Uma adaptação da heurística RINS aplicada a problemas binários utilizando resolvedor CBC

Haroldo G. Santos[1], **Marcone Jamilson F. Souza**[1], **Thiago M. Gomes**[1]

[1]Departamento de Computação - ICEB Universidade Federal de Ouro Preto (UFOP)
35.400-000 – Ouro Preto – MG – Brasil

{haroldo,marcone}@iceb.ufop.br, thiagomacg@gmail.com

Abstract. *This paper proposes an adaptation of the RINS MIP heuristic which explicitly explores pre-processing techniques. The method systematically searches for the ideal number of fixations to produce sub-problems of controlled size. These problems are explored in a Variable Neighborhood Descent fashion until a stopping criterion is met. Preliminary experiments implemented upon the open source MIP solver COIN-OR CBC are presented.*
KEYWORDS*. MIP Heuristics. Otimization. RINS. Mathematical Programming.*

Resumo. *Este artigo propõe uma adaptação da MIP heurística RINS que explora explicitamente técnicas de pré-processamento. O método procura sistematicamente por um número ideal de fixações, visando produzir sub-problemas de tamanho controlado. Então estes problemas são explorados de modo semelhante ao Variable Neighborhood Descent até que um critério de parada seja satisfeito. Resultados preliminares desta implementação utilizando o resolver MIP de código aberto COIN-OR CBC são apresentados.*
PALAVRAS CHAVE*. MIP Heurísticas. Otimização. RINS. Programação Matemática.*

1 Introdução

Uma das mais importantes técnicas para resolver problemas complexos de otimização é através da Mixed Integer Programming (MIP). Um problema MIP envolve um conjunto de variáveis, um conjunto de restrições sobre estas variáveis, um conjunto de restrições de integralidade e uma função objetivo a otimizar.

MIPs são tipicamente resolvidos por técnicas de branch-and-bound ou branch-and-cut. Estas abordagens exploram uma árvore de relaxação do MIP original, em que cada nó da árvore é dividido em dois conjuntos disjuntos pela imposição de restrições de limite sobre uma variável inteira. Os resolvedores MIP são aplicados a uma variedade de problemas, entretanto sua performance em produzir boas soluções viáveis ou bons limites duais difere em cada aplicação e suas respectivas formulações. Assim devem ser consideradas pelos pesquisadores como possibilidade a utilização de heurísticas como Busca Tabu Glover (1996), Algoritmos Genéticos Reeves (1993), Goldberg (1989), Reconexão por caminhos Glover (1986), dentre outras cujo objetivo é produzir ou mesmo oferecer melhorias a uma solução. Nos últimos anos, o uso de resolvedores MIP em diferentes domínios tem motivado o desenvolvimento de diversas MIP heurísticas, como por exemplo Feasibility Pump Fischetti (2005), Local Branching Fischetti (2003) and RINS Danna (2005).

Dentre os resolvedores de código aberto disponíveis, podemos destacar o COIN-OR CBC Forrest (2005) Lougee-Heimer (2003). É um resolvedor para programação linear e inteira. Neste pacote também estão incluidas ferramentas como pré-processamento, planos de corte, heurísticas e estratégias de branching. Inicialmente foi desenvolvido para ser utilizado como uma biblioteca, mas também possui um resolvedor independente que pode ser chamado pela linha de comando. Aceita arquivos no formato .lp e .mps. Pode ser executado de maneira paralela para utilizar as vantangens de um computador multi-core.

Neste trabalho é proposta uma modificação na heurística MIP RINS: Relaxation Induced Neighborhood Search proposta por Danna (2005) na literatura. O método busca por um número ideal de variáveis, de um dado problema, a fixar. Se este número é muito pequeno, o espaço de busca pode ser muito grande tornando a exploração ineficiente. Por outro lado, se o número de fixações é muito grande, o espaço de busca fica muito restrito e a solução não é melhorada. Todas as implementações foram codificadas utilizando as bibliotecas COIN-OR CBC.

O restante do artigo é dividido como segue. Na seção 2 é apresentado a revisão da literatura considerando problemas de programação inteira e metaheurísticas. Na seção 3 é proposto o método heurístico adaptativo RINS(pRins) e a metodologia utilizada. Na seção 4, as instâncias de teste são descritas, enquanto na seção 5 os experimentos e resultados computacionais são apresentados. A última seção conclui e sugere possíveis melhorias.

2 Revisão Bibliográfica

As heurísticas tiveram origem nas comunidades de Pesquisa Operacional e Inteligência Artificial. Em princípio, as combinações de heurística não foram exploradas, pois estas encontravam boas soluções separadamente. As técnicas de hibridização surgiram com o objetivo de explorar os benefícios da sinergia entre os métodos heurísticos, no entanto, não é trivial encontrar boas combinações Blum (2011).

Blum (2011) apresenta um levantamento sobre a hibridização de metaheurísticas com outros métodos de otimização para resolver problemas de natureza combinatória. Os autores ressaltam a importância dos métodos híbridos combinarem características de diversificação e intensificação na busca de uma solução. Entre os métodos de programação inteira, destacam-se aqueles baseados em relaxação Lagrangeana, bem como heurísticas iterativas: LPA (Linear Programming based Algorithm), IRH (Iterative Relaxation based Heuristic) e IIRH (Iterative Independent Relaxation based Heuristic), que fixam e resolvem os subproblemas encontrados em cada etapa.

Para Eckstein (2007), a motivação do uso de heurísticas em MIP é que estas devem ajudar a encontrar boas soluções mais cedo, evitando assim a busca em regiões de baixa qualidade na árvore, e, ao mesmo tempo, devem intensificar a busca em regiões promissoras. Em seu trabalho é apresentada uma heurística 0-1, de implementação stand-alone, ou seja, independente do branch-and-bound. Ela é construída em torno de uma função de mérito medindo a integralidade da solução. O método envolve 4 etapas: pivoteamento baseado no gradiente, pivoteamento por sondagem, cortes de convexidade/intersecção e exploração da árvore por blocos de variáveis.

Durante a resolução de um problema estão disponíveis duas soluções, uma solução inteira que atende aos requisitos de integralidade mas não é ótima e uma solução fracionária que não atende a integralidade das variáveis, entretanto possui um melhor valor. O método

Relaxation Induced Neighborhood Search (RINS), proposto por Danna (2005), parte então deste pressuposto fixando as variáveis que estão iguais nas duas soluções, já que estas atendem aos dois critérios. Em seguida é adicionado um corte, baseado no valor da função objetivo atual e resolvido o subproblema com as variáveis restantes. O método possui características semelhantes ao Path Relinking Glover (1986), pois conecta duas soluções.

Em Parisini (2011), os autores aplicam ao Problema do Caixeiro Viajante, duas técnicas de busca em árvore: Local Branching (LB) e Sliced Neighborhood Search (SNS). Enquanto o LB produz intensificação na região da solução incumbente explorando o espaço próximo, a técnica SNS atua diversificando a busca. A técnica SNS melhora a solução incumbente por exploração aleatória de espaços distantes da vizinhança. Ela explora o espaço considerando as disparidades entre a solução incumbente e seus vizinhos. A cada iteração um pedaço da vizinhança é explorado escolhendo-se um conjunto de variavéis ou usando pequenos limites de tempo. O resultado da combinação das duas técnicas se mostrou satisfatório, encontrando soluções de melhor qualidade.

Ghosh (2007) apresenta a heurística Distance Induced Neighbourhood Search (DINS). A idéia nesta heurística é usar uma métrica de distância entre a relaxação linear e a solução inteira corrente, explorando os nós gerados pela busca. O método DINS incorpora hard-fixating, soft-fixating e arredondamento de variavéis de acordo com a métrica definida. Segue a intuição de que boas soluções estão próximas da solução relaxada. O método também considera um critério para evitar fixações em número demasiado, caso muitas variáveis inteiras sejam fixadas.

Rothberg (2007) descreve uma abordagem evolutiva para melhorar as soluções dos modelos de programação inteira mista (MIP). Algoritmos evolutivos adotam uma analogia com a seleção natural, explorando conceitos como população, combinação, mutação e seleção para explorar um espaço diversificado de possíveis soluções para os problemas de otimização combinatória, enquanto, ao mesmo tempo, retém as propriedades desejáveis das soluções conhecidas. O método proposto mantém um conjunto (pool) de tamanho fixo, contendo as P melhores soluções distintas encontradas. Neste conjunto serão executadas as operações de combinação e mutação. Na mutação, uma solução é escolhida, em seguida um percentual das variáveis é fixado. O sub-problema resultante é resolvido, a melhor solução é adicionada ao conjunto e o percentual de fixação é atualizado. Na combinação é escolhido um par de soluções, as variáveis cujos valores coincidem em todas as soluções escolhidas são fixadas. Em seguida, o sub-problema resultante é resolvido e a melhor solução encontrada é adicionada ao pool de soluções. Os experimentos mostraram resultados satisfatórios.

A heuristica Relaxation Enforced Neighborhood Search (RENS) apresentada em Berthold (2009), trabalha com busca em vizinhança de grande porte. O método constrói um subproblema, considerando os arredondamentos viáveis de algum ponto fracionário normalmente o ótimo da relaxação LP do MIP original. A solução atual é o ponto de partida e são definidas vizinhanças por meio de fixações e adição de restrições. A ideia do RENS é fixar as variáveis com valor inteiro na solução relaxada e fazer a busca nas restantes, arredondando para o inteiro mais próximo.

3 A MIP heurística pRINS

A heurística desenvolvida neste trabalho é baseada no método RINS. Neste, em cada nó da árvore de branch-and-cut as seguintes operações são realizadas:

1. Fixar as variáveis com o mesmo valor na solução incumbente e na solução relaxada;
2. Definir um corte com base no valor da função objetivo na solução incumbente corrente;
3. Resolver o sub-MIP com as variáveis restantes.

Assim o método RINS fixa todas as variáveis que são iguais entre as duas soluções. Esta estratégia pode gerar ora sub-problemas com muita fixação (assim o espaço de soluções fica pequeno), ou sub-problemas com pouca fixação (com espaço de soluções extenso). O método proposto explora técnicas de pré-processamento existentes para gerar rapidamente sub-problemas com tamanho controlado. Estes sub-problemas são resolvidos de maneira semelhante ao VND (Variable Neighborhood Descida) Mladenovic (1997).

Algorithm 1: pRINS

Input: $Sol_f, Sol_i, mip, Req_{size}, N_{atpmax}, Dif_p$
Output: Sol^*
$Sol^* \leftarrow Sol_i$;
X_p recebe um vetor de variáveis ordenado de acordo com sua prioridade de fixação considerando as soluções inteiras e fracionárias (Sol_i, Sol_f);
repeat
  $fix_{size} \leftarrow$ buildSizes($X_i, X_p, N_{atpmax}, Req_{size}, Dif_p, Lim_{Rel}$);
  $mip'' \leftarrow$ createProblem($X_i, X_p, fix_{size}, mip$);
  if *mip'' custo da relaxação indica possibilidade de melhora* **then**
    $Sol_{rins} \leftarrow$ solve mip'';
    if *Encontrou uma solução melhor* **then**
      $Sol^* \leftarrow Sol_{rins}$;
      Recalcular o vetor de prioridades (X_p) considerando Sol^*, Sol_f ;
    else
      Atualizar o tamanho do sub-problema requerido Req_{size};
    end
  else
    Atualizar o tamanho do sub-problema requerido Req_{size};
  end
until *o tamanho de sub-problema requerido é menor que o número total de variáveis e o tempo limite não foi excedido* ;
return Sol^*;

O método pRINS(pre-processing RINS), algoritmo 1, estabelece um critério de parada na aplicação desta heurística adaptada. Dada uma solução inteira (Sol_i), uma solução fracionária (Sol_f), o tamanho inicial de sub-problema desejado (Req_{size}), o número máximo de tentativas para construir um sub-problema no tamanho desejado (N_{atpmax}), um percentual de diferença aceitável entre o tamanho desejado e encontrado (Dif_p) e o problema MIP original (mip). Inicialmente é criado um vetor ordenado indicando as prioridades de fixação, este é definido calculando-se a diferença em magnitude entre os valores da solução inteira e fracionária. As variáveis cuja diferença entre as soluções (Sol^*) e (Sol_f)

é zero ficam no início do vetor, enquanto as demais estão ordenadas pela diferença ascendente dos valores. O método buildSizes, definido no algoritmo 2, é usado para determinar o número de fixações (fix_{size}) necessária para atingir o tamanho de sub-problema desejado. Neste, é criado um novo sub-problema(mip''), que em seguida é pré-processado, considerando o número de fixações determinado previamente. Se o custo da relaxação(obtido no pré-processamento) não indica possibilidade de melhora, então o tamanho desejado é atualizado e incrementado. Se existe possibilidade de melhora, considerando o valor da relaxação, o sub-problema será resolvido. Ao resolver, se uma solução melhor é encontrada, a solução inteira é atualizada e as prioridades são recalculadas. De outra maneira se a solução encontrada é de piora, o tamanho desejado é atualizado e incrementado (Req_{size}) em um percentual para considerar um espaço de busca maior.

Algorithm 2: buildSizes

Input: $X_i, X_p, N_{atpmax}, N_{des}, Dif_p, Lim_{Rel}$
Output: N_{fix}
$N_a \leftarrow 0$;
$L_l \leftarrow 0$;
$L_u \leftarrow maxSize_{last}$;
$N_{fix} \leftarrow (L_l + L_u)/2$;
$Dif_{max} \leftarrow 0$;
repeat
  $N_a + +$;
  $mip'' \leftarrow$ createProblem(X_i, X_p, N_{fix}, mip);
  $Prob_{size} \leftarrow$ nvars(mip'');
  if $|N_{des} - Prob_{size}| \leq N_{des} * Dif_p$ **then**
    return N_{fix};
  end
  if $Prob_{size} == 0$ **or** $Prob_{size} < N_{des}$ **or** $Lim_{Rel} \geq 0.99 * cost(S_i)$ **then**
    $L_u \leftarrow N_{fix}$;
    $N_{fix} \leftarrow (L_l + L_u)/2$;
  else
    $L_l \leftarrow N_{fix}$;
    $N_{fix} \leftarrow (L_l + L_u)/2$;
  end
until $N_a < N_{atpmax}$ **or** $L_l < L_u$;
return N_{fix};

O método buildSizes, é usado para determinar o número ideal de fixações. Dado o tamanho desejado do problema (N_{des}), que no início da execução é um tamanho de problema mip suficientemente pequeno para que possa ser resolvido rapidamente, o número total de variáveis (size(X_i)), número máximo de tentativas (N_{atpmax}), o vetor de prioridade das fixações (X_p), o vetor de variáveis inteiras (X_i), então os sup-problemas MIPs serão criados e o número de fixações necessário é encontrado. Para alcançar o tamanho desejado, o número de fixações vai sendo testado através de uma busca binária até que o número máximo de tentativas se esgote, a busca binária termine ou o tamanho seja encontrado. Inicialmente o limite superior de fixações é o número total de variáveis do problema. Este limite é modificado ao final de cada busca binária, sendo atualizado para o número de fixações utilizado no último sub-problema resolvido. Este controle é feito utilizando

a variável ($maxSize_{last}$). O método chama outro (createProblem, definido no algoritmo 3) que verifica a viabilidade do sub-problema resultante. Depois de construído, a função nvars retorna o número de variáveis livres neste sub-problema. A busca termina quando o número de variáveis livres ($Prob_{size}$) atinge o tamanho desejado (N_{des}), dado um percentual de tolerância (Dif_p). Senão, a busca termina se o número máximo de tentativas (N_{atpmax}) é alcançado, ou o limite superior (L_u) é menor que o limite inferior (L_l).

Algorithm 3: createProblem

Input: X_i, X_p, N_{fix}, mip
Output: mip'
$mip' \leftarrow fix(mip, N_{fix}, X_p, X_i)$;
$mip'' \leftarrow preprocessed(mip')$;
if $mip''inviável$ **then**
  $mip'' \leftarrow$ null ;
end
return mip'';

O método createProblem, algoritmo 3, irá construir sub-problemas mip, considerando o número de variáveis a ser fixado (N_{fix}), e o vetor de prioridades (X_p). A função $fix(mip, N_{fix}, X_p, X_i)$ cria os sub-problemas, da fixação das (N_{fix}) primeiras variáveis do vetor (X_p), pela definição dos limites superiores e inferiores de cada uma das variáveis X_i. O sub-problema criado é pré-processado, gerando outro (mip''). Se o mip (mip'') é viável, o método retorna o tamanho deste. Se este é inviável, retorna 0 como tamanho.

4 Caracterização das Instâncias

Os modelos (instâncias) utilizados neste trabalho são relacionados a problemas binários. Eles foram obtidos de dois grupos:

- 25 problemas da biblioteca MIPLIB `http://miplib.zib.de/` Koch (2011).
- 12 problemas nurse scheduling utilizados na competição International Nurse Rostering Competition 2010 Haspeslagh (2010)

Uma descrição detalhada das instâncias, contendo o número de variáveis binárias, o número de restrições e o número de valores não-zero em restrições, está disponível na tabela 1.

Podemos perceber a diversidade das instâncias utilizados, observando a variação em número de variáveis e número restrições.

5 Experimentos e Resultados

A heurística proposta neste trabalho, lê um série de instâncias testes (inicialmente aplicado a problema binários, como descrito na seção anterior), a solução fracionária e uma solução inicial inteira viável para o problema. A solução inicial foi gerada utilizando Feasibility Pump e informada para todas as heurísticas MIP testadas.

Para a maioria das aplicações de pesquisa práticas, métodos de soluções serão somente usuais se são capazes de produzir uma solução satisfatória para o problema em um curto período de tempo. Assim, foi utilizado como o tempo limite de 300 segundos. O parâmetro tamanho inicial de problema a ser resolvido foi definido em 100 (número de

Tabela 1. Nurse Scheduling e problemas da biblioteca MIPLIB

instância	variáveis binárias	restrições	não-zeros
long01	51.695	17.241	1.011.556
long-hidden01	61.950	28.370	1.064.380
long-hint01	61.550	27.480	1.061.430
long-late01	61.750	27.875	1.062.795
medium01	29.605	8.668	621.829
medium-hidden01	36.690	16.070	635.220
medium-hint01	34.050	14.062	622.800
medium-late01	34.050	14.062	622.800
sprint01	3.522	10.230	204.000
sprint-hidden01	10.308	3.332	202.420
sprint-hint01	11.630	5.032	208.410
sprint-late01	11.630	5.032	208.410
air04	8.904	823	72.965
bley-xl1	5.831	175.620	869.391
cov1075	120	637	14.280
eil33-2	4.516	32	44.243
eilB101	2.818	100	24.120
iis-100-0-cov	100	3.831	22.986
iis-bupa-cov	345	4.803	38.392
iss-pima-cov	768	7.201	71.941
macrophage	2.260	3.164	9.492
mine-166-5	830	8.429	19.412
mine-90-10	900	6.270	15.407
n3div36	22.120	4.484	340.740
n3seq24	119.856	6.044	3.232.340
neos-1109824	1.520	28.979	89.528
neos-1337307	2.840	5.687	30.799
neos18	3.312	11.402	24.614
netdiversion	129.180	119.589	615.282
ns1688347	2.685	4.191	66.908
opm2-z7-s2	2.023	31.798	79.762
reblock67	670	2.523	7.495
rmine6	1.096	7.078	18.084
sp98ic	10.894	825	316.317
tanglegram1	34.759	68.342	205.026
tanglegram2	4.714	8.980	26.940
vpphard	51.471	47.280	372.305

variáveis livres) para estes testes. Outro parâmetro é o percentual de aumento no tamanho do problema (neste caso, usado como 50). Os testes foram executados usando a seguinte configuração de hardware: Intel (R) Core (TM) i7 CPU, 1.90GHz, 6 GB RAM. Testes também foram realizados para o método proposto considerando o tempo limite de 600 segundos, buscando avaliar a variação das soluções encontradas.

Os resultados preliminares descritos abaixo, consideram a implementação do método RINS adaptado no trabalho. As instâncias utilizadass nos testes são as descritas na tabela 1.

São comparados nossos resultados com o solver CBC stand-alone, e com uma implementação do método RINS na forma original. Ambos CBC e RINS usam a mesma solução inicial que nosso método.

Na tabela 2 são descritos os valores encontrados para cada instância, considerando as implementações usadas. Para cada instância, o melhor valor está destacado em negrito. Comparando os valores obtidos, podemos perceber que o método proposto alcança os melhores resultados em 14 instâncias se comparado ao CBC e é melhor em 18 instâncias quando comparado a forma original do RINS. Em 15 instâncias o método obtém o mesmo valor quando comparado ao valor encontrado resolvendo com o CBC, e também o resultado é o mesmo para 15 instâncias quando comparado ao RINS original.

As tabelas 3 e 4 apresentam os resultados considerando o número de vitórias, empates e derrotas comparando as implementações testadas. Em todos os grupos de instâncias o método proposto alcança bons resultados, encontrando soluções melhores ou iguais. Por exemplo, no grupo de instâncias da biblioteca MIPLib , o pRINS obtém 76% dos valores iguais ou melhores quando comparado ao CBC, e 88% comparado ao RINS.

A tabela 5 mostra os resultados para cada implementação testada considerando a soma e a média dos gaps (diferença percentual entre o valor obtido e o melhor valor conhecido). O método proposto é o que mais se aproxima, na média, dos melhores valores. A métrica gap da solução foi calculado de acordo com a seguinte expressão: $\min(100, (z - best) \div best \times 100)$.

A tabela 6 traz os resultados considerando a implementação testada, utilizando diferentes tempos de execução. Estes tempos foram fixados em 300, 600 e 900 segundos. Na maioria das instâncias não houve diferença dos valores de solução encontrados, nestes casos a execução terminou sem utilizar todo o tempo, pois o tamanho máximo de problema foi resolvido, ou ainda a execução terminou sem encontrar nova solução. Em apenas algumas instâncias houve melhora, para estes casos o aumento do tempo foi eficiente, na medida que novos tamanhos de problemas foram explorados.

6 Considerações Finais

Ainda que o trabalho esteja em desenvolvimento, resultados encorajadores foram obtidos por esta variante proposta do RINS, denominada aqui como pRINS. Ajustes estão sendo feitos no método para aumentar a velocidade das múltiplas fases de pré-processamento. Os resultados computacionais já demonstram que o método pRINS já é melhor ou igual aos outros métodos (usando somente CBC ou RINS original), na maioria dos casos, considerando a produção de boas soluções viáveis em tempo computacional restrito. O aumento do tempo limite não representou melhoras significativas no valor das soluções.

Tabela 2. Resultados

Instâncias	pRINS		CBC		RINS		Melhor
	z	gap %	z	gap %	z	gap %	z
long01	**161**	0,0	**161**	0,0	**161**	0,0	161
long-hidden01	2.745	100,0	2.747	100,0	**2.606**	100,0	346
long-hint01	**74**	0,0	**74**	0,0	**74**	0,0	74
long-late01	**2.895**	100,0	**2.895**	100,0	**2.895**	100,0	235
medium01	**374**	0,0	383	2,4	383	2,4	374
medium-hidden01	**1.287**	100,0	**1.287**	100,0	**1.287**	100,0	111
medium-hint01	**245**	0,0	255	4,0	250	2,0	245
medium-late01	**779**	100,0	970	100,0	966	100,0	157
sprint01	91	22,9	**74**	0,0	96	29,7	74
sprint-hidden01	92	100,0	**80**	100,0	99	100,0	32
sprint-hint01	**297**	0,0	400	34,6	496	67,0	297
sprint-late01	**74**	0,0	81	9,4	84	13,5	74
air04	**56.137**	0,0	56.138	0,0	60.093	7,0	56.137
bley-xl1	230	21,0	230	21,0	**215**	13,1	190
cov1075	**20**	0,0	**20**	0,0	**20**	0,0	20
eil33-2	1.011	8,2	**1.000,24**	7,0	1.858	98,9	934
eilB101	2.093	72,1	**1.529,88**	25,7	2.298	88,9	1216
iis-100-0-cov	30	3,4	**29**	0,0	32	10,3	29
iis-bupa-cov	**37**	2,7	38	5,5	38	5,5	36
iss-pima-cov	34	3,0	34	3,0	**33**	0,0	33
macrophage	**447**	19,5	809	100,0	596	59,3	374
mine-166-5	**-3,98740e+08**	29,6	**-3,98740e+08**	29,6	**-3,98740e+08**	29,6	-5,66396e+08
mine-90-10	**-3,98740e+08**	49,1	**-3,98740e+08**	49,1	**-3,98740e+08**	49,1	-7,84302e+08
n3div36	**136.000**	3,9	148.600	13,6	149.800	14,5	130.800
n3seq24	73.200	40,2	**62.800**	20,3	69.400	32,9	52.200
neos-1109824	467	23,5	**380**	0,5	760	100,0	378
neos-1337307	-201.447	0,4	**-201.466**	0,4	-201.447	0,4	-202319
neos18	**16**	0,0	**16**	0,0	**16**	0,0	16
netdiversion	**370**	52,8	451	86,3	451	86,3	242
ns1688347	**35**	29,6	**35**	29,6	**35**	29,6	27
opm2-z7-s2	-1.317	87,1	**-9.365**	8,9	-1.317	87,1	-10.280
reblock67	**-2,19704+e07**	36,5	**-2,19704+e07**	36,5	**-2,19704+e07**	36,5	-3,46306+e07
rmine6	**-449,25**	1,7	**-449,25**	1,7	**-449,25**	1,7	-457,18
sp98ic	**4,50307+e08**	0,2	4,87071+e08	8,4	4,68947+e08	4,4	4,49145+e08
tanglegram1	**5.494**	6,0	**5.494**	6,0	**5.494**	6,0	5182
tanglegram2	**443**	0,0	**443**	0,0	**443**	0,0	443
vpphard	**16**	68,7	22	100,0	22	100,0	5

Tabela 3. Número de vitórias e derrotas em cada grupo de instâncias

Grupo	Melhor Valor		
	pRINS	Igual	CBC
Biblioteca MIPLIB	8	11	6
Nurse Scheduling	6	4	2

Tabela 4. Número de vitórias e derrotas em cada grupo de instâncias

Grupo	Melhor Valor		
	pRINS	Igual	RINS
Biblioteca MIPLIB	11	11	3
Nurse Scheduling	7	4	1

Tabela 5. Soma e média dos gaps

	p-RINS	CBC	RINS
Soma	1.082,10	1.103,52	1.475,70
Média	29,24	29,82	39,88

Tabela 6. Resultados considerando diferentes tempos de execução

Instâncias	pRINS 300s	pRins 600s	pRins 900s
long01	161	161	161
long-hidden01	2.745	2.745	2.745
long-hint01	74	74	74
long-late01	2.895	2.895	2.895
medium01	374	349	348
medium-hidden01	1.287	1.287	1.287
medium-hint01	245	234	234
medium-late01	779	615	615
sprint01	91	91	91
sprint-hidden01	92	92	92
sprint-hint01	297	297	297
sprint-late01	74	74	74
air04	56.137	56.137	56.137
bley-xl1	230	230	230
cov1075	20	20	20
eil33-2	1.011	1.011	1.011
eilB101	2.093	1.691	1.691
iis-100-0-cov	30	30	30
iis-bupa-cov	37	37	37
iss-pima-cov	34	34	34
macrophage	447	446	446
mine-166-5	-3,98740e+08	-3,98740e+08	-3,98740e+08
mine-90-10	-3,98740e+08	-3,98740e+08	-3,98740e+08
n3div36	136.000	136.000	136.000
n3seq24	73.200	72.000	70.400
neos-1109824	467	467	467
neos-1337307	-201.447	-201.447	-201.447
neos18	16	16	16
netdiversion	370	370	370
ns1688347	35	34	34
opm2-z7-s2	-1.317	87,1	50
reblock67	-2,19704+e07	-2,19704+e07	-2,19704+e07
rmine6	-449,25	-449,25	-449,25
sp98ic	4,50307+e08	4,50307+e08	4,50307+e08
tanglegram1	5.494	5.494	5.494
tanglegram2	443	443	443
vpphard	16	14	14

7 Agradecimentos

Os autores agradecem pelo apoio oferecido pela UFOP, FAPEMIG e CNPq.

8 Referências

Berthold, T. (2009). Rens: The relaxation enforced neighborhood search. ZIB-Report / Konrad-Zuse-Zentrum für Informationstechnik Berlin. Konrad-Zuse-Zentrum für Informationstechnik.

Blum, C., Puchingerb, J., Raidl, G., and Roli., A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. In *Applied Soft Computing*, volume 11, pages 4135–4151.

Danna, E., Rothberg, E., and Le Pape, C. (2005). Exploring relaxation induced neighborhoods to improve mip solutions. volume 102, pages 71–90.

Eckstein, J. and Nediak, M. (2007). Pivot, cut, and dive: a heuristic for 0-1 mixed integer programming. In *Journal Heuristics*, volume 13(5), pages 471–503.

Fischetti, M., Glover, F., and Lodi., A. (2005). The feasibility pump. In *Mathematical Programming*, volume 104(1), page 9104.

Fischetti, M. and Lodi, A. (2003). Local branching. In *Mathematics Programming, ser. B 98*, volume 98(1-3), pages 23–47.

Forrest, J. and Lougee-Heimer, R. (2005). Cbc user guide. In *INFORMS Tutorials in Operations Research*, pages 257–277.

Ghosh, S. (2007). Dins, a mip improvement heuristic. In *International IPCO Conference*, volume IN IPCO, pages 310–323.

Glover, F. (1986). In *Future paths for Integer Programming and links to Artificial Intelligence*, pages 533–549.

Glover, F. (1996). Tabu search and adaptive memory programming - advances, applications and challenges. In *Interfaces in Computer Sciences and Operations Research*, volume 7, pages 1–75.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, volume 3. Addison-Wesley.

Haspeslagh, S., De Causmaecker, P., Stolevik, M., and A., S. (2010). First international nurse rostering competition 2010. codes, department of computer science. kuleuven campus kortrijk. belgium.

Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R., Danna, E., Gamrath, G., Gleixner, A., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D., and Wolter, K. (2011). Miplib 2010. mathematical programming computation. In Zuse Institute Berlin, Takustr. 7, . B. G., editor, *Mathematics and Statistics*, volume 3, pages 103–163. Springer Berlin/Heidelberg Vol.3.

Lougee-Heimer, R. (2003). The common optimization interface for operations research: Promoting open-source software in the operations research community. In *IBM Journal of Research and Development*, volume 47, pages 57–66.

Mladenovic, N. and Hansen, P. (1997). Variable neighborhood search. In *Computers and Operations Research*, volume 24, pages 1097–1100.

Parisini, F. and Milano, M. (2011). Improving cp-based local branching via sliced neighborhood search. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 887–892.

Reeves, C. (1993). Genetic algorithms. In *Modern Heuristic Techniques for Combinatorial Problems, Advanced Topics in Computer Science Series*, pages 151–196. Blackwell Scientific Publications.

Rothberg, E. (2007). An evolutionary algorithm for polishing mixed integer programming solutions. In *INFORMS Journal on Computing*, volume 19, pages 534–541.