HPBL: um algoritmo híbrido para resolução de Problemas Binários

Josiane da Costa Vieira Rezende
Programa de Pós-Graduação em Ciência da Computação
Universidade Federal de Ouro Preto
josianecvieira@gmail.com

Rone Ilidio da Silva
Departamento de Tecnologia, Engenharia Civil, Computação e Humanidades
Universidade Federal de São João Del Rei
rone@ufsj.edu.br

Marcone Jamilson Freitas Souza
Programa de Pós-Graduação em Ciência da Computação
Universidade Federal de Ouro Preto
marcone@iceb.ufop.br

RESUMO

Neste trabalho propomos um método híbrido, nomeado HPBL, para resolver problemas binários genéricos. O método combina os procedimentos GRASP, *Variable Neighborhood Descent*, propagação de restrições, e cortes *Local branching*. O método desenvolvido foi aplicado a um conjunto de problemas binários da biblioteca MIPLIB 2010 para verificar sua capacidade de obter soluções viáveis e 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 encontradas quanto o de melhores soluções do conjunto. Além disso, o método proposto se mostrou superior a outro método da literatura, bem como a dois outros resolvedores de código aberto.

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

Área Principal: OC - Otimização Combinatória

ABSTRACT

In this paper we propose a hybrid method, named HPBL, for solving generic binary problems. The method combines the procedures GRASP, Variable Neighborhood Descent, Constraint Propagation and Local Branching Cuts. The proposed method was applied to a set of binary problems from MIPLIB 2010 library in order to verify its ability to get feasible solutions and with good quality varying the processing time. Computational experiments showed that when the processing time increases, the method increase the number of feasible solutions found as well as the number of best solutions of the set. Besides it, the proposed method outperforms another method of literature, as well as two other open source solvers.

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

Main Area: OC - Combinatorial Optimization

1. Introdução

Problemas de Programação Linear Binária (PLB) são casos especiais de Programação Linear (PL) em que todas as variáveis de decisão são binárias, ou seja, assumem o valor 1 quando a característica de interesse está presente e 0, caso contrário [Garfinkel (1972)].

Podem ser encontrados na literatura dois tipos de métodos para a solução de problemas de PL: exatos e heurísticos. O primeiro encontra a melhor solução, dita solução ótima, 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 inviável se o problema for da classe NP-difícil. Por outro lado, o segundo tipo procura uma solução sem a garantia de que ela seja ótima; entretanto, a solução pode ser encontrada em tempo hábil, e estar muito próxima da ótima.

Na literatura encontram-se diversos softwares que utilizam uma dessas estratégias para solucionar os problemas de programação linear, os 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)], Local Solver [Benoist (2011)] e Local Branching [Fischetti (2003)].

A literatura também apresenta diversas estratégias para a solução de PLBs, entre elas: 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, por sua vez, é definido um espaço de busca em um conjunto de restrições e procura-se encontrar soluções viáveis, isto é, pontos que pertencem ao espaço de busca e que satisfazem as restrições.

Neste trabalho é proposto um método híbrido, nomeado HPBL, para a resolução de problemas PLBs baseado na metaheurística GRASP [Feo (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* [Mladenović (1997)], a qual faz uso de cortes *Local Branching* [Fischetti (2003)]. Experimentos realizados com problemas binários da biblioteca MIPLIB 2010 [Koch (2011)] mostram que o HPBL é capaz de produzir, em tempo restrito, um número maior de soluções viáveis nessa biblioteca quando comparado com outro trabalho da literatura e com dois resolvedores de código aberto. Além disso, a qualidade das soluções produzidas pelo método proposto é de qualidade superior.

O restante deste artigo está organizado como segue. Na Seção 2 o problema binário é descrito. Na Seção 3 é feita uma breve revisão dos métodos de solução relacionados. Nas seções 4 e 5 o método proposto é apresentado e comparado com outros da literatura, respectivamente. Finalmente, na Seção 6 são apresentadas as conclusões deste trabalho.

2. Descrição Formal do Problema

Seja x um vetor n-dimensional de 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 $1 \leq j \leq n$. O modelo de programação linear binária pode, então, ser expresso por:

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

$$\text{Sujeito a:} \sum_{j=1}^{n} a_{ij} x_j \leq b_i\ \forall i = 1, \cdots, m \tag{2}$$

$$x_j \in \{0,1\}\ \ \forall j = 1, \cdots, n \tag{3}$$

3. Trabalhos Relacionados

Dada a complexidade inerente à resolução de um problema de programação linear inteira (PLI), são encontrados na literatura vários trabalhos que procuram resolvê-lo de forma mais eficiente com relação ao tempo de processamento. Dentre os diversos métodos utilizados para a resolução desses problemas, alguns dos principais são descritos a seguir.

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

Em [Fischetti (2003)] os autores propõem o uso do 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. A vizinhança é obtida por meio da introdução, no modelo de programação linear inteira, de desigualdades inválidas, chamadas de cortes *local branching*. 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.

Em [Sandholm (2006)] é descrita uma técnica de planos de corte globalmente válidos para algoritmos de busca 0-1 a partir de informações aprendidas com 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, nos experimentos realizados é relatado que isoladamente tais cortes não ajudam a PI, pois alguns dos planos de corte gerados são fracos.

Em [Benoist (2011)] é apresentado um resolvedor comercial que utiliza uma heurística de busca local para otimizar problemas binários lineares e não-lineares. O software, chamado de *LocalSolver*, alcança boas soluções em alguns problemas difíceis da biblioteca MIPLIB 2010. Entretanto, devido à natureza comercial do produto, os algoritmos implementados são apenas superficialmente descritos pelos autores. Por outro lado, existe uma grande necessidade de programas resolvedores de alta qualidade, porém livres, de modo que o usuário possa modificá-los de acordo com suas necessidades a fim de obter melhores resultados de forma mais rápida.

Em [Gomes (2014)] é desenvolvido um algoritmo híbrido, denominado pRINS, que explora técnicas de pré-processamento, procurando por um número ideal de fixações, visando a produzir sub-problemas de tamanho controlado. As variáveis fixadas são organizadas por um vetor de prioridades. Posteriormente, os problemas são criados e resolvidos de modo semelhante ao método *Variable Neighborhood Descent* até que um critério de parada seja satisfeito.

Em [Brito (2014)] é proposta uma abordagem *Local Search* para 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 visa a produção rápida de soluções viáveis, resultado que é comprovado nos experimentos.

O COIN-OR-CBC [Foundation (2015)] é um resolvedor de código aberto que apresenta bons resultados quando comparado aos demais de sua categoria. Ele é voltado à resolução de problemas lineares e inteiros. Entretanto, em PLI, soluções são dificilmente encontradas em tempo aceitável em casos práticos.

4. O Método Híbrido HPBL para Problemas Binários

O método híbrido proposto, nomeado HPBL, é um algoritmo **H**íbrido baseado em GRASP para resolver **P**roblemas **B**inários **L**ineares. Seu pseudocódigo é apresentado no Algoritmo 1.

No Algoritmo 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 como 0.01, o número máximo

Algoritmo 1: HPBL

Entrada: *LP, tempoLimite, β, γ, max_iter, t_vnd, θ*
Saída: $s^\star$
1 $f^\star \leftarrow +\infty$; $t \leftarrow 0$;
2 **enquanto** $t <$ *tempoLimite* **faça**
3 $s \leftarrow$ *contruaSolucao(LP, β, γ, max_iter, θ)*;
4 *tempoVND* $\leftarrow \min\{$*tempoRestante, t_vnd*$\}$;
5 $s \leftarrow$ *VND(LP, s, tempoVND*;
6 **se** $f(s) < f^\star$ **então**
7 $s^\star \leftarrow s$; $f^\star \leftarrow f(s^\star)$;
8 **fim**
9 Atualize t;
10 **fim**
11 **Retorne** $s^\star$;

de iterações *max_iter* a ser passado para o Algoritmo 2, o tempo de execução *t_vnd* utilizado no Algoritmo 6, o valor θ utilizado para liberar as variáveis fixas com valores 0 e 1 quando o resolvedor encontrar dificuldade na relaxação das variáveis.

Tal como um algoritmo clássico GRASP, há duas fases: uma de construção e outra de refinamento. Essas duas fases são aplicadas iterativamente até que o tempo limite seja atingido.

Na linha 3 do Algoritmo 1 uma solução é construída por meio do procedimento *contruaSolucao(LP,β, γ,max_iter,θ)*, definido na Seção 4.1. Na linha 4 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 5 e descrito na Seção 4.2, onde a solução contruída é refinada. Na linha 6 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 9 o tempo é atualizado.

Nas subseções seguintes o Algoritmo 1 é detalhado.

4.1. Fase Construtiva

O procedimento *contruaSolucao(LP,β, γ,max_iter,θ)* tem como objetivo construir uma solução inicial para o Algoritmo HPBL. Seu pseudocódigo está definido pelo Algoritmo 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 2 é acionado o resolvedor *CBC* aplicado ao problema *LP* relaxando as variáveis binárias, isto é, considerando-as no intervalo real $[0, 1]$. Nas linhas 4 a 8 é verificado se o resolvedor não encontrou uma solução viável para 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 9, se foi encontrada uma solução viável, esta é atribuída ao vetor v. Na linha 10 é chamado o procedimento *ConstruaRCL(LP,v,γ, β)* definido na Subseção 4.1.1, 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 4.1.2. 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 14 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 18. Na linha 20 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 22 são executadas até o número máximo de iterações *max_iter*. Na linha 24 a melhor solução encontrada é retornada.

Algoritmo 2: contruaSolucao

```
   Entrada: LP, β, γ, max_iter, θ
   Saída: s*
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      fim
9      v ← solução do problema LP relaxado;
10     j* ← ConstruaRCL(LP, v, β, γ);
11     s_j* ← 1;
12     Fixe s_j* = 1 em LP;
13     I ← PropagacaodeRestricoes(LP,l,u,C);
14     se I = inviável então
15         Libere s_j* em s;   Libere s_j* em LP;
16     fim
17     senão
18         Fixe implicações I em s e em LP;
19     fim
20     se f(s) for melhor que f(s*) então
21         s* ← s;
22     fim
23 fim
24 Retorne s*;
```

4.1.1. 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 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 3: ConstruaRCL

```
   Entrada: LP, v, β, γ
   Saída: j*
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 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 qualidade. Uma vez que a escolha de β influencia no tamanho da RCL e, consequentemente, na qualidade das soluções obtidas nesta fase, à medida que o tamanho da RCL aumenta, em média diminui a qualidade da solução, mas em contrapartida aumenta a diversidade. A *RCL* é formada pelos índices j das variáveis s_j que satisfazem a Eq. (4):

$$s_j \geq max - \beta \times (max - min) \tag{4}$$

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 Eq. (4).

A partir desta lista de candidatos, na linha 5 é selecionado de forma aleatória um índice de uma variável que será o retorno do método.

4.1.2. 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 por restrições, cada restrição i é reescrita na forma da Eq. (5), na qual N é o conjunto de variáveis.

$$\sum_{j \in N} a_{ij} s_j \leq b_i \tag{5}$$

Assim, seja a medida U_{ij} determinada pela Eq. (6):

$$U_{ij^\star} = b_i - \left(\sum_{j \in N_i^+, j \neq j^\star} |a_{ij}|\, l_j + \sum_{j \in N_i^-, j \neq j^\star} |a_{ij}|\, u_j \right) \tag{6}$$

em que $N_i^+ = \{j \in N : a_{ij} \geq 0\}$ 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 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: 1) O valor binário que a variável s_j deve ser fixada, ou então, 2) a indefinição do valor a ser fixado para essa variável. Assim, este procedimento retorna o par *status,limite*, sendo 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 e 5 apresentam os pseudocódigos baseados na implementação do algoritmo de [Sandholm (2006)] para detectar conflitos e implicações.

No Algoritmo 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 onde 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 uma variável, cujo nome é definido como *novas_implicacoes*, é definido como *falso*. C é o conjunto das restrições em que a $j^\star$-ésima variável fixada aparece. Nas linhas 5 a 16 é realizado um laço iterativo em cada uma dessas restrições. Nas linhas 6 a 16, para cada uma das variáveis dessa 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 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 17, se não houve fixações, a função retorna o *status* SEM_IMPLICACOES. Nas linhas 20 a 25 para toda fixação realizada anteriormente adicionada ao conjunto I, os limites l_j e l_u destas variáveis são atualizados. Cada restrição em que essas variáveis aparecem são adicionadas ao conjunto C. Nas linhas 26 a 30 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 32 do Algoritmo 4, o procedimento *PropagacaodeRestricoes(LP,l,u,C)* retorna o par *status* e o conjunto I de fixações realizadas.

Algoritmo 4: PropagacaodeRestricoes

Entrada: *LP, l, u, C*
Saída: (*status, I*)

status ⟵ SEM_IMPLICACOES;
$I \longleftarrow \emptyset$;
repita
  novas_implicacoes ← *falso*;
  para todo *restrição c em C* **faça**
    para todo *variável* $s_{j^\star}$ *não fixada em c* **faça**
      (*limite,status*) ⟵ *avaliaInviabilidade(LP,l,u,c,*$j^\star$*)*;
      se *status = FIXA* **então**
        $I \longleftarrow I \cup \{(j^\star, limite)\}$;
        novas_implicacoes ← *verdadeiro*;
      fim
      senão se *status = CONFLITO* **então**
        return (*CONFLITO*, *I*);
      fim
    fim
  fim
  se $|I| = \emptyset$ **então**
    return (SEM_IMPLICACOES, I);
  fim
  para todo $(j, limite) \in I$ **faça**
    $l_j \leftarrow u_j \leftarrow limite$;
    para todo *c em C que contém uma ou mais variáveis fixadas* **faça**
      $C \longleftarrow C \cup \{c\}$;
    fim
  fim
  para todo *c em C* **faça**
    se *todas as variáveis da restrição c estão fixadas* **então**
      $C \longleftarrow C \setminus \{c\}$;
    fim
  fim
até $(C \neq 0)$ *e novas_implicacoes = verdadeiro* ;
Retorne *(status, I)*;

Algoritmo 5: avaliaInviabilidade

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

$U_{ij^\star}$ ⟵ Calcular segundo Eq. (6);
se $j^\star \in N_i^+$ **então**
  se $U_{ij^\star} < 0$ **então**
    limite ⟵ `NULO`; *status* ⟵ `CONFLITO`;
  fim
  senão se $a_{ij^\star} > U_{ij^\star}$ **então**
    limite ⟵ 0; *status* ⟵ `FIXA`;
  fim
  senão
    limite ⟵ `NULO`; *status* ⟵ `SEM_IMPLICACOES`;
  fim
fim
se $U_{ij^\star} \in N_i^-$ **então**
  se $a_{ij^\star} > U_{ij^\star}$ **então**
    limite ⟵ `NULO`; *status* ⟵ `CONFLITO`;
  fim
  senão se $U_{ij^\star} < 0$ **então**
    limite ⟵ 1; *status* ⟵ `FIXA`;
  fim
  senão
    limite ⟵ `NULO`; *status* ⟵ `SEM_IMPLICACOES`;
  fim
fim
Retorne (*status,limite*);

O Algoritmo 5 tem como dados de entrada o problema binário *LP*, os limites superior e inferior l e u de cada variável, a restrição i onde está contida a variável $s_{j^\star}$, e o índice $j^\star$ dessa variável a ser verificada.

4.2. Fase de refinamento - VND

Nesta fase, a solução proveniente da fase de construção é refinada durante um tempo determinado por um procedimento *Variable Neighborhood Descent* – VND [Mladenović (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 (7) a seguir:

$$\lceil 0.90 \times n_1 \rceil \leq \sum_{j \in N1} s_j \leq \lceil 0.95 \times n_1 \rceil \tag{7}$$

Desta maneira, a princípio busca-se 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, procura-se fixar um número grande de variáveis, mas com um grau de liberdade razoável, possibilitando ao resolvedor encontrar melhores soluções.

O Algoritmo 6 mostra o pseudocódigo do procedimento VND usado na fase de busca local do método HPBL implementado para gerar os cortes *Local Branching*.

Algoritmo 6: VND

Entrada: *LP, s, tempoVND*
Saída: $s^\star$
1 *nLvizinhanca[]* $\leftarrow \{0.95, 0.90, 0.85, 0.80, 0.75, 0.70\}$;
2 $t \leftarrow 0$; $k \leftarrow 0$;
3 **repita**
4 $n_1 \leftarrow$ número de variáveis de *s* fixadas no valor 1;
5 $\alpha_l \leftarrow$ *nLvizinhanca[k]*; $\alpha_u \leftarrow$ *nLvizinhanca[k+1]*;
6 $coef_{sup} \leftarrow \lceil n_1 \times \alpha_l \rceil$; $coef_{inf} \leftarrow \lceil n_1 \times \alpha_u \rceil$;
7 *tempoLB* $\leftarrow$ *tempoVND* $-$ t;
8 $s \leftarrow$ *LocalBranching (LP, tempoLB,* $coef_{sup}$, $coef_{inf}$, *s)* ;
9 **se** $f(s)$ *é melhor que* $f(s')$ **então**
10 $s^\star \leftarrow s$; $k \leftarrow 0$;
11 **fim**
12 **senão**
13 k++;
14 **fim**
15 **até** $t >$ *tempoVND* ;
16 **Retorne** $s^\star$;

Nesse Algoritmo são dados de entrada o problema linear *LP*, a solução *s*, e o tempo de execução *tempoVND*. É importante ressaltar que na linha 7 é calculado o tempo disponível para a execução do procedimento *LocalBranching (LP,tempoLB,*$coef_{sup}$*,*$coef_{inf}$*,s)*, apresentado na Seção

4.2.1. 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)*. O Algoritmo retorna na linha 16 a melhor solução encontrada.

4.2.1. *Local Branching*

O procedimento aqui proposto é baseado no trabalho [Fischetti (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.

A cada restrição não satisfeita do problema na solução corrente são adicionadas ou subtraídas, conforme o caso, variáveis de folga de forma a torná-la atendida. Tais variáveis são acrescentadas também à função objetivo do problema e recebem coeficientes altos em relação aos demais. Com tais variáveis, junto com as variáveis originais do problema, são criadas, então, duas novas restrições que cumprem a função do *local branching* na forma da Eq. (7), em que no lugar dos valores 0.90 e 0.95 são colocados dois valores k_1 e k_2 contíguos dentro do conjunto de vizinhanças *nLvizinhanca* = $\{0.95, 0.90, \cdots, 0.75, 0.70\}$. O Algoritmo 7 apresenta o pseudocódigo do procedimento de busca local *LocalBranching* utilizado para resolver o problema binário descrito.

Algoritmo 7: *LocalBranching*

Entrada: *LP, t,* $coef_{sup}$, $coef_{inf}$, *s*
Saída: $s^\star$
1 **se** *solução é inviável* **então**
2 | *LP* ← variáveis de folga;
3 **fim**
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 $s^\star$ ← otimiza();
9 **Retorne** $s^\star$;

O Algoritmo 7 recebe como dados de entrada o problema linear *LP* e um tempo *tempoLB* de execução. No que se refere à linha 1 é verificada se a solução é inviável, e caso afirmativo, na linha 2 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 (2003)]. Por se tratarem de problemas de minimização, com o objetivo de as variáveis de folga possuirem 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. 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 não houver dificuldade de ser encontrada a solução, o procedimento retorna a solução obtida pelo resolvedor *CBC*.

5. Experimentos Computacionais

O método proposto HPBL, descrito pelo Algoritmo 1, foi escrito na linguagem C utilizando-se para a leitura dos 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 3,4 GHz Intel Core i7-3770R com 16 GB de RAM executando em um sistema operacional openSUSE 12.3 Linux.

Para os experimentos foram utilizados 32 problemas binários da biblioteca MIPLIB 2010 [Koch (2011)]. A biblioteca MIPLIB foi criada em 1992 e obteve sua última atualização em 2010. Esta biblioteca foi montada a partir da coletânia de problemas reais obtidos da indústria ou da comunidade acadêmica. Os problemas-teste contidos nessa biblioteca têm sido largamente usados para comparar o desempenho de resolvedores de programação inteira.

O método HPBL 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.

Após uma bateria preliminar de testes os parâmetros do método HPBL 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 era 60 segundos, então t_vnd assumia o valor 30 segundos, e quando o tempo total de execução do método era 300 segundos, t_vnd = 150 segundos. O valor *max_iter* inicia com 10 quando o tempo limite é 60 segundos e inicia com 20 quando o tempo limite é 300 segundos, e a cada iteração de busca local ele é multiplicado por 4.

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 (2014)]. Em todas as comparações o critério de parada foi o mesmo.

Tabela 1: Valor da melhor solução encontrada em 60 segundos

Prob.-Teste	CBC	GLPK	BPLS	HPBL	
				Melhor	Média
acc-tight5					
air04	**56137**	57830	x	56138	56549,77
bab5					
bley xl1					
bnatt350					
cov1075	**20**	**20**	x	**20**	20,03
eil33.2	993,61	**934**	x	987,67	988,00
eilB101	1529,89	1374,96	x	**1326,05**	1343,07
ex9					
iis-100-0-cov	**29**	30	x	**29**	**29**
iis-bupa-cov	40	40	x	**36**	36,23
iis-pima-cov	36	36	x	**33**	33,93
m100n500k4r1	**-24**	**-24**	x	**-24**	-23,70
macrophage	392	495	x	**386**	468,17
mine-166-5	**-553252300**	-531532749,40	x	-3459148,24	680296,82
mine-90-10			x	**-13321729,51**	719767,38
mspp16					
n3div36	**135000**		x	165200	172606,67
n3seq24			x	**63000**	35733,33
neos-1109824	**378**	**378**		**378**	317,10
neos-1337307	**-202191**	-201708		-202038	-6734,60
neos18	**16**	22	x	**16**	16,07
neos-849702					
netdiversion					
ns1688347					
opm2-z7-s2	-8239	-9430	x	**-9841**	-1038,60
reblock67	**-34388335**		x	-2423910,13	-449419,31
rmine6	**-456,94**	-455	x	-454,99	-74,61
sp98ic	451003580	520753842,70	x	**450568218,08**	454242165,50
tanglegram1			x		
tanglegram2	515	**443**	x	**443**	**443**
vpphard					
Nº Soluções Viáveis	19	17	20	21	21
Nº Melhores Soluções	11	5		14	2

Nas tabelas 1 e 2 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 método homônimo de [Brito (2014)]. A coluna "HPBL" é subdividida em duas colunas, sendo que na primeira é reportado o melhor resultado encontrado em 30 execuções desse método, enquanto na segunda consta o resultado médio das execuções. Como em [Brito (2014)] não é 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.

Pela Tabela 1 observa-se que o método 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 método proposto, 14.

Tabela 2: Valor da melhor solução encontrada em 300 segundos

Prob.-Teste	CBC	GLPK	BPLS	HPBL	
				Melhor	Média
acc-tight5					
air04	**56137**	56143	x	**56137**	56408,97
bab5	-103368,24			**-105984,96**	-78130,51
bley xl1					
bnatt350					
cov1075	**20**	**20**	x	**20**	**20**
eil33.2	**934**	**934**	x	**934**	934,01
eilB101	1227,71	1310,27	x	**1216,92**	1270,05
ex9					32,40
iis-100-0-cov	**29**	**29**	x	**29**	**29**
iis-bupa-cov	37	39	x	**36**	36,10
iis-pima-cov	34	36	x	**33**	33,87
m100n500k4r1	**-24**	**-24**	x	**-24**	-23,70
macrophage	**378**	481	x	386	393,10
mine-166-5	**-566395707,87083**	-531532749,40	x	-3459148,24	400413,19
mine-90-10	-783742624,048894		x	**-13321729,51**	-287284,39
mspp16					
n3div36	**131400**	179600	x	140400	164386,67
n3seq24			x	**63000**	35226,67
neos-1109824	**378**	**378**	x	**378**	315,13
neos-1337307	**-202319**	-201708		-202038	-6734,60
neos18	**16**	20	x	**16**	**16**
neos-849702					
netdiversion					
ns1688347					
opm2-z7-s2	-10145	-10205	x	**-10257**	-9996
reblock67	-34630648	**-15541889,16**	x	-2423910,13	-449419,31
rmine6	**-457,01**	-455	x	-454,99	-74,61
sp98ic	451968910	**487333102,70**	x	449213315,20	450379464,59
tanglegram1			x		
tanglegram2	**443**	**443**	x	**443**	**443**
vpphard	**66**				
Nº Soluções Viáveis	22	19	21	22	22
Nº Melhores Soluções	14	8		15	4

Por outro lado, pela Tabela 2 observa-se que o método proposto foi capaz de melhorar a qualidade das soluções obtidas em relação à Tabela 1, alcançando, assim como o CBC, o maior número de soluções viáveis encontradas, no caso, 22. Esse valor representa duas soluções a mais que o método BPLS, e 3 a mais que o GLPK. Em relação à qualidade das soluções observa-se que o CBC detém 14 melhores resultados, enquanto o GLPK detém 8 e o método proposto 15.

6. Conclusões e Trabalhos Futuros

Neste trabalho foi proposto um método híbrido para resolver problemas de programação linear binária, denominado HPBL.

Experimentos computacionais realizados em um conjunto de problemas-teste da biblioteca MIPLIB 2010 mostraram que o método proposto é superior a outros métodos de solução de problemas binários da literatura. Quando o tempo de execução é limitado a 60 segundos, o método proposto encontra um maior número de soluções viáveis do que o método BPLS de [Brito (2014)] e do que dois dos melhores resolvedores de código aberto de programação inteira disponíveis: CBC e GLPK. Além disso, o método HPBL produz soluções de melhor qualidade em relação a esses métodos. Por outro lado, quando o tempo de processamento é aumentado, o método HPBL 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 HPBL é o que apresenta o maior número de melhores soluções para o conjunto de problemas-teste.

Esses fatos mostram a superioridade do método proposto em relação a outros da literatura no que diz respeito tanto com relação à capacidade de encontrar uma solução viável em um tempo restrito quanto em relação à qualidade da solução final produzida.

Agradecimentos

Os autores agradecem às agências FAPEMIG, CNPq e CAPES, bem como ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Ouro Preto pelo apoio.

Referências

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

Benoist, T., Estellon, B., Gardi, F., Megel, R., e 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., e Santos, B. H. M. (2014). A local search approach for binary programming: Feasibility search. *Lecture Notes in Computer Science*, 8457:45–55.

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

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

Fischetti, M. e 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*, pages 325–328.

Garfinkel, R. S. e Nemhauser, G. L. (1972). *Integer programming*, volume 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.

IBM (2015). *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. e Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Sandholm, T. e Shields, R. (2006). Nogood learning for mixed integer programming. In *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.