamento é aumentado, o algoritmo HGVPRLB consegue um número ainda maior de soluções viáveis que os encontrados pelo método BPLS e também pelo resolvedor GLPK, e empata no número de soluções viáveis encontradas pelo resolvedor CBC. Já com relação à qualidade da solução final, observa-

se que ela é melhorada, sendo que, ao comparar os resultados obtidos com os de outros métodos, o HGVPRLB é o que apresenta o maior número de melhores soluções para o conjunto de problemas-teste.

Tendo-se por base os melhores resultados encontrados pelo algoritmo desenvolvido, observa-se sua superioridade em relação a outros da literatura, tanto com relação à capacidade de encontrar uma solução viável em um tempo restrito, quanto em relação à qualidade da solução final produzida. Entretanto, tendo-se por base as soluções medianas encontradas, verifica-se que o método proposto não consegue o mesmo desempenho, carecendo, assim de investigações futuras para que esse indicador seja melhorado.

6.2 Trabalhos Futuros

Como trabalhos futuros propõe-se:

- Desenvolver técnicas que explorem a redução do espaço de busca. Essas técnicas consistiriam em limitar os movimentos realizados no espaço de busca, de maneira a evitar movimentos que não sejam promissores;
- Fazer um estudo estatístico para definir os melhores parâmetros para o algoritmo desenvolvido, de forma a melhorar seu desempenho mediano;
- Adicionar ao conjunto de restrições, durante a fase de refinamento, um corte canônico, isto é, uma nova restrição formada pela função objetivo limitada pelo melhor valor da função objetivo encontrado até então ($z^\star$).

  $$\sum_{j \in N} c_j x_j \leq z^\star$$

  Espera-se que essa restrição reduza o tempo de execução, ao impedir a geração de soluções com qualidade inferior à atual.
- Modificar o algoritmo proposto para tratar problemas que envolvam variáveis inteiras;
- Estudar e implementar outras formas heurísticas nas fases de construção e refinamento, buscando melhores resultados.

Apêndice A

Publicações

A seguir são listados os trabalhos oriundos desta dissertação que foram aceitos para apresentação em eventos científicos e publicados nos seus anais:

1. **Título**: HPBL: um algoritmo híbrido para resolução de Problemas Binários
   Autores: Josiane da Costa Vieira Rezende, Rone Ilídio da Silva e Marcone Jamilson Freitas Souza
   Evento: XLVII Simpósio Brasileiro de Pesquisa Operacional – SBPO 2015
   Local: Porto de Galinhas, Brasil
   Período: 25 a 28 de agosto de 2015

2. **Título**: HGVPRLB: a hybrid algorithm for solving binary problems
   Autores: Josiane da Costa Vieira Rezende, Marcone Jamilson Freitas Souza e Rone Ilídio da Silva
   Evento: Trabalho aceito para apresentação na XLI Conferência Latino-americana en Informática – CLEI 2015
   Local: Arequipa, Peru
   Período: 19 a 23 de outubro de 2015

Referências Bibliográficas

Apt, K. R.: 1999, The essence of constraint propagation, *Theoretical computer science* **221**(1), 179–210.

Benoist, T., Estellon, B., Gardi, F., Megel, R. and Nouioua, K.: 2011, Localsolver 1. x: a black-box local-search solver for 0-1 programming, *4OR* **9**(3), 299–316.

Brito, S. S., Santos, H. G. and Santos, B. H. M.: 2014, A local search approach for binary programming: Feasibility search, *Lecture Notes in Computer Science* **8457**, 45–55. Proceedings of the Hybrid Metaheuristics 2014.

Carter, M. W. and Tovey, C. A.: 1992, When is the classroom assignment problem hard?, *Operations Research* **40**(1-supplement-1), S28–S39.

Dantzig, G. B.: 1951, *Activity Analysis of Production and Allocation*, John Wiley & Sons, New York, chapter Maximization of a Linear Function of Variables Subject to Linear Inequalities, pp. 339–347.

de Werra, D.: 1985, An introduction to timetabling, *European Journal of Operational Research* **19**(2), 151–162.

Feo, T. A. and Resende, M. G.: 1995, Greedy randomized adaptive search procedures, *Journal of global optimization* **6**(2), 109–133.

Fischetti, M. and Lodi, A.: 2003, Local branching, *Mathematical programming* **98**(1-3), 23–47.

Foundation, C.-O.: 2015, *CBC - COIN-OR Branch-and-Cut MIP Solver*, https://projects.coin-or.org/Cbc.

Fourier, J.: 1890, Second extrait, *Oeuvres* pp. 325–328.

Fréville, A.: 2004, The multidimensional 0–1 knapsack problem: An overview, *European Journal of Operational Research* **155**(1), 1–21.

Garfinkel, R. S. and Nemhauser, G. L.: 1972, *Integer programming*, Vol. 4, Wiley New York.

Geoffrion, A. M.: 1974, *Lagrangean relaxation for integer programming*, Springer.

GLPK: 2015, *GLPK - GNU Project*, http://www.gnu.org/software/glpk/.

Gomes, T. M.: 2014, *pRINS: uma matheurística para problemas binários*, Dissertação de mestrado, Programa de Pós-graduação em Ciência da Computação, Universidade Federal de Ouro Preto, Ouro Preto. Disponível em http://www.decom.ufop.br/pos/site_media/uploads_ppgcc/publicacao/gomes2014.pdf.

IBM: 2015, *CPLEX - IBM CPLEX Optimizer*, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R. E., Danna, E., Gamrath, G., Gleixner, A. M., Heinz, S. et al.: 2011, Miplib 2010, *Mathematical Programming Computation* **3**(2), 103–163.

lp solve: 2015, *lp solve reference guide*, http://lpsolve.sourceforge.net/5.5/.

MINTO: 2015, *MINTO - Mixed INTeger Optimizer*, http://coral.ie.lehigh.edu/ minto/.

Mladenović, N. and Hansen, P.: 1997, Variable neighborhood search, *Computers & Operations Research* **24**(11), 1097–1100.

Sandholm, T. and Shields, R.: 2006, Nogood learning for mixed integer programming, *Workshop on Hybrid Methods and Branching Rules in Combinatorial Optimization*, Montréal. Disponível em http://users.encs.concordia.ca/ chvatal/sandholm2.pdf.

SCIP: 2015, *Solving Constraint Integer Programs*, http://scip.zib.de/.

SYMPHONY: 2015, *SYMPHONY*, https://projects.coin-or.org/SYMPHONY.

Weisstein, E. W.: 2000, Bin-packing problem.