Arredondamento Estocástico Estruturado para Programação Inteira 0-1 Mista

Wendel Melo
Faculdade de Computação
Universidade Federal de Uberlândia
wendelmelo@ufu.br

Marcia Fampa
Instituto de Matemática e COPPE
Universidade Federal do Rio de Janeiro
fampa@cos.ufrj.br

Fernanda Maria Pereira
Laboratório Nacional de Computação Científica
fernanda@lncc.br

ABSTRACT

Este trabalho introduz uma heurística de viabilidade para programação inteira 0-1 mista. Nossa heurística procura construir soluções viáveis respeitando classes especiais de restrições presentes no problema e fazendo uso de um esquema de arredondamento. Adicionalmente, buscas locais são aplicadas com a expectativa de melhorar a solução obtida. Resultados computacionais apontaram a competitividade do método proposto frente a abordagens de mesma natureza para problemas de programação não linear inteira 0-1 mista.

PALAVRAS CHAVE: Programação Inteira 0-1 Mista, Heurística de viabilidade, Muriqui Optimizer.
Área de classificação principal: Otimização Combinatória.

ABSTRACT

This paper introduces a feasibility heuristic for mixed 0-1 integer programming. Our heuristic tries to build feasible solutions respecting special classes of constraints present in the problems, making use of a rounding scheme. Additionally, local search is applied with the expectation of improving the solution obtained. Computational results pointed out the competitiveness of the proposed method against approaches of the same nature for mixed 0-1 integer nonlinear programming problems.

KEYWORDS: Mixed 0-1 Integer Programming, Feasibility Heuristic, Muriqui Optimizer.
Main area: Combinatorial Optimization.

1. Introdução

Neste trabalho, abordamos problemas gerais de programação inteira 0-1 mista que incorporem restrições lineares sobre as variáveis inteiras:

$$\begin{array}{lll} (P) & \min\limits_{x,y} & f(x,y) \\ & \text{s. a} & g(x,y) \leq 0, \\ & & Ay \ \{\leq, =, \geq\} \ b, \\ & & x \in \mathbb{R}^{n_x}, \ y \in \{0,1\}^{n_y}, \end{array} \tag{1}$$

Problemas de programação inteira 0-1 mista podem modelar aplicações diversas e costumam exigir árduo trabalho para sua resolução. A pesquisa por heurísticas de viabilidade tem se mostrado altamente relevante devido, entre outras, às seguintes razões: (1) Em muitos casos, a obtenção de uma solução viável para esse tipo de problema já é uma tarefa difícil, não podendo ser realizada em tempo hábil por métodos exatos; (2) Algumas situações práticas requerem a obtenção dinâmica de soluções viáveis, o que demanda o uso de métodos de baixo custo computacional; (3) Métodos exatos, como *Branch-and-Bound* e métodos de aproximação linear para problemas com funções não lineares, podem ter seu desempenho acelerado caso uma solução viável seja conhecida de ante-mão (veja [Melo et al., 2020b] para um apanhado geral). (4) Heurísticas de melhoramento, como *Local Branching* [Fischetti and Lodi, 2003], precisam partir de alguma solução viável já conhecida para buscar por melhores soluções; (5) De modo a conseguir tratar com sucesso uma ampla gama de problemas com características distintas, pacotes computacionais de programação inteira 0-1 mista devem possuir um arsenal vasto e diverso de heurísticas de viabilidade a serem aplicadas.

Diante das razões expostas, este trabalho apresenta uma heurística de viabilidade para problemas de programação inteira 0-1 mista que incorporem restrições lineares. Chamamos nossa heurística de *Arredondamento Estocástico Estruturado* (AEE). A ideia central de AAE consiste na fixação estocástica de variáveis inteiras observando classes especiais de restrições lineares que envolvam variáveis inteiras. AAE adota também um esquema de arredondamento estocástico para fixar variáveis inteiras que não tenham sido fixadas no procedimento anterior. Por fim, buscas locais são aplicadas para melhorar a solução obtida.

Este texto está organizado do seguinte modo: a Seção 2 apresenta os fundamentos de AEE com seus respectivos subprocedimentos, ao passo que a Seção 3 traz resultados computacionais da aplicação de AEE em instâncias convexas de programação não linear inteira 0-1 mista, com comparação com algoritmos da literatura para esse tipo de problema.

2. Arredondamento Estocástico Estruturado

Uma ideia simples para tentar obter uma solução viável para (P) seria resolver sua relaxação contínua e arredondar o valor de cada variável inteira para o número inteiro mais próximo. Todavia, esse procedimento pode fracassar em muitos casos. Suponha, por exemplo, um problema de programação inteira 0-1 incorporando a seguinte restrição:

$$y_1 + y_2 + y_3 + y_4 = 1 \tag{2}$$

Suponha agora que a relaxação contínua desse mesmo problema tenha produzido como solução $\tilde{y} = [0{,}25,\ 0{,}25,\ 0{,}25,\ 0{,}25]$. Note que o arredondamento de $\tilde{y}$ produzirá $\bar{y} = [0,\ 0,\ 0,\ 0]$, que não satisfaz à equação (2). Tomando em conta exemplos como esse, nossa motivação neste trabalho é construir um procedimento que determine valores para as variáveis inteiras observando a existência de classes especiais de restrições. Por exemplo, para satisfazer a restrição (2) em particular, podemos escolher uma das variáveis y_i, $i = 1, \ldots, 4$, para fixar em 1 e fixar as demais variáveis em zero. Tomando apenas (2) em separado, não temos elementos para definir a melhor escolha para a variável a ser fixada em 1. Nesse contexto, pode ser interessante realizar essa escolha aleatoriamente, de modo a minimizar as chances do procedimento ficar entravado ao lidar com um número maior restrições e aumentar as chances de sucesso do algoritmo com o aumento do número de tentativas de aplicação.

Diante do exposto, nosso algoritmo, denominado *Arredondamento Estocástico Estruturado* (AEE), tem cinco elementos principais: (1) Uma *rotina de pré-processamento* de restrições lineares que pode fixar variáveis inteiras baseado em análise das restrições do problema; (2) Um *procedimento estocástico de fixação de variáveis* que procura atender certas classes especiais de

restrições lineares; (3) Uma etapa de *arredondamento estocástico* para lidar com variáveis inteiras que não tenham sido fixadas no procedimento anterior; (4) A *resolução do problema contínuo* resultante da fixação das variáveis inteiras; (5) Um procedimento de busca local para a melhoria da solução obtida no passo anterior.

2.1. Pré-processamento de restrições lineares

É comum a adoção de etapas de pré-processamento em resolvedores de programação inteira com o objetivo de facilitar o tratamento do problema abordado ou reformulá-lo para algum formato de interesse a uma determinada metodologia. Através de técnicas de pré-processamento, pode-se, por exemplo, eliminar restrições e variáveis redundantes, reduzir as dimensões do problema através de substituição de variáveis e reduzir os intervalos de variação (caixa) de variáveis e restrições. Uma rotina de pré-processamento de restrições lineares é fundamental ao algoritmo AEE, pois, além de poder ser aplicada no início do procedimento (imediatamente após a leitura dos dados de entrada), ela também pode ser utilizada ao longo da execução do algoritmo. Esse uso suplementar de pré-processamento tem como expectativa a fixação de um conjunto adicional de variáveis a cada vez que um outro conjunto de variáveis são fixadas, aumentando assim as chances de obtenção de uma solução viável ao final do processo. Nesse trabalho, utiliza-se pré-processamento de restrições lineares baseado nas técnicas apresentadas em [Savelsbergh, 1994].

2.2. Fixação Estocástica Estruturada

O procedimento de *Fixação Estocástica Estruturada* (FEE) é o pilar central de AEE. Sua finalidade é a procura por certas classes especiais de restrições que são comuns em formulações de programação inteira 0-1, com a definição de valores estocásticos para um subconjunto de variáveis que garanta o atendimento dessas restrições. Assim, FEE varre as restrições do problema e, ao detectar restrição em uma das classes especiais, realiza o tratamento adequado. Sempre que uma variável é fixada em um determinado valor, ela não pode mais ser desfixada posteriormente (a menos que todo o processo seja reiniciado com a execução de uma nova iteração geral no algoritmo AEE).

Para descrever o tratamento dado a cada uma das classes de restrições, é conveniente fazer as seguintes definições: Sejam l_y e u_y vetores com, respectivamente, os limites inferiores e superiores correntes para as variáveis inteiras y. Seja l_{y_i} e u_{y_i} os limites inferior e superior correntes de y_i, respectivamente. Inicialmente, $l_{y_i} = 0$ e $u_{y_i} = 1$ para todo y_i. Ao longo da execução do algoritmo, l_{y_i} ou u_{y_i} são alterados para fixar cada y_i em 0 ou em 1. Dado um subconjunto I de índices arbitrários de variáveis inteiras 0-1 e um valor $v \in \{0, 1\}$, seja $I^v = \{i | i \in I \wedge l_{y_i} = v \wedge u_{y_i} = v\}$, isto é, I^v é o subconjunto de I referente às variáveis correntemente já fixadas no valor v. Assim I^0 é o subconjunto de I referente às variáveis já fixadas em 0, e I^1 é o subconjunto de I referente às variáveis já fixadas em 1. Definimos ainda I^F como o subconjunto de I referente às variáveis que ainda não estão fixadas ("livres"), isto é, $I^F = \{i | i \in I \wedge l_{y_i} \neq u_{y_i}\}$. Por fim, definimos como i_1, i_2, ..., $i_{|I|}$ os índices pertencentes aos conjunto I. De modo a facilitar o entendimento, diremos que o índice i foi fixado em v quando a variável y_i for fixada em v. Descrevemos a seguir as classes de restrições consideradas por FEE e seu respectivo tratamento.

2.2.1. Classe 0: Soma de variáveis inteiras 0-1 igual à constante

Esta classe é composta por restrições na forma:

$$y_{i_1} + y_{i_2} + \ldots + y_{i_{|I|}} = b, \tag{3}$$

onde $b \geq 0$. Esse tipo de restrição é tratada da seguinte forma: seja $\bar{b} = b - |I^1|$. Se $\bar{b} < 0$ ou $|I^F| < \bar{b}$, declara-se impossibilidade de atender a restrição. Caso contrário, escolhe-se aleatoriamente $\bar{b}$ índices em I^F para fixar em 1, e fixa-se os demais índices em I^F em zero.

2.2.2. Classe 1: Conservação de fluxo

Esta classe é composta por restrições na forma:

$$y_{i_1} + y_{i_2} + \ldots + y_{i_{|I|}} - y_{j_1} - y_{j_2} - \ldots - y_{j_{|J|}} = b, \tag{4}$$

onde $I \neq \emptyset$ e $J \neq \emptyset$. A filosofia geral de tratamento desse tipo de restrição consiste em sortear índices em I^F e J^F de modo a garantir o seu atendimento. Por exemplo, considere o caso onde $b = 0$, $I^1 = \emptyset$ e $J^1 = \emptyset$. Note que, nessa situação, temos:

$$y_{i_1} + y_{i_2} + \ldots + y_{i_{|I|}} = y_{j_1} + y_{j_2} + \ldots + y_{j_{|J|}}. \tag{5}$$

Nesse contexto específico, sorteamos uma quantidade aleatória r de índices em I^F para fixar em 1. A mesma quantidade de índices r deve ser fixada em 1 em J^F. Os demais índices em I^F e J^F devem ser fixados em 0. O tratamento do caso geral deve considerar as situações em que $b \neq 0$, que exigirá um número maior de fixações em I^F, se $b > 0$, ou em J^F, se $b < 0$. Também é preciso levar em conta as situações onde $I^1 \neq \emptyset$ e $J^1 \neq \emptyset$, isto é, quando já se tem variáveis previamente fixadas em 1. O Algoritmo 1 detalha o tratamento do caso geral da Classe 1 considerando todas essas nuances.

```
1  b̄ := b − |I¹| + |J¹|;
2  se b̄ ≥ 0 então
3      se b̄ > |I^F| então
4          Declarar impossibilidade de atender à restrição e encerrar o algoritmo;
5      r̄ := min(|J^F|, |I^F| − b̄);
6      Sortear um número inteiro r no intervalo [0, r̄] ;
7      Fixar r índices aleatórios de J^F em 1 e fixar demais índices de J^F em 0;
8      Fixar r + b̄ índices aleatórios de I^F em 1 e fixar demais índices de I^F em 0;
9  senão
10     se |b̄| > |J^F| então
11         Declarar impossibilidade de atender à restrição e encerrar o algoritmo;
12     r̄ := min(|I^F|, |J^F| − |b̄|);
13     Sortear um número inteiro r no intervalo [0, r̄];
14     Fixar r índices aleatórios de I^F em 1 e fixar demais índices de I^F em 0;
15     Fixar r + |b̄| índices aleatórios de J^F em 1 e fixar demais índices de J^F em 0;
```

Algoritmo 1: Tratamento de uma restrição da Classe 1.

2.2.3. Classe 2: soma de variáveis inteiras 0-1 menor ou igual à constante

Esta classe é composta por restrições na forma:

$$y_{i_1} + y_{i_2} + \ldots + y_{i_{|I|}} \leq b, \tag{6}$$

onde $b \geq 0$. Por se tratar de desigualdades, este caso permite que algumas variáveis sejam deixadas sem fixação. Por exemplo, considere um caso particular onde $I^F = I$, isto é, todos os índices de I não estão fixados e $b \leq |I|$. Como apenas temos que garantir que não mais do que b índices de I sejam fixados em 1, escolhemos aleatoriamente b índices em I para deixar sem fixação, fixando os demais índices em 0. Note que poderíamos ter fixado os b índices deixados "livres" em 1, mas, ao não fazê-lo, deixamos um grau maior de liberdade para o tratamento de outras restrições, ao mesmo tempo em que garantimos a satisfação da restrição em questão.

Assim, a estratégia para o caso geral dessa classe consiste em: Seja $\bar{b} = b - |I^1|$. Se $\bar{b} < 0$, declarar impossibilidade de atender a restrição. Caso contrário, escolher $\min(\bar{b}, |I^F|)$ índices aleatórios em I^F para deixar sem fixação, e fixar os demais índices em I^F em zero.

2.2.4. Classe 3

Esta classe é composta por restrições na forma:

$$y_{i_1} + y_{i_2} + \ldots + y_{i_{|I|}} - y_{j_1} - y_{j_2} - \ldots - y_{j_{|J|}} \leq b, \tag{7}$$

onde $I \neq \emptyset$ e $J \neq \emptyset$. Como na Classe 2, por se tratar de uma desigualdade, este caso permite que deixemos algumas das variáveis envolvidas sem fixação. Por exemplo, considere o caso particular onde $b = 0$, $I^1 = \emptyset$ e $J^1 = \emptyset$. Note que, nessa situação, temos:

$$y_{i_1} + y_{i_2} + \ldots + y_{i_{|I|}} \leq y_{j_1} + y_{j_2} + \ldots + y_{j_{|J|}}. \tag{8}$$

Uma forma de atender (8) seria sortear r índices tanto em I^F quanto em J^F para fixar em 1, fixando os demais índices em I^F e J^F em zero. Todavia, pode ser mais proveitoso não fixar todas as variáveis envolvidas em (8) de modo a deixar mais variáveis livres para eventualmente serem fixadas no tratamento de outras restrições. Assim, após fixar os r índices de J^F em 1, podemos deixar os demais índices de J^F livres se garantirmos que, no máximo, r índices de I estarão fixos em 1. Então, escolhemos $|I| - r$ índices de I para fixar em 0, deixando assim r índices de I livres para serem fixos posteriormente. O Algoritmo 2 descreve essa estratégia para o atendimento do caso geral dado por (7).

$\bar{b} := b - |I^1| + |J^1|$;
se $\bar{b} \geq 0$ **então**
  se $\bar{b} > |I^F|$ **então**
    Declarar impossibilidade de atender à restrição e encerrar o algoritmo;
  $\bar{r} := \min(|J^F|, |I^F| - \bar{b})$;
  Sortear um número inteiro r no intervalo $[0,\ \bar{r}]$;
  Fixar r índices aleatórios de J^F em 1 (não fixar os demais índices de J^F);
  Escolher $r + \bar{b}$ índices aleatórios de I^F para deixar livre (sem fixação) e fixar demais índices de I^F em 0;
senão
  se $|\bar{b}| > |J^F|$ **então**
    Declarar impossibilidade de atender à restrição e encerrar o algoritmo;
  $\bar{r} := \min(|I^F|, |J^F| - |\bar{b}|)$;
  Sortear um número inteiro r no intervalo $[0,\ \bar{r}]$;
  Escolher r índices aleatórios de I^F para deixar livre (sem fixação) e fixar demais índices de I^F em 0;
  Fixar $r + |\bar{b}|$ índices aleatórios de J^F em 1 (não fixar os demais índices em J^F);

Algoritmo 2: Tratamento de uma restrição da Classe 3.

2.2.5. Classe 4: soma de variáveis inteiras 0-1 maior ou igual à constante

Esta classe é composta por restrições na forma:

$$y_{i_1} + y_{i_2} + \ldots + y_{i_{|I|}} \geq b, \tag{9}$$

onde $b \geq 0$. Assim como na Classe 2, por se tratar de desigualdades, este caso permite que algumas variáveis sejam deixadas sem fixação. Aqui, só precisamos garantir que ao menos b índices de I estejam fixos em 1. Seja $\bar{b} = b - |I^1|$. Se $\bar{b} > |I^F|$, declara-se impossibilidade de atender a restrição. Caso contrário, escolhe-se aleatoriamente $\bar{b}$ índices em I^F para fixar em 1 e deixa-se os demais índices de I^F sem fixação.

2.2.6. Classe 5

Esta classe é composta por restrições na forma:

$$a_{i_1} y_{i_1} + a_{i_2} y_{i_2} + \ldots + a_{i_{|I|}} y_{i_{|I|}} \geq b, \tag{10}$$

onde $a_{i_1}, a_{i_2}, \ldots, a_{i_{|I|}}$ são constantes positivas, $\exists a_{i_j} \neq 1$ e $b \geq 0$. Como no caso anterior, no atendimento dessas restrições não fixamos variáveis em 0. Aqui, escolhemos índices aleatórios em I^F para fixar em 1 até que a restrição esteja atendida, como detalhado no Algoritmo 3.

```
1 b̄ := Σ_{i∈I^1} a_i y_i;
2 se b − b̄ > Σ_{i∈I^F} a_i y_i então
3     Declarar impossibilidade de atender à restrição e encerrar o algoritmo;
4 enquanto b̄ < b faça
5     Escolher aleatoriamente um índice i de I^F e fixá-lo em 1;
6     b̄ := b̄ + a_i y_i;
```

Algoritmo 3: Tratamento de uma restrição da Classe 5.

2.2.7. Classe 6: mochila 0-1

Esta classe é composta por restrições na forma:

$$a_{i_1} y_{i_1} + a_{i_2} y_{i_2} + \ldots + a_{i_{|I|}} y_{i_{|I|}} \leq b, \tag{11}$$

onde $a_{i_1}, a_{i_2}, \ldots, a_{i_I}$ são constantes positivas, $\exists a_{i_j} \neq 1$, e $b \geq 0$. Essa classe é tratada de forma especial pelo procedimento FEE. A cada vez que uma variável y_i é fixada em 1 no tratamento de alguma restrição em qualquer outra classe, o procedimento verifica se esta variável aparece em alguma restrição da classe 6, e, em caso positivo, verifica se a fixação de y_i em 1 permite a fixação de outras variáveis em 0 segundo a(s) respectiva(s) restrição(ões) de mochila em que y_i aparece. Por exemplo, considere o seguinte caso particular:

$$2y_1 + 3y_2 + 2y_3 + y_4 \leq 4\ . \tag{12}$$

Observando (12), podemos notar que se y_1 ou y_3 forem fixas em 1, y_2 deverá ser fixa em 0. Por outro lado, se y_2 for fixa em 1, ambas y_1 e y_3 deverão ser fixas em zero. Se apenas y_4 for fixa em 1, nenhuma outra variável deve ser fixa com base nessa restrição, mas, por outro lado, se y_1 e y_4 forem fixas em 1, tanto y_2 quanto y_3 deverão ser fixas em 0.

2.2.8. Algoritmo de Fixação Estocástica Estruturada

Tendo apresentado as classes especiais de restrições consideradas e seus respectivos tratamentos, estamos aptos a apresentar, no Algoritmo 4, o procedimento geral de FEE.

O Algoritmo 4 trata as restrições por classe, isto é, todas as restrições de uma classe são tratadas antes de se avançar para uma nova classe. De modo a aumentar a diversidade na busca por

Entrada: (P): problema abordado, l_y: limite inferior inicial de y, u_y: limite superior inicial de y.
Saída: $status$: status da aplicação do algoritmo, l_y: limite inferior final de y, u_y: limite superior final de y.

```
1 para cada k ∈ {0,...,5} faça
2     para cada restrição j pertencente à classe k faça
3         se a restrição j não puder ser tratada com sucesso então
4             Declarar falha e encerrar o algoritmo;
5         Tratar a restrição j segundo regras da classe k atualizando l_y e u_y;
6         (l_y, u_y, status) := preprocessamento((P), l_y, u_y);
7         se status = inviabilidade então Declarar falha e encerrar o algoritmo ;
8 Declarar sucesso e encerrar o algoritmo;
```

Algoritmo 4: Fixação Estocástica Estruturada (FEE).

soluções, pode ser interessante considerar as classes de restrições em ordem aleatória a cada execução. Nesta mesma linha de raciocínio, também pode ser vantajoso tratar as restrições dentro de cada classe em ordem aleatória, o que também minimiza os efeitos da ordem em que as restrições são formuladas no problema. Observamos que as restrições de mochila 0-1 (Classe 6) são tratadas a cada vez em que alguma variável é fixada em 1 e também pelo pré-processamento executado na linha 6, cujo objetivo é descobrir fixações de variáveis após o tratamento de cada restrição. Também é válido apontar que o pré-processamento pode detectar inviabilidade após a fixação de um subconjunto de variáveis, indicando que o respectivo ajuste não pode conduzir à solução viável. Nessa situação, encerramos o algoritmo declarando falha na linha 7. Por fim, ressaltamos novamente que sempre que um índice qualquer é fixado em algum valor nesse procedimento, este não pode mais ser desfixado de modo a garantir o atendimento de todas as restrições já tratadas.

2.3. Arredondamento Estocástico

O procedimento de arredondamento estocástico é aplicado após a FEE com o objetivo de determinar valores para variáveis inteiras que ainda estejam desfixadas após a aplicação deste último. Aqui, esse arrendondamento é feito tomando por base uma probabilidade definida pelo resíduo decimal de uma solução (não inteira) para a relaxação contínua de (P), chamada aqui como *solução de referência*. Qualquer solução para relaxação contínua que respeite às restrições de caixa de y pode ser utilizada, inclusive uma solução inviável.

Por exemplo, se uma solução de referência $\tilde{y}$ apresenta $\tilde{y_1} = 0{,}3$, podemos usar esse valor de $0{,}3$ como a probabilidade de y_1 ser arredondada para cima (ao passo que a probabilidade do arredondamento para baixo seria de $0{,}7$).A finalidade desta estratégia é tentar aproximar a solução inteira obtida da própria solução de referência, que, idealmente, é uma solução de boa qualidade para a relaxação contínua de (P), sem perder o fator estocástico, que pode ser importante para manter a diversidade da exploração. Nesse contexto, uma boa sugestão para a solução de referência poderia ser uma solução ótima da relaxação contínua de (P).

Pode ser interessante garantir uma probabilidade mínima ϕ (por exemplo, $0{,}1$) para uma solução ser arredondada para cima ou para baixo. Assim, o Algoritmo 5 (Arredondamento Estocástico) considera essa probabilidade ϕ para realizar o arredondamento. Na linha 7, o procedimento $aleatorio(0, 1)$ gera um número real aleatório no intervalo $[0,\ 1)$. Observe que, após fixar uma variável inteira não fixa y_i (linha 8), o procedimento utiliza a rotina de pré-processamento (linha 9) para descobrir fixações obrigatórias em decorrência da fixação de y_i. Devido a isso, pode ser

vantajoso percorrer as variáveis inteiras em ordem aleatória para um maior nível de diversidade na busca e para minimizar a influência da ordem em que as variáveis são apresentadas no modelo de otimização.

Entrada: (P): problema abordado, $\tilde{y}$: solução de referência, l_y: limites inferiores correntes para y, u_y: limites superiores correntes para y.
Saída: $status$: status da aplicação do algoritmo, $\bar{y}$: solução inteira para y (não necessariamente viável).

para cada $i \in \{1, 2, \ldots, n_y\}$ **faça**
  se $l_{y_i} = u_{y_i}$ **então**
    $\bar{y}_i := l_{y_i}$; `// y_i já está fixa`
  senão
    $\overline{p} := \max(\tilde{y}_i - \lfloor \tilde{y}_i \rfloor, \phi)$;
    $\overline{p} := \min(\overline{p}, 1 - \phi)$;
    $r := aleatorio(0, 1)$; `// gera aleatório real entre 0 e 1`
    se $r < \overline{p}$ **então** $\bar{y}_i := 1$ **senão** $\bar{y}_i := 0$;
    $(l_y, u_y, status) := preprocessamento((P), l_y, u_y)$;
    se $status$ = *inviabilidade* **então** Declarar falha e encerrar o algoritmo;
Declarar sucesso e encerrar o algoritmo;

Algoritmo 5: Arredondamento Estocástico.

2.4. Busca Local

Note que FEE e arredondamento estocástico não levam em conta a função objetivo de (P), o que pode conduzi-los a soluções viáveis ruins. De modo a remediar essa inconveniência, após a construção de uma solução inteira $\bar{y}$, AEE usa um esquema de busca local para melhoramento de soluções. A ideia do procedimento de busca local é resolver um novo problema de programação inteira derivado de (P) que só permita soluções viáveis na vizinhança de $\bar{y}$. Para variáveis inteiras 0-1, podemos incorporar uma restrição de k-vizinhança, onde limitamos a quantidade de componentes diferentes de $\bar{y}$ que uma solução pode apresentar. Para a definição dessa nova restrição, convém definir $\bar{I}^1$ como o conjunto de índices de $\bar{y}$ com o valor 1, isto é, $\bar{I}^1 = \{i | \bar{y}_i = 1\}$ e $\bar{I}^0$ como o conjunto de índices de $\bar{y}$ com o valor 0, isto é, $\bar{I}^0 = \{i | \bar{y}_i = 0\}$. Assim, nossa restrição de k-vizinhança pode ser escrita como:

$$\sum_{i \in \bar{I}^1} (1 - y_i) + \sum_{i \in \bar{I}^0} y_i \leq k \tag{13}$$

Note que (13) permite que, no máximo, k componentes de uma solução y sejam diferentes dos respectivos componentes de $\bar{y}$. Uma ideia para o ajuste de k é calculá-lo em termos da dimensão de y, como, por exemplo, $k = \lceil 0{,}3 n_y \rceil$, onde permite-se soluções com até 30% dos componentes diferentes dos de $\bar{y}$. O uso de k-vizinhança é bastante difundido em heurísticas para programação inteira, como, por exemplo, *Local Branching* [Fischetti and Lodi, 2003]. Assim, a fase de busca local resolve um problema (P^S) composto da função objetivo e restrições de (P) acrescido de (13):

$$\begin{aligned} (P^S) \quad \min_{x,\,y} \quad & f(x, y) \\ \text{s. t.} \quad & g(x, y) \leq 0, \\ & A^y y \, \{\leq, =, \geq\} \, b, \\ & \sum_{i \in I^1} (1 - y_i) + \sum_{i \in I^0} y_i \leq k, \\ & x \in \mathbb{R}^{n_x}, \; y \in \{0{,}1\}^{n_y}, \end{aligned} \tag{14}$$

Qualquer método apropriado pode ser usado para resolver (P^S). Na realidade, não é preciso resolver (P^S) até a otimalidade. Para não gastar demasiado tempo tratando (P^S), pode-se definir um alto valor para o *gap* de otimalidade relativa, como, por exemplo $0{,}3$.

2.5. Algoritmo Arredondamento Estocástico Estruturado

O Algoritmo 6 traz o algoritmo de Arredondamento Estocástico Estruturado (AEE). O algoritmo resolve primeiramente a relaxação contínua de (P) na linha 2, com o objetivo de obter valores iniciais $\tilde{y}$ que serão utilizados para definição de probabilidades no procedimento de arredondamento estocástico (linha 10). Alternativamente, poderíamos substituir a resolução desta relaxação pelo ajuste de $\tilde{y}$ em algum valor arbitrário no intervalo $[0\ 1]^{n_y}$.

A partir daí, são realizadas até $MaxTries$ tentativas de obtenção de uma solução viável. De modo a facilitar a fixação de variáveis, o algoritmo mantém, em cada tentativa, os vetores l_y e u_y (inicializados nas linhas 6 e 7) para armazenar os limites inferiores e superiores das variáveis inteiras. Na linha 8, é realizado o procedimento de FEE (Algoritmo 6), que fixa variáveis inteiras atendendo às classes de restrições apresentadas na Seção 2.2.

Caso o procedimento FEE (linha 8) não resulte em falha, AEE prossegue fixando as variáveis inteiras ainda não fixadas. Isto é feito na linha 10 através do procedimento de arredondamento estocástico (Algoritmo 5). Assim, se não houver falha na execução da linha 10, o algoritmo AEE tem um valor inteiro inicial $\bar{y}$ para y. O passo seguinte é resolver o problema $(P_{\bar{y}})$ (linha 14), que é o subproblema de (P) obtido ao se fixar y em $\bar{y}$. Note que, se $(P_{\bar{y}})$ for viável e, tomando por $\bar{x}$ sua solução ótima, temos que $(\bar{x}, \bar{y})$ será a melhor solução para (P) tendo $\bar{y}$ como valor para y (ou uma das melhores, caso $(P_{\bar{y}})$ tenha múltiplas soluções ótimas). A resolução de $(P_{\bar{y}})$ é comumente empregada em algoritmos para Programação Não Linear Inteira Mista (para mais detalhes, consulte [Melo et al., 2020b]) .

É válido ressaltar que esse primeiro valor $\bar{y}$ é obtido sem levar em conta a função objetivo do problema. Por essa razão, a qualidade da solução obtida na linha 14 pode ser ruim. Assim, o algoritmo AEE passa a um ciclo de melhoramento de soluções. Tomando por $(\bar{x}, \bar{y})$ a melhor solução conhecida pelo algoritmo, o algoritmo realiza busca local por uma solução melhor na vizinhança de $\bar{y}$, conforme discutido na Seção 2.4.

Por fim, embora AEE seja um algoritmo estocástico, pode-se usá-lo de forma determinística através do controle da semente de geração dos números pseudo-aleatórios demandados.

3. Resultados Computacionais

Apresentamos aqui resultados computacionais da aplicação de AEE sobre um conjunto de 226 problemas convexos de Programação Não Linear Inteira 0-1 Mista (PNLIM 0-1) de [CMU-IBM, 2012, Leyffer, 2013, World, 2014]. Informações gerais sobre os problemas testes podem ser conferidas na Tabela 1. Comparamos o desempenho de AEE com o de 5 heurísticas: *Feasibility Pump (FP)* [Bonami and Gonçalves, 2008], *Heurística de Mergulho, versão fracionária (Diving)* [Bonami and Gonçalves, 2008], *Feasibility Pump baseado em aproximação externa versão 1 (OA-FP*), [Bonami et al., 2009], *Algoritmo de Minimização do Gap de Integralidade, versões 1 (IGMA1) e 2 (IGMA2)* ambas de [Melo et al., 2018].

IGMA1 e IGMA2 foram propostos para problemas inteiros 0-1. As demais abordagens podem tratar problemas inteiros gerais. O algoritmo AEE foi executado permitindo apenas uma melhora de solução na busca local. O algoritmo de PNLIM usado nessa busca foi *Branch-And-Bound* baseado em Programação Linear [Melo et al., 2020a]. Para a restrição de k-vizinhança, adotamos $k = \lceil 0{,}3 n_y \rceil$. O parâmetro ϕ, a probabilidade mínima de arredondamento, foi ajustado em $0{,}1$. O parâmetro $MaxTries$ foi ajustado em 10000.

Entrada: (P): problema abordado, $MaxTries$: número máximo de tentaivas de obtenção de solução viável, $MaxImprovments$: número máximo de melhoramentos.
Saída: $(\bar{x}, \bar{y})$: solução viável para (P) (ou falha).

Sejam l_y e u_y vetores com limites inferiores e superiores, respectivamente, para y;
Seja $(\tilde{x}, \tilde{y})$ a solução da relaxação contínua de (P);
$i := 0$;
enquanto $i < MaxTries$ **faça**
    $i := i + 1$;
    $l_y := [0]^{n_y}$;
    $u_y := [1]^{n_y}$;
    $(status, l_y, u_y) := fixacaoEstocasticaEstruturada((P), l_y, u_y)$;
    se *status* = *falha* **então** Vá para a linha 4;
    $(status, \bar{y}) := arredondamentoEstocastico((P), \tilde{y}, l_y, u_y)$;
    se *status* = *falha* **então** Vá para a linha 4;
    Seja $(P_{\bar{y}})$ o subproblema de (P) obtido ao se fixar y em $\bar{y}$;
    se *Se* $(P_{\bar{y}})$ *é inviável* **então** Vá para a linha 4;
    Seja $(\bar{x})$ uma solução ótima de $(P_{\bar{y}})$;
    $j := 0$;
    enquanto *j* < *MaxImprovments* **faça**
        $j := j + 1$;
        $(\bar{x}, \bar{y}, status) := buscaLocal(\bar{x}, \bar{y})$;
        se $status = sem\ melhora$ **então** Vá para a linha 20;
    retorna $(\bar{x}, \bar{y})$;
Declarar falha e encerrar o algoritmo;

Algoritmo 6: Arredondamento Estocástico Estruturado.

Tabela 1: Estatísticas sobre problemas testes.

	min	**max**	**média**	**mediana**
Variáveis	4	107222	1435,99	252,5
Variáveis inteiras 0-1	2	1500	125,53	60
Restrições	1	108217	1747,96	354,5
Restrições lineares	1	108217	1728,45	330

Os testes rodaram em uma máquina com 16GB de RAM e processador Intel Xeon E5-2680 V3 3.1 Ghz com o sistema operacional Mint Linux 20.0 em *thread* de execução única com tempo máximo de CPU de 600 segundos por instância por algoritmo. Implementamos todos os algoritmos mencionados em C++. Para resoluções de subproblemas de Programação Não Linear, foi usado Ipopt 3.13.2 [Wächter and Biegler, 2006]. Para a resolução de problemas de Progamação Linear Inteira Mista, foi utilizado Cplex 12.10 [IBM, 2015].

A seguir, apresentamos os gráficos de *performance profile* para nossos testes. Além do desempenho dos algoritmos mencionados, consideramos o desempenho de AEE em dois aspectos, chamados de AEE e AEE - 1ª Sol. O primeiro aspecto (AEE) apresenta o desempenho do algoritmo em sua totalidade, considerando a solução final apresentada, que passa pela etapa de busca local com a resolução do problema (P^S). O segundo (AEE - 1ª Sol) considera apenas o desempenho com relação a primeira solução obtida por AEE, que é aquela obtida antes da etapa de busca local. O objetivo desta última configuração é avaliar a solução obtida apenas com os procedimentos de

Fixação Estocástica Estruturada e Arredondamento Estocástico, seguido da resolução do problema $(P_{\bar{y}})$. A Figura 1 apresenta comparação relativa a melhor solução encontrada pelas abordagens:

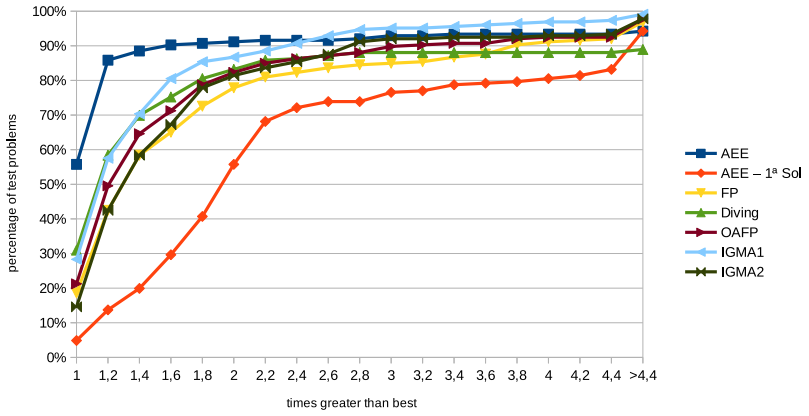


Figura 1: Comparação relativa de heurísticas para PNLIM 0-1 - Melhor solução.

Observando a Figura 1, podemos observar que AEE obteve os melhores resultados quando se considera soluções até $2{,}4$ vezes piores que o melhor resultado. AEE - 1ª Sol, por sua vez, teve desempenho acentuadamente inferior ao das demais heurísticas, o que, de certo modo, já era esperado, pois a primeira solução obtida por AEE não leva diretamente em conta a função objetivo do problema abordado. A comparação dos desempenhos de AEE e AEE - 1ª Sol demonstra o impacto do procedimento de busca local na heurística AEE.

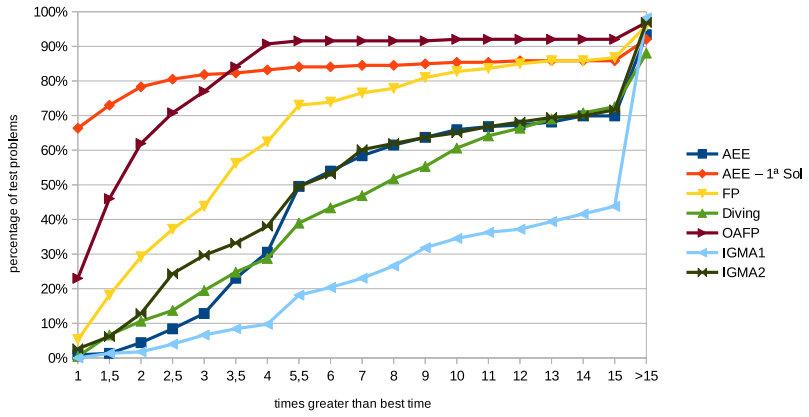


Figura 2: Comparação relativa de heurísticas para PNLIM 0-1 - Tempo Computacional.

A Figura 2 apresenta a comparação dos algoritmos quanto ao tempo computacional despendido (tempo de processamento em CPU). Analisando essa figura, podemos observar que AEE - 1ª Sol apresenta menor tempo computacional para 66% das instâncias de teste, o melhor resultado dentre as heurísticas avaliadas. A curva de AEE - 1ª Sol apresenta o melhor desempenho para tempo computacional de até $3{,}4$ maior do que o melhor resultado, sendo então superada, a partir daí, pela curva de desempenho de OA-FP. Por sua vez, o desempenho de AEE é competitivo ao de IGMA1, IGMA2 e Diving, o que evidencia o impacto da busca local de AEE em termos do

tempo computacional. A partir das Figuras 1 e 2, nota-se que AEE e AEE - 1ª Sol estão em lados opostos com relação aos conceitos de eficiência e eficácia. AEE - 1ª Sol é mais indicada que AEE quando o tempo computacional é mais crítico que a qualidade da solução (maior eficiência). AEE, por sua vez, é mais indicada quando a qualidade da solução é mais importante que a urgência em obtê-la (maior eficácia). AEE - 1ª Sol pode ser uma boa heurística a ser incorporada em *Branch-and-Bound* (BB) para PNLIM 0-1, uma vez que, além de não ser cara, ela pode se valer da solução da relaxação contínua nas partições do espaço, que já precisa ser calculada pelo próprio BB, o que aceleraria AEE ainda mais. Um baixo valor para $MaxTries$ pode ser adotado nessa utilização integrada ao BB. Em nossos testes, AEE demandou em média 101 tentativas para obter a primeira solução viável nos casos de sucesso. O valor da mediana para esse número de tentativas até a primeira solução viável é de apenas 1. As implementações aqui utilizadas estão disponíveis no pacote `Muriqui Optimizer` [Melo et al., 2014, 2020b].

Referências

Pierre Bonami and João Gonçalves. Heuristics for convex mixed integer nonlinear programs. *Computational Optimization and Applications*, pages 1–19, 2008. ISSN 0926-6003. 10.1007/s10589-010-9350-6.

Pierre Bonami, Gérard Cornuéjols, Andrea Lodi, and François Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119:331–352, 2009. ISSN 0025-5610. 10.1007/s10107-008-0212-2.

CMU-IBM. Open source minlp project, http://egon.cheme.cmu.edu/ibm/page.htm, 2012. URL `http://egon.cheme.cmu.edu/ibm/page.htm`.

Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003. ISSN 0025-5610. 10.1007/s10107-003-0395-5.

IBM. *IBM ILOG CPLEX V12.6 User's Manual for CPLEX*, 2015.

Sven Leyffer. Macminlp: Test problems for mixed integer nonlinear programming, 2003. https://wiki.mcs.anl.gov/leyffer/index.php/macminlp, 2013.

Wendel Melo, Marcia Fampa, and Fernanda Raupp. Integrating nonlinear branch-and-bound and outer approximation for convex mixed integer nonlinear programming. *Journal of Global Optimization*, 60(2): 373–389, 2014. ISSN 0925-5001. doi: 10.1007/s10898-014-0217-8.

Wendel Melo, Marcia Fampa, and Fernanda Raupp. Integrality gap minimization heuristics for binary mixed integer nonlinear programming. *Journal of Global Optimization*, 71(3):593–612, 2018. ISSN 1573-2916. doi: 10.1007/s10898-018-0623-4.

Wendel Melo, Marcia Fampa, and Fernanda Raupp. Two linear approximation algorithms for convex mixed integer nonlinear programming. *Annals of Operations Research*, pages 1–21, 2020a. ISSN 1572-9338. doi: 10.1007/s10479-020-03722-5.

Wendel Melo, Marcia Fampa, and Fernanda Raupp. An overview of minlp algorithms and their implementation in muriqui optimizer. *Annals of Operations Research*, 286(1):217–241, May 2020b. ISSN 1572-9338. doi: 10.1007/s10479-018-2872-5.

M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994. doi: 10.1287/ijoc.6.4.445.

Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006. ISSN 0025-5610. 10.1007/s10107-004-0559-y.

GAMS World. Minlp library 2, http://www.gamsworld.org/minlp/minlplib2/html/, 2014.