UM NOVO ALGORITMO HÍBRIDO PARA RESOLUÇÃO DE PROBLEMAS BINÁRIOS

Josiane da Costa Vieira Rezende, Marcone Jamilson Freitas Souza
Programa de Pós-graduação em Ciência da Computação
Universidade Federal de Ouro Preto (UFOP)
CEP: 35.400-000 – Ouro Preto – MG – Brasil
josianecvieira@gmail.com, marcone@iceb.ufop

Luciano Perdigão Cota
Programa de Pós-graduação em Engenharia Elétrica
Universidade Federal de Minas Gerais (UFMG)
CEP: 31.270-901 – Belo Horizonte – MG – Brasil
lucianoufop@gmail.com

RESUMO

Neste trabalho é proposto um novo algoritmo híbrido, nomeado HGVPRLB-CC, para resolver problemas genéricos de programação linear binária. Ele é guiado pela metaheurística *Greedy Randomized Adaptive Search Procedures* – GRASP. Para gerar uma solução inicial, ele mescla relaxamentos de programação linear e programação por propagação de restrições. Para refinar a solução construída é realizada uma busca local, guiada pela heurística *Variable Neighborhood Descent* – VND, a qual, por sua vez, faz uso de 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 de problemas-teste. Além disso, o método proposto se mostrou superior à sua versão prévia e a outro algoritmo da literatura, bem como aos resolvedores GLPK e CBC.

PALAVRAS CHAVE. Problemas binários, Heurística, GRASP, *Variable Neighborhood Descent*, *Local Branching*, Programação por Propagação de Restrições.

ABSTRACT

In this work it is proposed a new hybrid algorithm, so-called HGVPRLB-CC, for solving binary problems. It is guided by the Greedy Randomized Adaptive Search Procedures metaheuristic. In order to generate an initial solution, it gathers linear programming relaxations and constraint programming. To refine the built solution, it is realized a local search, guided by the Variable Neighborhood Descent heuristic, which, in turn, uses Local Branching cuts. The algorithm was tested in a set of binary problems from MIPLIB 2010 in order to check both its ability to obtain feasible solutions as its ability to improve the value of these solutions varying the processing time. Computational experiments showed that when the processing time increases, the algorithm can increase the number of feasible solutions found in the set of instances as well the quality of the solutions. Besides it, the proposed algorithm outperforms its previous version and other algorithm of literature, as well as the solvers GLPK and CBC.

KEYWORDS. Binary Problems, Heuristic, GRASP, *Variable Neighborhood Descent*, *Local Branching*, Constraint Programming.

1. Introdução

Este trabalho tem seu foco no desenvolvimento de métodos para solução de problemas binários genéricos. Problemas binários aparecem em vários contextos, como por exemplo, na programação de horários (*timetabling*) [da Fonseca (2014)], na alocação de aulas a salas (*Classroom Assignment Problem*) [Subramanian (2011)] e na alocação de tripulantes a aeronaves (*Airline Crew Scheduling Problems*) [Toffolo (2006)], entre outros.

Problemas dessa natureza podem ser resolvidos por métodos de programação matemática, dito exatos, ou por métodos heurísticos. Tendo em vista a dificuldade de solução de grande parte dos problemas binários, que são da classe NP-difícil, grande esforço é feito no sentido de se desenvolver algoritmos heurísticos que resolvam eficientemente tal classe de problemas.

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, dos chamados 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 sugerida uma técnica de planos de corte globalmente válidos para algoritmos de busca 0-1 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. Os experimentos realizados pelos autores mostraram 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. Tal software, chamado de *LocalSolver*, alcança boas soluções em alguns problemas da biblioteca MIPLIB 2010 em casos classificados como difícil. Entretanto, devido à natureza comercial do produto, os algoritmos implementados são apenas superficialmente descritos pelos autores.

Em [Gomes (2014)] é desenvolvido o algoritmo pRINS, que explora técnicas de pré-processamento, procurando por um número ideal de fixações, de forma 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*. 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 realizados.

Em [Rezende (2015)] é proposto um método híbrido para a resolução de problemas binários que mescla as técnicas GRASP, *Variable Neighborhood Descent*, programação por propagação de restrições, e cortes *Local branching*. Esse método se mostrou superior ao de [Brito (2014)], bem como a dois outros resolvedores de código aberto da literatura.

Neste trabalho é proposto um novo algoritmo híbrido, nomeado HGVPRLB-CC, para resolver problemas binários baseado na metaheurística *Greedy Randomized Adaptive Search Procedures* – GRASP [Feo (1995)]. Em sua fase construtiva ele mescla relaxamentos de programação linear e programação por propagação de restrições para a obtenção de uma solução inicial. Em seguida, é realizada uma busca local, guiada por uma heurística *Variable Neighborhood Descent* – VND [Mladenović (1997)], a qual, por sua vez, faz uso de cortes *Local Branching* [Fischetti (2003)].

Este trabalho representa um aperfeiçoamento do trabalho de [Rezende (2015)], ao introduzir no algoritmo desses autores cortes canônicos durante a fase de refinamento e fazer uma análise estatística dos resultados. Os experimentos computacionais realizados mostram que, apesar de não haver diferença estatística entre essa nova versão do algoritmo e a versão anterior, ele produz

soluções finais de melhor qualidade quando comparado com a versão anterior e com os resolvedores de código aberto GLPK e CBC. Além disso, quando a comparação de resultados é pelo número de soluções viáveis encontradas em um dado tempo de processamento, o algoritmo proposto supera tanto o algoritmo anterior quanto outro algoritmo da literatura, bem como os dois resolvedores mencionados anteriormente.

O restante deste artigo está organizado como segue. Na seção 2 o algoritmo proposto é apresentado. Na seção 3 são descritos e analisados os resultados dos experimentos computacionais realizados. Finalmente, na seção 4 são apresentadas as conclusões deste trabalho.

2. O Algoritmo Híbrido HGVPRLB-CC

O algoritmo híbrido proposto para resolver Problemas Binários genéricos, chamado de HGVPRLB-CC, combina os procedimentos GRASP, VND, Programação por Propagação de Restrições e *Local Branching*. Seu pseudocódigo é apresentado no Algoritmo 1.

Algoritmo 1: HGVPRLB-CC

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

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 em 0.01, o número máximo de iterações *max_iter*, o tempo de execução *t_vnd* e o valor θ utilizado para liberar as variáveis fixas com valores 0 e 1 quando o resolvedor encontrar dificuldade na resolução do problema.

O algoritmo tem duas fases, construção e refinamento, que são aplicadas iterativamente até que um tempo limite seja atingido. Na linha 4 do Algoritmo 1 uma solução é construída por meio do procedimento *construaSolucao*(). 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 2.1, onde 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. Sendo a resposta afirmativa, $s^\star$ é atualizada. Na linha 11 o tempo é atualizado.

Como o algoritmo proposto difere do de [Rezende (2015)] apenas na fase de refinamento, mais especificamente, na aplicação do *Local Branching*, então apenas essa fase do Algoritmo 1 é detalhada no presente trabalho.

2.1. Fase de refinamento - VND

A fase de refinamento é feita pela aplicação por um tempo determinado do procedimento *Variable Neighborhood Descent* – VND [Mladenović (1997)]. 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 a aplicação de todas as estruturas de vizinhança sem se conseguir uma melhora na solução corrente.

Neste trabalho considera-se que cada vizinhança é definida pela fixação no valor 1 de um certo percentual de variáveis da solução oriunda da fase de construção com valor igual a 1. As

vizinhanças diferem entre si por 5% no número de variáveis a serem fixadas, sendo esse percentual variável no intervalo 95% a 70%. Assim, a primeira vizinhança contém 95% das variáveis fixadas no valor 1, a segunda 90% e assim por diante.

Para exemplificar, sejam $N1 = \{j : s_j = 1\}$ na fase de construção e $n_1 = |N1|$. Se a vizinhança em análise for a primeira, então 95% das variáveis devem ser fixadas no valor 1, sendo para tanto adicionadas duas restrições ao problema LP, dadas pelas equações (1):

$$\lceil 0.90 \times n_1 \rceil \leq \sum_{j \in N1} s_j \leq \lceil 0.95 \times n_1 \rceil \quad (1)$$

Desta maneira, a princípio se busca apertar a solução fixando-se 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 número menor de variáveis fixadas. Assim, dá-se ao resolvedor gradativamente maior chance de encontrar melhores soluções.

O Algoritmo 2 mostra o pseudocódigo do procedimento VND usado na fase de busca local do Algoritmo Híbrido HGVPRLB-CC implementado para gerar os cortes *Local Branching*.

Algoritmo 2: 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] ;
6 $\alpha_u \leftarrow$ *nLvizinhanca*[$k+1$];
7 $coef_{sup} \leftarrow \lceil n_1 \times \alpha_l \rceil$;
8 $coef_{inf} \leftarrow \lceil n_1 \times \alpha_u \rceil$;
9 *tempoLB* $\leftarrow$ *tempoVND* $-$ t;
10 $s \leftarrow$ *LocalBranching (LP, tempoLB,* $coef_{sup}$, $coef_{inf}$, *s)* ;
11 **se** $f(s)$ *é melhor que* $f(s')$ **então**
12 $s^\star \leftarrow s$;
13 $k \leftarrow 0$;
14 **fim**
15 **senão**
16 k++;
17 **fim**
18 **até** $t >$ *tempoVND* ;
19 **Retorne** $s^\star$;

No Algoritmo 2 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, nomeado *nLvizinhanca*[].

Na linha 2 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 variável *n* recebe o número de variáveis que na solução possuem valor igual a 1. Estas serão as variáveis utilizadas para os cortes *Local Branching*. Esses cortes possuem limites inferior e superior que serão definidos a partir de valores da estrutura de vizinhança corrente.

Na linha 5 α_l recebe o valor da posição do vetor *nLvizinhanca*[], referente à posição atual de *k*. Na linha 6 α_u recebe o valor da posição do vetor *nLvizinhanca*[], referente à posição k + 1.

Nas linhas 7 e 8 são calculados os valores de $coef_{sup}$ e $coef_{inf}$, de acordo com o valor de n, 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 9 é calculado o tempo disponível para a execução do procedimento *LocalBranching (LP,tempoLB,*$coef_{sup}$,$coef_{inf}$,*s)*, apresentado em detalhes na Seção 2.2. Assim, a variável *tempoLB* recebe o tempo de entrada *tempoVND* diminuído do tempo corrente *t*. Esse tempo pode ser

gasto totalmente ou parcialmente pelo procedimento *LocalBranching*().

Na linha 11 é verificado se a solução melhorou, e em caso afirmativo, a nova solução é armazenada, e a variável k recebe o valor 0. Se isto não ocorrer, na linha 16 o valor de k é incrementado em uma unidade. As linhas 3 a 18 são repetidas enquanto houver disponibilidade de tempo. Na linha 19 é retornada a melhor solução encontrada.

Na próxima Seção é detalhado o procedimento *LocalBranching*(), linha 10 do Alg. 2.

2.2. *Local Branching*

O procedimento aqui utilizado 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. Neste trabalho 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 (2):

$$a_{i,n+i} = b_i - \sum_{j=1}^{n} a_{ij}s_j \tag{2}$$

resultando, assim, na nova restrição i válida, dada pela Equação (3):

$$\sum_{j=1}^{n} a_{ij}s_j - a_{i,n+i}s_{n+i} \leq b_i \tag{3}$$

De forma análoga, a cada restrição i não satisfeita do tipo $\geq$ é adicionada ao lado esquerdo da restrição uma variável s_{j+1}, cujo coeficiente $a_{i,n+1}$ é dado pela Equação (4):

$$a_{i,n+i} = \sum_{j=1}^{n} a_{ij}s_j - b_i \tag{4}$$

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. Tais variáveis também são acrescentadas à função objetivo do problema e recebem coeficientes positivos altos em relação aos demais. Tornadas satisfeitas todas as restrições, a seguir são criadas duas novas restrições, definidas pela Equação (5). Essas restrições são adicionadas ao modelo original de programação linear inteira e cumprem a função do *local branching*.

$$\lceil \alpha_l \times n_1 \rceil \leq \sum_{j \in N1} s_j \leq \lceil \alpha_u \times n_1 \rceil \tag{5}$$

Na Equação (5), α_l e α_u são valores consecutivos pertencentes ao conjunto de vizinhanças *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*.

Além dos cortes *Local Branching*, são introduzidos cortes canônicos [Balas (1972)] no algoritmo, fazendo com que o resolvedor busque somente por soluções que sejam melhores que a solução atual, ou seja, em um problema de minimização é adicionada uma nova restrição na qual o valor da função objetivo buscada é menor ou igual ao valor da solução atual decrescida do valor 1 (quando os coeficientes da função objetivo forem inteiros), ou menor ou igual ao valor da solução atual decrescida de um valor ξ suficientemente pequeno (no caso de os coeficientes forem reais). Em caso de problemas de maximização o oposto acontece, sendo adicionada uma restrição na qual o valor da função objetivo seja maior ou igual ao valor da solução atual acrescido do valor 1 ou ξ de forma análoga ao caso anterior. A introdução desses cortes é um aperfeiçoamento do algoritmo de [Rezende (2015)].

O Algoritmo 3 apresenta o pseudocódigo do procedimento de busca local *LocalBranching* utilizado para resolver o problema binário descrito.

Algoritmo 3: *LocalBranching*

```
   Entrada: LP, t, coef_sup, coef_inf, s
   Saída: s*
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  LP ← adiciona corte canônico;
8  define limite de tempo t para otimização;
9  s* ← otimiza();
10 Retorne s*;
```

O Algoritmo 3 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 adicionado o corte canônico com o objetivo de buscar soluções melhores que a atual. Na linha 8 é definido o tempo de otimização. Na linha 9 o problema é otimizado, e finalmente, na linha 10, 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*.

3. Experimentos Computacionais

O Algoritmo HGVPRLB-CC foi escrito na linguagem C e compilado no GCC, versão 4.6.3, utilizando-se para a leitura de problemas-teste o resolvedor de código aberto COIN-OR. Os experimentos foram realizados em um computador Intel Core i7-3770R com 16 GB de RAM e 3,4 GHz de clock, sob sistema operacional openSUSE 12.3 Linux. Para testar o algoritmo foram utilizados 32 problemas binários da biblioteca MIPLIB 2010 [Koch (2011)].

O algoritmo proposto foi testado com relação à sua capacidade de encontrar uma solução viável de qualidade variando-se o tempo de processamento. Para 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 Algoritmo HGVPRLB-CC 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 algoritmo era 300 segundos, t_vnd = 150 segundos. O valor *max_iter* inicia com 10 quando o temço 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.

Diferentemente de [Rezende (2015)], que utilizou a versão *Trunk*, revisão 2166, do *CBC*, aplicou-se a revisão 2277. Em vista disso, os resultados do algoritmo de [Rezende (2015)] foram refeitos nessa nova versão do resolvedor, de forma a permitir uma comparação mais justa entre os métodos, já que há diferença entre os resultados de [Rezende (2015)] e os novos resultados relatados.

Os resultados dos experimentos realizados foram comparados com os de dois dos melhores resolvedores de código aberto para programação inteira: CBC e GLPK, e também com os métodos BPLS de [Brito (2014)] e HGVPRLB de [Rezende (2015)]. Em todas as comparações o critério de parada foi o mesmo.

Para verificar se existe diferença estatística entre os resultados dos algoritmos HGVPRLB de [Rezende (2015)] e do novo algoritmo HGVPRLB-CC proposto foi aplicado o teste ANOVA [Montgomery (2007)] com 95% de grau de confiança ($threshold = 0.05$).

Na Figura 1 é apresentado um gráfico box plot dos resultados para 60 segundos. No teste encontrou-se $p = 0.26$; como $p > threshold$, tem-se um indicativo forte que não existe diferença estatística entre os resultados.

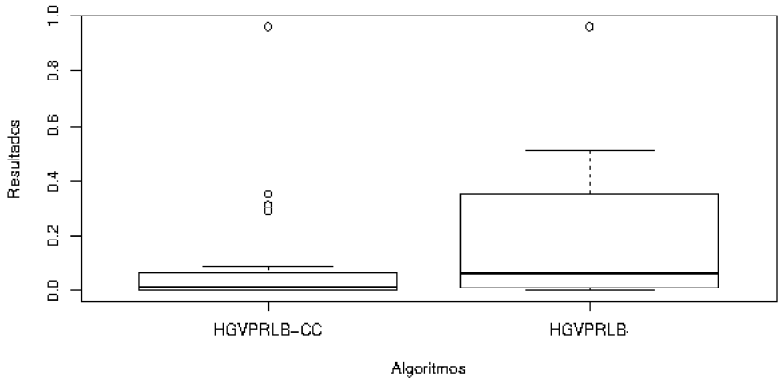

Figura 1: Box plot dos resultados com tempo limite de 60 segundos

A premissa de normalidade dos dados foi verificada aplicando-se o teste Shapiro-Wilk [Shapiro (1965)] e a análise gráfica (vide Figura 2).

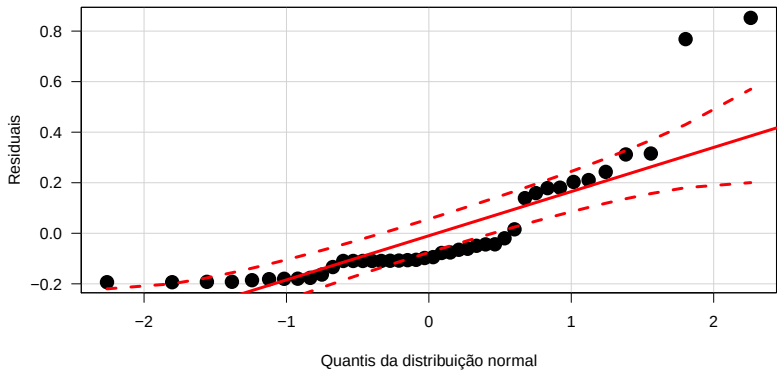

Figura 2: Gráfico de normalidade dos resultados com tempo limite de 60 segundos

Com 95% de grau de confiança encontrou-se $p = 2.27 \times 10^{-7}$ e $W = 0.73406$, mostrando que os dados não seguem uma distribuição normal. Por outro lado, a análise gráfica apresentada na Figura 2 sugere que os dados seguem uma distribuição normal, apesar da existência de alguns *outliers*. Sabe-se que o teste ANOVA é robusto quanto a violações moderadas de normalidade, desde que o tamanho da amostra seja grande o suficiente. Como neste trabalho cada algoritmo foi executado 30 vezes para cada problema-teste, pode-se concluir que o resultado do teste ANOVA é confiável.

De maneira análoga, foi aplicado o teste ANOVA com 95% de grau de confiança para verificar se existe diferença estatística entre os resultados do algoritmo HGVPRLB e do novo algoritmo proposto no tempo limite de 300 segundos. Nesses experimentos foram analisados somente os casos em que os dois algoritmos convergiram. Um gráfico box plot dos resultados para o tempo limite de 300 segundos é apresentado na Figura 3. Para esse teste encontrou-se $p = 0.0509$; como $p > threshold$, existe um indicativo forte que não existe diferença estatística entre os resultados.

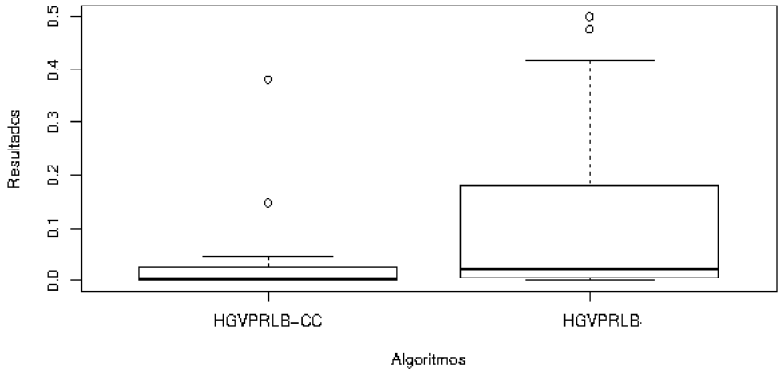

Figura 3: Box plot dos resultados com tempo limite de 300 segundos

Tal como no caso anterior, foi aplicado o teste Shapiro-Wilk e feita a análise gráfica (vide Figura 4) para verificar se os dados seguem uma distribuição normal. O teste indicou, com 95% de grau de confiança, que os dados não seguem uma distribuição normal ($p = 8.947 \times 10^{-8}$ e $W = 0.72319$). No entanto, a análise gráfica da Figura 4 sugere que os dados seguem uma distribuição normal, apesar da existência de alguns *outliers*.

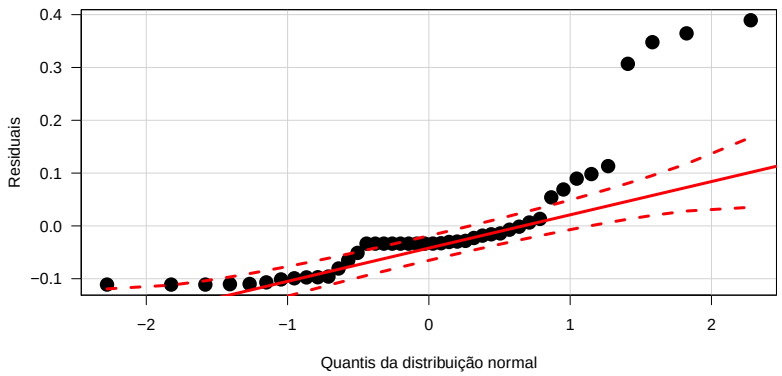

Figura 4: Gráfico de normalidade dos resultados com tempo limite de 300 segundos

Nas tabelas 1 e 2 são apresentados os valores do melhor *GAP* e do *GAP* médio alcançados por cada método de solução em 60 e 300 segundos, respectivamente.

Tabela 1: Valor do melhor *GAP* e *GAP* médio em 60 segundos

Prob.-Teste	CBC	GLPK	BPLS	HGVPRLB		HGVPRLB-CC	
				Melhor	Média	Melhor	Média
acc-tight5							
air04	**0,000000**	0,030158	x	0,000018	0,011997	**0,000000**	0,000775
bab5							
bley xl1							
bnatt350							
cov1075	**0,000000**	**0,000000**	x	**0,000000**	**0,000000**	**0,000000**	**0,000000**
eil33.2	0,063807	**0,000011**	x	0,057451	0,209233	**0,000011**	0,031552
eilB101	0,257181	0,129869	x	0,089680	0,372193	**0,033220**	0,089680
ex9							
iis-100-0-cov	**0,000000**	0,034483	x	**0,000000**	0,017241	**0,000000**	**0,000000**
iis-bupa-cov	0,111111	0,111111	x	**0,000000**	0,013889	**0,000000**	**0,000000**
iis-pima-cov	0,090909	0,090909	x	**0,000000**	0,030303	**0,000000**	0,015152
m100n500k4r1	**0,040000**	**0,040000**	x	**0,040000**	0,060000	**0,040000**	0,060000
macrophage	0,048128	0,323529	x	0,128342	0,128342	**0,032086**	0,065508
mine-166-5	0,022522	0,060896	x	0,016969	0,404094	**0,006184**	0,289934
mine-90-10			x	**0,000326**	0,505483	**0,000326**	0,033551
mspp16							
n3div36	**0,032110**		x	0,136086	0,332569	0,094801	0,312691
n3seq24			x	0,206897	0,352490	**0,153257**	0,352490
neos-1109824	**0,000000**	**0,000000**		**0,000000**	0,132275	**0,000000**	**0,000000**
neos-1337307	**0,000633**	0,003020		0,001147	0,001451	0,000746	0,001473
neos18	**0,000000**	0,375000	x	**0,000000**	**0,000000**	**0,000000**	**0,000000**
neos-849702							
netdiversion							
ns1688347							
opm2-z7-s2	0,198541	0,082685	x	0,049708	0,961819	**0,042704**	0,961819
reblock67	0,006117		x	0,008419	0,509145	**0,003922**	0,065823
rmine6	0,000555	0,004790	x	0,000405	0,001632	**0,000094**	0,003155
sp98ic	0,004462	0,159808	x	0,003493	0,007960	**0,003418**	0,011675
tanglegram1			x				
tanglegram2	0,162528	**0,000000**	x	**0,000000**	0,013544	**0,000000**	0,004515
vpphard							
Nº Soluções Viáveis	19	17	20	21	21	21	21
Nº Melhores Soluções	8	5		9	2	19	5

Nessas tabelas, o *GAP* médio no problema-teste *i* é calculado pela expressão: $GAP_i = \left(\overline{f}_i - f_i^*\right)/f_i^*$, em que f_i^* é o melhor valor da literatura para o problema-teste *i* e $\overline{f}_i$ o valor médio em 30 execuções do algoritmo nesse problema-teste. O melhor *GAP*, por sua vez, é calculado substituindo-se na expressão anterior $\overline{f}_i$ por f_i^{melhor}, sendo f_i^{melhor} o melhor valor encontrado em 30 execuções do algoritmo. 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 de programação inteira, e as colunas "BPLS", "HGVPRLB" e "HGVPRLB-CC" os métodos de [Brito (2014)], [Rezende (2015)] e o proposto neste trabalho, respectivamente. As colunas "HGVPRLB" e "HGVPRLB-CC" são subdivididas em duas colunas, sendo na primeira reportado o melhor *GAP* encontrado em 30 execuções do método, enquanto na segunda consta o resultado do *GAP* mediano 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. 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.

Tabela 2: Valor do melhor *GAP* e *GAP* médio em 300 segundos

Prob.-Teste	CBC	GLPK	BPLS	HGVPRLB		HGVPRLB-CC	
				Melhor	Média	Melhor	Média
acc-tight5							
air04	**0,000000**	0,000107	x	**0,000000**	0,046030	**0,000000**	0,000009
bab5	0,028604			**0,004013**	0,418003	**0,004013**	0,017660
bley xl1							
bnatt350							
cov1075	**0,000000**	**0,000000**	x	**0,000000**	**0,000000**	**0,000000**	**0,000000**
eil33.2	**0,000000**	**0,000000**	x	**0,000000**	0,209233	**0,000000**	**0,000000**
eilB101	0,008871	0,076710	x	**0,000000**	0,200577	**0,000000**	0,019318
ex9						**0,000000**	**0,000000**
iis-100-0-cov	**0,000000**	**0,000000**	x	**0,000000**	**0,000000**	**0,000000**	**0,000000**
iis-bupa-cov	0,027778	0,083333	x	**0,000000**	0,013889	**0,000000**	**0,000000**
iis-pima-cov	0,030303	0,090909	x	**0,000000**	0,015152	**0,000000**	0,015152
m100n500k4r1	**0,040000**	**0,040000**	x	**0,040000**	0,060000	**0,040000**	**0,040000**
macrophage	**0,010695**	0,286096	x	0,032086	0,088235	**0,010695**	0,032086
mine-166-5	**0,000699**	0,060896	x	0,993888	0,500520	0,006184	0,026431
mine-90-10	0,000328		x	**0,000326**	0,000751	**0,000326**	0,005213
mspp16							
n3div36	**0,004587**	0,373089	x	0,073394	0,165138	0,073394	0,146789
n3seq24			x	0,206897	0,180077	**0,153257**	0,193276
neos-1109824	**0,000000**	**0,000000**	x	**0,000000**	0,011905	**0,000000**	**0,000000**
neos-1337307	**0,000000**	0,003020		0,001389	0,001201	0,001389	0,001201
neos18	**0,000000**	0,250000	x	**0,000000**	**0,000000**	**0,000000**	**0,000000**
neos-849702							
netdiversion							
ns1688347							
opm2-z7-s2	0,013132	0,007296	x	**0,002237**	0,475681	**0,002237**	0,046887
reblock67	**0,000886**	0,550812	x	0,929945	0,013623	0,003922	0,003922
rmine6	0,000390	0,004790	x	0,004803	0,004163	**0,000094**	**0,000094**
sp98ic	0,006612	0,085374	x	**0,000475**	0,009764	**0,000475**	0,003136
tanglegram1			x				
tanglegram2	**0,000000**	**0,000000**	x	**0,000000**	0,030474	**0,000000**	**0,000000**
vpphard	**12,200000**					**7,400000**	**7,400000**
Nº Soluções Viáveis	22	19	21	22	22	24	24
Nº Melhores Soluções	13	6		15	3	20	11

Pela Tabela 1 observa-se que o algoritmo HGVPRLB-CC encontrou o mesmo número de soluções viáveis que o algoritmo HGVPRLB, no caso, 21, o que representa uma solução a mais que o método 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 8 melhores resultados, o GLPK detém 5, enquanto o HGVPRLB detém 9 e o algoritmo HGVPRLB-CC proposto, 19.

Por outro lado, pela Tabela 2, que mostra os resultados dos métodos quando o tempo de processamento é aumentado, observa-se que o algoritmo HGVPRLB-CC foi capaz de melhorar a qualidade das soluções obtidas em relação à Tabela 1, alcançando o maior número de soluções viáveis encontradas, no caso, 24. Esse valor representa três soluções a mais que o método BPLS, e 5 a mais que o GLPK e duas soluções a mais que o CBC e o HGVPRLB. Em relação à qualidade das soluções, observa-se que o CBC detém 13 melhores resultados, o GLPK 6, o HGVPRLB 15 e o novo algoritmo, 20.

Finalmente, em ambos os testes, sendo eles de 60 ou 300 segundos, podemos afirmar que, apesar de não haver diferença estatística entre os métodos, vê-se, claramente, pelas Figuras 1 e 3, que no algoritmo proposto a variabilidade do *GAP* médio é bem menor que no algoritmo HGVPRLB de [Rezende (2015)]. Além disso, o algoritmo HGVPRLB-CC, além de conseguir gerar o maior número de melhores soluções entre os métodos comparados, foi capaz de encontrar soluções que o algoritmo HGVPRLB não encontrou. Esses resultados são reportados nas tabelas 1 e 2.

4. Conclusões e Trabalhos Futuros

Neste trabalho foi proposto um novo algoritmo híbrido, nomeado HGVPRLB-CC, para resolver problemas de programação linear binária, sendo ele avaliado tanto com relação à capacidade de alcançar uma solução viável, quanto á capacidade de encontrar uma solução 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, tanto o algoritmo HGVPRLB de [Rezende (2015)], quanto o algoritmo HGVPRLB-CC, encontram um maior número de soluções viáveis do que o algoritmo BPLS de [Brito (2014)] e do que os resolvedores de código aberto CBC e GLPK. Além disso, quando o critério de comparação é a qualidade da solução, o algoritmo HGVPRLB-CC se mostra superior a todos os demais métodos, produzindo um maior número de melhores soluções.

Por outro lado, quando o tempo de processamento é aumentado, o algoritmo HGVPRLB-CC consegue aumentar o número de soluções viáveis e superar os resultados dos métodos BPLS, HGVPRLB e dos resolvedores GLPK e CBC. Já com relação à qualidade da solução, observa-se que ela é melhorada, sendo o HGVPRLB-CC o método que apresenta o maior número de melhores soluções para o conjunto de problemas-teste.

Finalmente, comparando-se os algoritmos HGVPRLB e HGVPRLB-CC no tempo de 300 segundos de execução, foi mostrado que, em 8 dos 32 problemas-teste, os algoritmos não foram capazes de encontrar soluções viáveis, enquanto em 2 apenas o algoritmo proposto foi capaz de encontrar soluções viáveis. Dos 22 problemas-teste em que ambos os algoritmos encontraram soluções viáveis, verificou-se que estatisticamente não há diferença entre eles; entretanto, pela análise gráfica de variabilidade das soluções finais produzidas pelos algoritmos, percebe-se, claramente, que o algoritmo proposto produz soluções com menor variabilidade em termos de *gap* médio.

Esses fatos mostram a superioridade do algoritmo proposto 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.

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 ao desenvolvimento deste trabalho.

Referências

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

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.

da Fonseca, G. H. G., Santos, H. G., Toffolo, T. A. M., Brito, S. S., e Souza, M. J. F. (2014). GOAL solver: a hybrid local search based solver for high school timetabling. *Annals of Operations Research*, 239(1):77–97.

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.

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.

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.

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

Montgomery, D. (2007). *Design and Analysis of Experiments*. John Wiley & Sons, New York, NY, fifth edition.

Rezende, J. C. V., Souza, M. J. F., e Silva, R. I. (2015). HGVPRLB: A hybrid algorithm for solving binary problems. In *Proceedings of the XLI Latin American Computing Conference (XLI CLEI)*, pages 249–257, Arequipa, Peru. IEEE Sección Perú.

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.

Shapiro, S. S. e Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, 52:591–611.

Subramanian, A., Medeiros, J. M. F., Cabral, L. A. F., e Souza, M. J. F. (2011). Aplicação da metaheurística busca tabu ao problema de alocação de aulas a salas em uma instituição universitária. *Revista Produção Online*, 11:54–75.

Toffolo, T. A. M., Souza, M. J. F., Pontes, R. C. V., e Silva, G. P. (2006). HeurÃstica de recobrimento aplicada à escala de tripulações aéreas. In *Anais do XXXVIII Simpósio Brasileiro de Pesquisa Operacional - XXXVIII SBPO*, volume 38, pages 1637–1647, Goiânia. SOBRAPO.