Marcos Daniel V. Baroni

A Hybrid Heuristic for the Multi-objective Knapsack Problem

Vitória - Espírito Santo - Brasil
November 27, 2017

Marcos Daniel V. Baroni

A Hybrid Heuristic for the Multi-objective Knapsack Problem

Tese de Doutorado apresentada de acordo com o regimento do Programa de Pós-graduação em Informática da Universidade Federal do Espírito Santo.

Universidade Feredal do Espírito Santo – UFES
Departamento de Informática
Programa de Pós-Graduação em Informática

Advisor: Dr. Flávio Miguel Varejão

Vitória - Espírito Santo - Brasil
November 27, 2017

Marcos Daniel V. Baroni

A Hybrid Heuristic for the Multi-objective Knapsack Problem

Tese de Doutorado apresentada de acordo com o regimento do Programa de Pós-graduação em Informática da Universidade Federal do Espírito Santo.

Work approved. Vitória - Espírito Santo - Brasil, November 27, 2017:

Dr. Flávio Miguel Varejão
Advisor

Dr.ª Maria Claudia Silva Boeres
Member

Dr. Arlindo Gomes de Alvarenga
Guest

Dr.ª Simone de Lima Martins
Guest

Vitória - Espírito Santo - Brasil
November 27, 2017

Resumo

Many real applications like project selection, capital budgeting and cutting stock involves optimizing multiple objectives that are usually conflicting and can be modelled as a multi-objective knapsack problem (MOKP). Unlike the single-objective case, the MOKP is considered a NP-Hard problem with considerable intractability. This work propose a hybrid heuristic for the MOKP based on the shuffled complex evolution algorithm. A multi-dimensional indexing strategy for handling large amount of intermediate solutions are proposed as an optimization, which yields considerable efficiency, especially on cases with more than two objectives. A series of computational experiments show the applicability of the proposal to several types of instances.

Keywords: Multi-objective Knapsack Problem, Metaheuristic, Shuffled Complex Evolution, Multi-dimensional indexing

Contents

1 Introdução 7

2 O Problema da Mochila Multi-objetivo 9

2.1 Métodos Exatos 11

2.1.1 As relações de dominância 12

2.1.2 Geração de conjuntos cobertura e independente 14

2.2 Detalhes de implementação 16

2.2.1 Ordem dos itens 16

2.2.2 Relações de dominância 16

2.2.3 Detalhes de implementação 20

2.3 Métodos Heurísticos 22

2.4 Evolução estocástica por complexos 22

2.5 O EEC para o MOKP 23

3 A k-d tree 25

3.1 Verificação de dominância e a busca de faixa 25

4 Experimentos computacionais 29

5 Conclusão 31

References 33

1 Introdução

Intro...

2 O Problema da Mochila Multi-objetivo

Em problemas reais é comum a existência de situações em que deseja-se otimizar mais de um objetivo os quais, geralmente, são conflitantes. Estes problemas são chamados multi-objetivos e tipicamente não possuem uma solução sendo a melhor em todos os objetivos, mas as possuem várias soluções de interesse chamadas *soluções eficientes*.

Um problema de otimização multi-objetivo com m objetivos pode ser descrito como uma função vetorial $f(x) = \Big(f_1(x), \ldots, f_p(x)\Big)$ para a qual deseja-se encontrar um vetor $x \in X$ que maximize simultaneamente as m funções objetivo. Formalmente:

$$\begin{aligned} &\max\ f(x) = \Big(f_1(x), f_2(x), \ldots, f_m(x)\Big) \\ &\text{sujeito a } x \in X \end{aligned}$$

Definição 1 (Dominância, Eficiência e conjunto Pareto)**.** *Considere um problema de otimização multi-objetivo. Diz-se que uma solução $x \in X$* domina *uma solução $y \in X$, denotado por $dom(x, y)$ se, e somente se, x é ao menos tão boa quanto y em todos os objetivos e melhor que y em ao menos um dos objetivos. Formalmente:*

$$dom(x, y) = \begin{cases} \forall i \in \{1, 2, \ldots, m\} : f_i(x) \geqslant f_i(y)\ e \\ \exists j \in \{1, 2, \ldots, m\} : f_j(x) > f_j(y) \end{cases} \tag{2.1}$$

Uma solução $x \in X$ é dita eficiente*, denotado por $\mathit{eff}(x)$, se, e somente se, x não é dominada por nenhuma outra solução pertencente a X. Formalmente:*

$$eff(x) \iff \nexists\Big(y \in X \wedge dom(y, x)\Big)$$

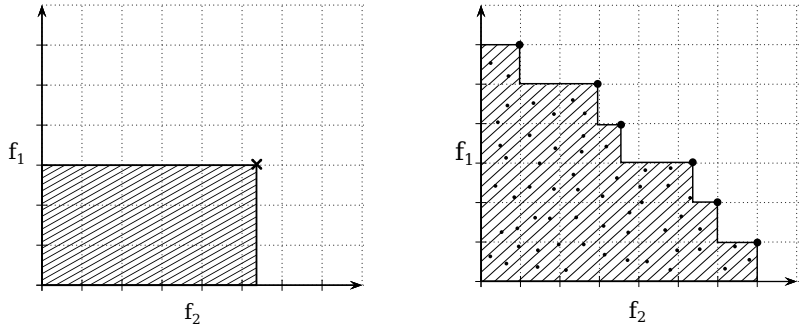
O conjunto de todas as soluções eficientes de um problema multi-objetivo, denotado por $Par(X)$, é chamado de conjunto Pareto *ou* conjunto Pareto-ótimo*. Formalmente:*

$$Par(X) = \{x \in X \mid \mathit{eff}(x)\}$$

Resolver um problema multi-objetivo consiste em determinar seu conjunto Pareto. Este conceito foi primeiramente elaborado por Vilfredo Pareto em 1896, que enunciou a relação Pareto-Ótima que diz: "não é possível melhorar uma característica do problema sem piorar outra", o que caracteriza a relação conflitante entre os objetivos na otimização multi-objetivo.

Na Figura 1a ilustra o conceito de dominância. A solução marcada domina todas as soluções existentes na área hachurada. As soluções em destacadas na Figura 1b formam um conjunto Paretopor dominarem sobre todas as outras soluções.

Um dos problemas multiobjetivos mais importantes da literatura é o problema da mochila multiobjetivo (MOKP). Muitas problemas reais podem ser modelados como uma



(a) Região de dominância de uma solução.

(b) Exemplo de conjunto Pareto.

Figura 1: Exemplos de solução dominante e conjunto Pareto.

instância do MOKP como seleção de projetos (TENG; TZENG, 1996), orçamento de capital (ROSENBLATT; SINUANY-STERN, 1989), carregamento de carga (TENG; TZENG, 1996) e planejamento de estoque (ISHIBUCHI; AKEDO; NOJIMA, 2015).

Comentar sobre a dificuldade de problemas MObj. Explosão do pareto com o aumento da quantidade de objectivos. Poucos métodos extados eficientes, geralmente utiliza-se métodos heurísticos.

O problema da mochila multi-objetivo pode ser descrito como uma função vetorial f que mapeia uma variável de decisão (solução) a uma tupla de m valores (objetivos). Formalmente:

$$\begin{aligned} \max\ & y = f(x) = \Big(f_1(x), f_2(x), \ldots, f_m(x)\Big) \\ \text{sujeito a}\ & x \in X \end{aligned}$$

onde x é a *variável de decisão*, X denota o conjunto de soluções viáveis e y representa o *vetor de objetivos* para os quais deseja-se maximizar.

Vale resaltar que o tamanho do conjunto Paretopara o problema em questão tende a crescer rapidamente com o tamanho do problema, especialmente com o número de objetivos.

Uma instância de um problema da mochila multi-objetivo (MOKP) com m objetivos consiste em uma capacidade inteira $W > 0$ e n itens. Cada item i possui um peso inteiro positivo w_i e m lucros inteiros $p_i^1, p_i^2, \ldots, p_i^m$ não negativos. O lucro p_i^k representa a contribuição do i-ésimo item para com o k-ésimo objetivo. Uma solução é representada por um conjunto $x \subseteq \{1, \ldots, n\}$ contendo os índices dos itens incluídos na mochila. Uma solução é viável se o peso total incluído na mochila não ultrapassa a capacidade da

mochila. Formalmente a definição do problema é a seguinte:

$$\begin{aligned} \max\ & f(x) = \Big(f_1(x), f_2(x), \ldots, f_m(x)\Big) \\ \text{subject to } & w(x) \leqslant W \\ & x \in \{0,1\}^n \end{aligned}$$

where

$$f_j(x) = \sum_{i=1}^{n} p_i^j x_i \quad j = 1, \ldots, m$$
$$w(x) = \sum_{i=1}^{n} w_i x_i$$

O MOKP é considerado um problema $\mathcal{NP}$-Hard visto set uma generalização do bem conhecido problema da mochila $0-1$, para o qual $m = 1$. É consideravelmente difícil determinar o conjunto Paretopara um MOKP, especialmente para vários objetivos. Até mesmo para casos bi-objetivos, problemas pequenos podem se apresentar intratáveis. Por este motivo interessa-se no desenvolvimento de métodos eficientes para manipular uma grande quantidade de soluções, o que pode eventualmente trazer tratabilidade a instâncias antes intratáveis.

A literatura contém várias propostas para resolver o MOKP de forma exata. Porém, nenhum método tem provado ser eficiente para grande instâncias com mais de dois objetivos. Mesmo para problemas bi-objetivo, algumas instâncias de tamanho considerado médio têm aprestando difculdades na determinação da solução exata, o que tem motivado o desenvolvimento de métodos heurísticas que buscam determinar um conjunto Paretoaproximado em tempo computacional razoável.

2.1 Métodos Exatos

Comentar sobre background de propostas de métodos exatos.

Comentar sobre o método da Bazgan como sendo considerado o melhor, citar melhorias propostas.

Explicar que o algoritmo DP considera um conceito generalizado de dominância aplicado a cada iteração.

Dizer que a explicação do desenvolvimento do algoritmo será reproduzido segundo o artigo original.

O processo sequencial executado pelo algoritmo de programação dinâmica consiste de n iterações. A cada k-ésima iteração é gerado o conjunto de estados S^k, que representa

Algoritmo 1: O algoritmo de Nemhauser e Ullmann para o MOKP.

input: $\boldsymbol{p}, \boldsymbol{w}, W$

begin
  $S^0 = \{(0, \ldots, 0)\}$;
  for $k \leftarrow 1, n$ **do**
    $S_k^0 \leftarrow S_{k-1} \cup \left\{(s_{k-1}^1 + p_k^1, \ldots, s_{k-1}^m + p_k^m, s_{k-1}^{m+1} + w_k) \mid s_{k-1}^{m+1} + w_k \leqslant W,\ s_{k-1} \in S_{k-1}\right\}$;
  end
  $P = \{s \in S^n \mid \nexists (a \in S^n \mid a \underline{\Delta} s),\ s^{m+1} \leqslant W\}$;
  return P;

todas as soluções viáveis compostas de itens exclusivamente pertencentes aos k primeiros itens ($k = 1, \ldots, n$). Um estado $s_k = (s_k^1, \ldots, s_m^k, s_{m-1}^k) \in S_k$ represena uma solução viável que tem valor s_k^i como i-ésimo objetivo ($i = 1, \ldots, m$) e s_k^{m-1} de peso. Portanto, temos $S_k = S_{k-1} \cup \{(s_{k-1}^1 + p_k^1)\}$

Comentar sobre as estratégias de redução dos conjuntos de estados, motivando as definições a seguir.

Definir conjunto cobertura, conjunto independente, conjunto eficiente reduzido, etc., definir parte simétrica/assimétrica de $\underline{\Delta}$?

Definição 2 (Extensão, Restrição e Complemento)**.** *Considere o Algoritmo 1 e qualquer estado $s_k \in S_K (k < n)$. Um* complemento *de s_k é qualquer subconjunto $J \subseteq \{k+1, \ldots, n\}$ tal que $s_k^{m+1} + \sum_{j \in J} w_j \leqslant W$. Assumiremos que qualquer estado $s_n \in S_n$ admite o conjunto vazio como único complemento. Um estado $s_n \in S_n$ é uma* extensão *de $s_k \in s_k (k \leqslant n)$ se, e somente se, existe um complemento I de s_k tal que $s_n^j = s_k^j + \sum_{i \in I} p_i^j) para\ j = 1, \ldots, m$ e $s_n^{p+1} = s_k^{p+1} + \sum_{i \in I} w_i$. O conjunto de extenções de s_k é denotado por $Ext(s_k) (k \leqslant n)$. Um estado $s_k \in S_k (k \leqslant n)$ é uma* restrição *do estado $s_n \in S_n$ se, e somente se, s_n é uma extensão de s_k.*

2.1.1 As relações de dominância

Introdução às relações de dominância?

Definição 3 (Relação de dominância entre soluções)**.** *Uma relação D_k sobre $S_k, k = i,\ ldots, n,$ é uma* relação de dominância *se, e somente se, para todo $s_k, s_{k'} \in S_k$,*

$$s_k D_k s_{k'} \Rightarrow \forall s_{n'} \in Ext(s_{n'}), \exists s_n \in Ext(s_k), s_n \underline{\Delta} s_{n'} \tag{2.2}$$

Apesar das relações de dominância não serem transitivas por definição, costumam ser transitivas por construção, como é o caso das três relações de dominância da Seção 2.1.1. Vale notar que se $D_k^i,\ i = 1, \ldots, p$ são relações de dominância então $D_k = \bigcup_{i=1}^{p} D_k^i$ é também uma relação de dominância, geralmente não transitiva mesmo se $D_k^i,\ i = 1, \ldots, p$ forem transitivas.

Para se ter uma implementação eficiência do algoritmo de programação dinâmica é recomendável utilizar múltiplas relações de dominância $D_k^1, \ldots, D_k^p (p \leqslant 1)$ a cada execução da k-ésima iteração $k = 1, \ldots, n$ uma vez que cada relação D_k^i explora características específicas.

Algoritmo 2: Programação dinâmica utilizando múltiplas relações de dominância.

```
1 begin
2     C^0 ← {(0, ..., 0)};
3     for k ← 1, n do
4         C_k^0 ← C_{k-1} ∪ {(s_{k-1}^1 + p_k^1, ..., s_{k-1}^m + p_k^m, s_{k-1}^{m+1} + w_k) | s_{k-1}^{m+1} + w_k ⩽
              W, s_{k-1} ∈ C_{k-1}};
5         for i ← 1, p do
6             Determinar um conjunto C_k^i cobertura do conjunto C_{k-1}^i com respeito
                a D_k^i;
7         C_k ← C_k^p;
8     return C_n;
```

Proposição 1. *Para quaisquer relações de dominância $D_k^1, \ldots, D_k^p (p \geqslant 1)$ sobre S_k, o conjunto C_k^p obtido pelo Algoritmo 2 em cada iteração é um conjunto cobertura de C_k^0 com respeito a $D_k = \bigcup_{i=1}^{p} D_k^i$ $(k = 1, \ldots, n)$.*

Demonstração. Considere $s_k \in C_k^0 \backslash C_k^p$, este foi removido quando selecionado um conjunto cobertura na iteração da linha 5. Seja $i_1 \in \{1, \ldots, p\}$ a iteração da linha 5, tal que $s_k \in C_k^{i_1 - 1} \backslash C_k^{i_1}$. Uma vez que $C_k^{i_1}$ é um conjunto cobertura de $C_k^{i_1 - 1}$ com respeito a $D_{k'}^{i_1}$, existe $s_{k'}^{(1)} \in C_k^{i_1}$ tal que $s_{k'}^{(1)} D_k^{i_1} s_k$. Se $s_{k'}^{(1)} \in C_k^m$ então as propriedades de cobertura são válidas, uma vez que $D_k^{i_1} \subseteq D_k$. Caso contrário, existe uma iteração $i_2 > i_1$, correspondente à iteração da linha 5 tal que $s_{k'}^{(1)} \in C_k^{i_2 - 1} \backslash C_k^{i_2}$. Como anteriormente, estabelecemos que existe $s_{k'}^{(2)} \in C_k^{i_2}$ tal que $s_{k'}^{(2)} D_k^{i_2} s_{k'}^{(1)}$. Uma vez que $D_k^{i_2} \subseteq D_k$ temos que $s_{k'}^{(2)} D_k s_{k'}^{(1)} D_k s_k$ e por transitividade de D_k garantimos que $s_{k'}^{(2)} D_k s_k$. Pela repetição desde processo podemos garantir que a existência de um estado $s_{k'} \in C_k^m$, tal que $s_{k'} D_k s_k$. □

Temos agora condições para garantir que o Algoritmo 2 gera o conjunto de vetores objetivos não dominados.

Teorema 1. *Para qualquer família de relações de dominância $D_k^i (i = 1, \ldots, p; k = 1, \ldots, n)$, o Algoritmo 2 retorna C_n que é um conjunto cobertura de S_n com respeito*

a $\underline{\Delta}$*. Além disso, se na iteração n utilizarmos ao menos uma relação* $D_n^i = \underline{\Delta}$ *e garantirmos que o conjunto cobertura* C_n^i *é também independente com respeito a* D_n^i *então* C_n *representa o conjunto* ND *de vetores objetivos não dominados.*

Demonstração. Considere $s_{n'} \in S_n \backslash C_n$, todas as restrições foram removidas ao reter-se o conjunto cobertura com respeito a $D_k = \cup_{i=1}^{m} D_k^i$ durante as iterações $k \leqslant n$. Seja k_1 a iteração mais alta em que $C_{k_1}^0$ ainda contém restrições de $s_{n'}$, as quais serão removidas aplicando uma das relações $D_{k_1}^i (i = 1, \ldots, p)$. Considere alguma destas restrições, denotada por $s_{k_1'}^{(n)}$. Uma vez que $s_{k_1'}^n \in C_{k_1}^0 \backslash C_{k_1}$ sabemos, pela Proposição 1, que existe $s_{k_1} \in C_{k_1}$ tak que $s_{k_1} D_{k_1} s_{k_1'}^{(n)}$. Pela Equação 2.2, uma vez que D_k é uma relação de dominância, temos que todas as extensões de $s_{k_1}^{(n)}$, e particularmente para $s_{n'}$, existe $s_{n_1} \in Ext(s_{k_1})$ tal que $s_{n_1} \underline{\Delta} s_{n'}$. Se $s_{n_1} \in C_n$, então a propriedade de cobertura é válida. Caso contrário, existe uma iteração $k_2 > k_1$, correspondente a iteração mais alta em que $C_0^{k_2}$ ainda contém restrições de s_{n_1}, que serão removidas aplicando-se uma das relações $D_{k_2}^i (i = 1, \ldots, p)$. Considere uma destas restrições, denotada por $s_{k_2}^{(n_1)}$. Como anteriormente, estabelecemos a existência de um estado $s_{k_2} \in C_{k_2}$ tal que existe $s_{n_2} \in Ext(s_{k_2})$ tal que $s_{n_2} \underline{\Delta} s_{n_1}$. A transitividade de $\underline{\Delta}$ garante que $s_{n_2} \underline{\Delta} s_{n'}$. Pela repetição deste processo, estabelecemos a existência de um estado $s_{k_2} \in C_{k_2}$ tal que existe $s_{n_2} \in Ext(s_{k_2})$ tal que $s_n \underline{\Delta} s_{n'}$.

Além disso, selecionando-se um conjunto C_n^i que é conjunto independente com relação a $D_n^i = \underline{\Delta}$, esta propriedade se mantém válida para C_n^p o qual é subconjunto de C_n^i. Dessa forma C_n, que corresponde ao conjunto eficiente reduzido, representa o conjunto de vetores objetivos não dominados. □

O teorema anterior apenas querer que um dos $n.p$ conjuntos coberturas seja independente com respeito a sua relação de dominância. Mesmo se todos os conjunto C_k^i....

> ...concluir parágrafo.

2.1.2 Geração de conjuntos cobertura e independente

Apresentamos no Algoritmo 3 uma maneira de produzir C_k^i um conjunto cobertura e independente de C_k^{i-1} com respeito a relação transitiva D_k^i (linha 6 do Algoritmo 2).

Proposição 2. *Para qualquer relação de dominância* D_k^i *sobre* S_k*, o Algoritmo 3 retorna* C_k^i *um conjunto cobertura e independente de* C_k^{i-1} *com respeito a* $D_k^i (j = 1, \ldots, n; i = 1, \ldots, p)$.

Demonstração. Claramente C_k^i é independente com relação a D_k^i, uma vez que inserimos o estado s_k ao conjunto C_k^i na linha ???? apenas se não for dominado por nenhum outro estado em C_k^i (linha ???) e todos os estados dominados por s_k tenham sido removidos de C_k^i (linha ??? e ???).

Algoritmo 3: Computação do conjunto C_k^i cobertura e independente do conjunto C_k^{i-1} com respeito a relação transitiva D_k^i.

input: $C_k^{i-1} = \{s_{k(1)}, \ldots, s_{k(r)}\}$

begin
  $C_k^i \leftarrow \{s_{k(1)}\}$;
  for $h \leftarrow 2, r$ **do**
    $C_k^i = \{s_{k'(1)}, \ldots, s_{k'(l_h)}\}$;
    $dominated \leftarrow false$;
    $dominates \leftarrow false$;
    $j \leftarrow 1$;
    while $j \leqslant l_h$ ***and*** $\neg dominated$ ***and*** $\neg dominates$ **do**
      if $s_{k'(j)} D_k^i s_{k(h)}$ **then**
        $dominated \leftarrow true$;
      else if $s_{k(h)} D_k^i s_{k'(j)}$ **then**
        $C_k^i \leftarrow C_k^i \backslash \{s_{k'(j)}\}$;
        $dominates \leftarrow true$;
      $j \leftarrow j + 1$;
    if $\neg dominated$ **then**
      while $j \leqslant l_h$ **do**
        if $s_{k(h)} D_k^i s_{k'(j)}$ **then**
          $C_k^i \leftarrow C_k^i \backslash \{s_{k'(j)}\}$;
        $j \leftarrow j + 1$;
      $C_k^i \leftarrow C_k^i \cup \{s_{k(h)}\}$;
  return C_k

Mostramos agora que C_k^i é um conjunto covertura de C_k^{i-1} com respeito a D_k^i. Considere $s_{k'} \in C_k^{i-1} \backslash C_k^i$. Isto ocorre tanto porque não passa no teste da linha ??? quanto porque foi removido na linha ??? ou ???. Isto é devido respectivamente a um estado s_{-k} já existente em C_k^i ou a ser incluído em C_k^i (na linha ???) tal que $s_{-k} D_k^i s_{k'}$. Pode ser que s_{-k} seja removido de C_k^i em uma iteração posterior da linha ??? se existir um novo estado $s^{\hat{k}} \in C_k^{i-1}$ a ser incluído em C_k^i, tal que $s_{\hat{k}} D_k^i s_{-k}$. Entretanto, a transitividade de D_k^i garante a existência, ao fim da iteração k, de um estado $s_k \in C_k^i$ tal que $s_k D_k^i s_{k'}$. □

O Algoritmo 3 pode ser aprimorado uma vez que geralmente é possível gerar estados de $C_k^{i-1} = \{s_{k(1)}, \ldots, s_{k(r)}\}$ conforme uma *ordem de preservação de dominância* para D_k^i de forma que todo $l < j (1 \leqslant l, j \leqslant r)$ temos tanto $s_{k(l)} D_k^i s_{k(j)}$ ou $\neg(s_{k(j)} D_k^i s_{k(l)})$. A proposição seguinte provê a condição necessária e suficiente para estabelecer a existência da ordem de preservação de dominância para uma relação de dominância.

Proposição 3. *Seja D_k uma relação de dominância sobre S_k. Existe uma ordem de preservação de dominância para D_k se, e somente se, D_k não admite ciclos em sua parte assimétrica.*

Demonstração.
$\Rightarrow$ A existência de um ciclo na parte assimétrica de D^k implicaria na existência de dois estados consecutivos $s_{k(j)}$ e $s_{k(l)}$ neste ciclo sendo $j > l$, o que é uma contradição.
$\Leftarrow$ Qualquer ordem topológica baseada na parte assimétrica de D_k é uma ordem de preservação de dominância para D_k. □

Na Seção seguinte é apresentado um exemplo de ordem de preservação de dominância. Se os estados em C_k^{i-1} são gerados conforme uma ordem de preservação de dominância para D_k^i, a linha ??? e o laço ???-??? do Algoritmo 3 podem ser omitidos.

2.2 Detalhes de implementação

Primeiramente é apresentada a ordem na qual serão considerados os itens no processo sequencial do algoritmo. Posteriormente são apresentados a três relações de dominância utilizadas no algoritmo ??? e a maneira com que serão utilizadas.

2.2.1 Ordem dos itens

A ordem na qual os itens são considerados é uma questão crucial na implementação do algoritmo. Sabe-se que no caso do problema da mochila unidimensional, para se obter uma boa solução, os itens devem ser geralmente considerados em ordem decrescente segundo a proporção p_i/w_i (EHRGOTT, 2013; MARTELLO; TOTH, 1990). Porém, para o caso multi-objetivo não existe uma ordem natural.

São introduzidas duas ordenações $\mathcal{O}^{sum}$ e $\mathcal{O}^{max}$ derivadas da agregação das ordens $\mathcal{O}^j$ inferidas pelas proporções p_i^j/w_i para cada objetivo ($j = 1, \ldots, m$). Considere r_i^l o rank (ou posição) do item l na ordenação $\mathcal{O}^j$. $\mathcal{O}^{sum}$ denota uma ordenação segundo valores crescentes da soma dos ranks dos itens nas m ordenações ($i = 1, \ldots, m$). $\mathcal{O}^{max}$ denota uma ordenação segundo valores crescentes de máximo ou piores ranks de itens nas m ordenações $\mathcal{O}^j(j = 1, \ldots, m)$, onde o pior rank do item l nas m ordenações $\mathcal{O}^j(j = 1, \ldots, m)$ é calculado como $max_{i=1}^m\{r_l^i\} + \frac{1}{mn}\sum_{i=1}^m r_l^i$ para distinguir itens com o mesmo rank máximo.

> Dizer que o artigo original conclui atraves de testes qual é a melhor ordenação para ser utilizada nas iterações do algoritmo.

2.2.2 Relações de dominância

Cada relação de dominância exploca uma consideração específica. Por isto recomenda-se utilizar relações de dominância que sejam complementares. Além disso, para se escolher uma relação de dominância é necessário considerar sua potencial capacidade de descarte de estados diante do custo computacional requerido.

A seguir serão apresentas as três relações de dominância utilizadas no algoritmo. As primeiras duas relações são reazoavelmente triviais de se estabelecer sendo a última ainda considerada, mesmo sendo um tanto mais complexa, devido à sua complementaridade diantes das duas primeiras.

A primeira relação de dominância se estabelece segundo a seguinte observação. Quando a capacidade residual de uma solução associada a um estado s_k da iteração k é maior ou igual a soma dos pesos dos itens restantes, i.e. itens $k+1, \ldots, n$, o único complemento de s_k que pode resultar em uma solução eficiente é o complemento máximo $I = \{k+1, \ldots, n\}$. Portanto, neste contexto, não se faz necessário gerar as extensões de s_k que não contenham todos os itens restants. Define-se então a relação de dominância D_k^r sobre S^k para $k = 1, \ldots, n$ como:

$$\forall s_k, s_{k'} \in S_k, \;\; s_k D_k^r s_{k'} \Leftrightarrow \begin{cases} s_{k'} \in S_{k-1}, \\ s_k = (s_{k'}^1 + p_k^1, \ldots, s_{k'}^m + p_k^m, s_{k'}^{m+1} + w_k), \text{ e} \\ s_{k'}^{m+1} \leqslant W - \sum_{i=k}^n w_i \end{cases}$$

A seguinte proposição mostra que D_k^r é de fato uma relação de dominância e apresenta propriedades adicionais de D_k^r.

Proposição 4 (Relação D_k^r)**.**

(a) D_k^r *é uma relação de dominância*

(b) D_k^r *é transitiva*

(c) D_k^r *admite ordem de preservação de dominância*

Demonstração.

(a) Considere dois estados s_k e $s_{k'}$ tal que $s_k D_k^r s_{k'}$. Isto implica que $s_k \underline{\Delta} s_{k'}$. Além disso, uma vez que $s_k^{m+1} = s_{k'^{m+1}} + w_k \leqslant W - \sum_{i=k+1}^n w_i$, qualquer subconjunto $I \subseteq \{k+1, \ldots, n\}$ é um complemento de $s_{k'}$ e s_k. Portanto, para todo estado $s_{n'} \in Ext(s_{k'})$, existe $s_n \in Ext(s_k)$, baseado no mesmo complemento que $s_{n'}$, tal que $s_n \underline{\Delta} s_{n'}$. Isto estabelece que D_k^r satisfaz a Equação 2.2 da Definição 3.

(b) Trivial.

(c) Pela Proposição 3, uma vez que D_k^r é transitiva. □

Esta relação de dominância é um tanto fraca, uma vez que em cada k-ésima iteração ela pode apenas..... Apesar disso, é uma relação de dominância extremamente fácil de ser verificada, uma vez que, ao ser o valor de $W - \sum_{i=k}^n$ computado, a relação D_k^r requer apenas uma comparação para ser estabelecida entre dois estados.

A relação de dominância seguinte é a generalização para o caso multi-objetivo da relação de dominância proposta por Weingartner e Ness (WEINGARTNER; NESS, 1967)

e utilizada no clássico algoritmo de Nemhauser e Ulmann (NEMHAUSER; ULLMANN, 1969). Esta segunda relação de dominância é definida sobre s_k para $k = 1, \ldots, n$ por:

$$\forall s_k, s_{k'} \in S_k,\ s_k D_k^{\underline{\Delta}} s_{k'} \Leftrightarrow \begin{cases} s_k \underline{\Delta} s_{k'} & \text{e} \\ s_k^{m+1} \leqslant s_{k'}^{m+1} & \text{se } k < n \end{cases}$$

Observe que a condição sobre os pesos s_k^{m+1} e $s_{k'}^{m+1}$ garante que todo complemento de $s_{k'}$ é também um complemento de s_k. A seguinte proposição mostra que $D_k^{\underline{\Delta}}$ é de fato uma relação de dominância e apresenta propriedades adicionais de $D_k^{\underline{\Delta}}$.

Proposição 5 (Relação $D_k^{\underline{\Delta}}$)**.**

(a) $D_k^{\underline{\Delta}}$ *é uma relação de dominância*

(b) $D_k^{\underline{\Delta}}$ *é transitiva*

(c) $D_k^{\underline{\Delta}}$ *admite ordem de preservação de dominância*

(c) $D_k^{\underline{\Delta}} = \underline{\Delta}$

Demonstração.

(a) Considere estados s_k e $s_{k'}$ tal que $s_k D_k^{\underline{\Delta}} s_{k'}$. Isto implica que $s_k \underline{\Delta} s_{k'}$ Além disso, uma vez que $s_k^{m+1} \leqslant s_{k'}^{m+1}$, qualquer subconjunto $J \cup \{k+1, \ldots, n\}$ complemento de $s_{k'}$ é também complemento de s_k. Portanto, para todo $e_{n'} \in Ext(s_{n'})$, existe $e_n \in Ext(s_n)$, baseado no mesm complemento de $s_{n'}$. O que estabelece que $D_k^{\underline{\Delta}}$ satisfaz a Equação 2.2 da Definição 3.

(b) Trivial.

(c) Pela Proposição 3, uma vez que $D_k^{\underline{\Delta}}$ é transitiva.

(d) Por definição. □

A relação $D_k^{\underline{\Delta}}$ é uma relação de dominância poderosa, uma vez que um estado pode possivelmente dominar todos os outros estados de maior peso. Esta relação requer no máximo $m + 1$ comparações para ser estabelecida entre dois estados.

A terceira relação de dominância é baseada na comparação entre extensões específicas de um estado e um limite superior de extensões de outros estados. Um limite superior de um estado é definido segundo no contexto a seguir.

Definição 4 (Limite superior)**.** *Um vetor objetivo* $u = (u^1, \ldots, u^m)$ *é um* limite superior *de um estado* $s_k \in S_k$ *se, e somente se,* $\forall s_n \in Ext(s_k)$ *tem-se que* $u^i \geqslant s_n^i, i = 1, \ldots, m$.

Um tipo geral de relação de dominância pode ser derivada da seguinte maneira: considere dois estados $s_k, s_{k'} \in S_k$, se existe um complemento I de s_k e um limite superior $\tilde{u}$ de $s_{k'}$ tal que $s_k^j + \sum_{i \in I} p_i^j \geqslant \tilde{u^j},\ j = 1, \ldots, m$, então s_k domina $s_{k'}$.

Este tipo de relação de dominância pode ser implementara apenas para complementos específicos e limites superiores. Na implementação do algoritmo em questão são apenas considerados dois complementos específicos I' e I'' obtidos pelo simples algoritmo de preenchimento *guloso* definido a seguir. Após rerotular os itens $k+1, \ldots, m$ de acordo com a ordenação $\mathcal{O}^{sum}$ (e $\mathcal{O}^{max}$ respectivamente), o complemento I' (e I'' respectivamente) são obtidos através a inserção sequencial dos itens restantes na respectiva solução de forma que a restrição de capacidade é respeitada.

Para computar u, utiliza-se o limite inferior proposto em (MARTELLO; TOTH, 1990) para cada valor de objetivo. Considere $\overline{W}(s_k) = W - s_k^{m+1}$ a capacidade residual associada ao estado $s_k \in S_k$. Denota-se por $c_j = min\Big\{l_j \in \{k_1, \ldots, n\} \mid \sum_{i=k+1}^{l_j} w_i > \overline{W}(s_k)\Big\}$ a posição do primeiro item que não pode ser adicionado ao estado $s_k \in S_k$ quando os itens $k+1, \ldots, n$ são rerotulados de acordo com a ordenação $\mathcal{O}^j$. Desde modo, segundo (MARTELLO; TOTH, 1990), quando os itens $k+1, \ldots, n$ são rerotulados de acordo com a ordenação $\mathcal{O}^j$, um limite inferior para o j-ésimo valor objetivo de $s_k \in S_k$ é para $j = 1, \ldots, m$:

$$u^j = s_k^j + \sum_{i=k+1}^{c_j - 1} p_i^j + max\left\{\left\lfloor \overline{W}(s_k)\frac{p_{c_j+1}^j}{w_{c_j+1}} \right\rfloor, \left\lfloor p_{c_j}^j - \Big(w_{c_j} - \overline{W}(s_k)\Big).\frac{p_{c_j-1}^j}{w_{c_j-1}} \right\rfloor\right\} \tag{2.3}$$

Finalmente definimos D_k^b uma relação de dominância como caso particular do tipo geral para $k = 1, \ldots, n$ por:

$$\forall s_k, s_{k'} \in S_k,\ s_k D_k^b s_{k'} \Leftrightarrow \begin{cases} s_k^j + \sum_{i \in I'} p_i^j \geqslant \tilde{u^i}, & j = 1, \ldots, m \\ \text{ou} & \\ s_k^j + \sum_{i \in I''} p_i^j \geqslant \tilde{u^i}, & j = 1, \ldots, m \end{cases}$$

onde $\tilde{u} = (\tilde{u}_1, \ldots, \tilde{u}_m)$ é o limite superior para $s^{k'}$ computado de acordo com a Equação 2.3.

A seguinte proposição mostra que D_k^b é de fato uma relação de dominância e apresenta propriedades adicionais de D_k^b.

Proposição 6 (Relação D_k^b)**.**

(a) D_k^b *é uma relação de dominância*

(b) D_k^b *é transitiva*

(c) D_k^b *admite ordem de preservação de dominância*

(c) $D_n^b = \underline{\Delta}$

Demonstração.

(a) Considere os estados s_k e $s_{k'}$ tal que $s_k D_k^b s_{k'}$. Isto implica que existe $I \in I', I''$ que capaz de gerar um estado $s_n \in Ext(s_{k'})$. Além disso, uma vez que $\tilde{u}$ é um limite superior de $s_{k'}$, temos $\tilde{u} \underline{\Delta} s_{n'}, \forall s_{n'} \in Ext(s_{n'})$. Portanto, por transitividade de $\underline{\Delta}$, temos que $s_n \underline{\Delta} s_{n'}$, o que estabelece que D_k^b satisfaz a condição da Equação 2.2 da Definição 3.

(b) Considere os estados $s_k, s_{k'}$ e s_{-k} tal que $s_k D_k^b s_{k'}$ e $s_{k'} D_k^b s_{-k}$. Isto implica que, por um lado, existe $I_1 \in \{I', I''\}$ tal que $s_k^j + \sum_{i \in I_1} p_i^j \geqslant \tilde{u}_j (j = 1, \ldots, m)$ por outro lado, existe $I_2 \in \{I', I''\}$ tal que $s_{k'}^j + \sum_{i \in I_2} p_i^j \geqslant \bar{u}_j (j = 1, \ldots, m)$ Uma vez que $\tilde{u}$ é um limite superior para $s_{\tilde{k}}$ temos que $\tilde{u}_j \geqslant s_{k'}^i + \sum_{j \in J_2} p_i^j (i = 1, \ldots, m)$. Portanto temos que $s_k D_k^b s_{-k}$.

(c) Pela Proposição 3, uma vez que D_k^b é transitiva.

(d) Por definição. □

A relação D_k^b é mais difícil de ser verificada do que as relações D_k^r e $D_k^{\underline{\Delta}}$ uma vez que requer uma maior quantidade de comparações e informações sobre outros estados.

Obviamente, a relação D_k^b seria mais forte se utizados complementos adicionais (de acordo com outras ordenações) para s_k e computado, ao invés de apenas um limite superior u, um conjunto de limites superiores, como por exemplo, a proposta apresentada em (EHRGOTT; GANDIBLEUX, 2007). Porém, no contexto em questão, uma vez que temos que verificar D_k^b para muitos estados, enriquecer D_k^b dessa forma demandaria um esforço computacional alto demais.

2.2.3 Detalhes de implementação

Para se ter uma implementação eficiente, são utilizadas as três relações de dominância apresentadas na Seção 2.2.2 em cada iteração. Como dito na Seção anterior uma relação de dominância demanda maior esforço computacional do que outra para ser vericada. Além disso, mesmo que sejam parcialmente complementares, é frequente mais de uma relação de dominância ser válida para o mesmo par de estados. Por esse motivo é natural aplicar primeiramente relações de dominância que podem ser verificadas facilmente (como D_k^r e $D_k^{\underline{\Delta}}$) para então verificar num conjunto reduzido de estados uma relações mais custosas (como D_k^b).

A seguir será descrito os detalhes da implementação e amplicação das três relações de dominância. O Algoritmo 4, que computa, na k-ésima iteração, o subconjunto de estados candidatos C_k a partir de um subconjunto $C_{k-1} (k = 1, \ldots, n)$, substitui as linhas ??? a ??? do Algoritmo 3.

A utilização das dominâncias D_k^r e $D_k^{\underline{\Delta}}$ é descrita nas linhas ???-??? e a utilização da dominância D_k^b é descrita nas linhas ???-???. O algoritmo utiliza dois sub-

procedimentos: `MantainNonDominated`, que remove estados D_k^{Δ}-dominados, e procedimento `KeepNonDominated`, que é utilizado durante a aplicação da relação D_k^b.

Algoritmo 4: O algoritmo de Nemhauser e Ullmann para o MOKP.

input: $C_{k-1} = \{s_{k-1(1)}, \ldots, s_{k-1(r)}\}$ tal que $s_{k-1(i)} \geqslant_{lex} s_{k-1(j)}\ \forall i < j (1 \leqslant i, j \leqslant r)$

begin
  $C_k \leftarrow \emptyset; M_k \leftarrow \emptyset; i \leftarrow 1; j \leftarrow 1;$
  while $j \leqslant r$ ***and*** $s_{k-1(1)}^{m+1} + \sum_{l=k}^{n} w_l \leqslant W$ **do**
    $j \leftarrow j + 1;$
  while $i \leqslant r$ ***and*** $s_{k-1(1)}^{m+1} + w_k \leqslant W$ **do**
    $s_k \leftarrow (s_{k-1(1)}^1 + p_k^1, \ldots, s_{k-1(1)}^m + p_k^m, s_{k-1(1)}^{m+1} + w_k);$
    while $j \leqslant r$ ***and*** $s_{k-1(j)} \leqslant_{lex} s_k$ **do**
      `MantainNonDominated`$(s_{k-1(j)}, M_k, C_k);\ j \leftarrow j + 1;$
    `MantainNonDominated`$(s_k, M_k, C_k);\ i \leftarrow i + 1;$
  while $j \leqslant r$ **do**
    `MantainNonDominated`$(s_{k-1(j)}, M_k, C_k);\ j \leftarrow j + 1;$
  if $k = n$ **then**
    $C_n \leftarrow M_n;$
  else
    $F \leftarrow \emptyset;$
    for $\mathcal{O} \in \{\mathcal{O}^{sum}, \mathcal{O}^{max}\}$ **do**
      for $s_k \in M_k$ **do**
        Rerotular itens $k+1, \ldots, n$ segundo ordenação $\mathcal{O}$;
        $s_n \leftarrow s_k;$
        for $j \leftarrow k+1, \ldots, n$ **do**
          if $s_n^{m+1} + w_j \leqslant W$ **then**
            $s_n \leftarrow (s_n^1 + p_j^1, \ldots, s_n^m + p_j^m, s_n^{m+1} + w_j);$
        $F \leftarrow$ `KeepNonDominated`$(s_n, F);$
    $i \leftarrow 1; remove \leftarrow true; F \leftarrow \{s_{n(1)}, \ldots, s_{n(h)}\};$
    while $i \leqslant c$ ***and*** $remove$ **do**
      Computar limite superior u para $s_{k(i)}$ conforme Equação 2.3;
      $j \leftarrow 1; remove \leftarrow false;$
      while $j \leqslant h$ ***and*** $s_{n(j)} \underline{\Delta}_{lex} u$ ***and*** $\neg remove$ **do**
        if $s_{n(j)} \underline{\Delta} u$ **then**
          $remove \leftarrow true;$
        else
          $j \leftarrow j + 1;$
      if $remove$ **then**
        $C_k \leftarrow C_k \backslash \{s_{k(i)}\}; i \leftarrow i + 1;$
  return $C_k;$

2.3 Métodos Heurísticos

2.4 Evolução estocástica por complexos

O Algoritmo de Evolução estocástica por complexos (EEC), *Shuffled Complex Evolution* em inglês, é um algoritmo evolutivo populacional proposto por Duan (DUAN; SOROOSHIAN; GUPTA, 1992) e tem sido utilizado com sucesso em problemas de escalonamento (ZHAO et al., 2015), seleção de projetos (ELBELTAGI; HEGAZY; GRIERSON, 2007), problema da mochila $0-1$ (BHATTACHARJEE; SARMAH, 2014) e problema da mochila multi-dimensional (BARONI; VAREJÃO, 2015; BARONI; VAREJÃO, 2016).

O EEC é inspirado na evolução natural que ocorre de forma simultânea em comunidades independentes. O algoritmo trabalha com uma população particionada em N comunidades, ou complexos, cada uma contendo M indivíduos. Inicialmente a população de $N * M$ indivíduos é tomada aleatoriamente do espaço de soluções viáveis. Após esta inicialização a população é ordenada em ordem descrecente de aptidão e o melhor global é identificado. Toda a população é então particionada em N complexos, cada um contendo M indivíduos. Neste processo de distribuição aleatória o primeiro indivíduo é incluído no primeiro complexo, o segundo indivíduo no segundo complexo, o M-ésimo indivíduo no M-ésimo complexo, o $M+1$-ésimo indivíduo no primeiro complexo, e assim por diante.

O próximo passo após a distribuição dos indivíuos em complexos é evoluir cada complexo uma quantidade constante K'. Neste processo, considerando que os indivíduos em cada complexo estão ordenados em ordem decrescente de aptidão, um subcomplexo é de P indivíduos é selecionado dentre o complexo utilizando uma distribuição triangular, em que o i-ésimo indivíduo tem a probabilidade $p_i = \frac{2(n+1-i)}{n(n+1)}$ de ser selecionado. A utilização da distribuição triangular tem por objetivo priorizar os indivíduos com melhor aptidão, favorecendo assim a convergência do algoritmo.

Após a seleção do subcomplexo, o seu pior indivíduo é identificado para ser subtituido por um novo indivíduo. Este novo indivíduo é gerado através do cruzamento do pior indivíduo e um outro indivíduo com melhor aptidão. Primeiramente o melhor indivíduo do subcomplexo é considerado. Caso o indivíduo gerado não seja melhor que o pior indivíduo selecionado, o melhor indivíduo do complexo é então usado no cruzamento. Se este último cruzamento não resultou em melhoria, o melhor indivíduo de toda a população é considerado. Finalmente, se todos os cruzamentos não foram capaz de gerar um melhor indivíduo, esta pior solução selecionada é substituída por um novo indivíduo retirado de forma aleatória do espaço de soluções viáveis. Este último procedimento é importante para impedir que o algoritmo fique *preso* num ótimo local. A Figura 3 apresenta o procedimento de evolução descrito acima em um fluxograma.

Após evoluir todos os N complexos toda a população é novamente ordenada em ordem descrecente de aptidão e o processo continua até que a condição de parada seja satisfeita.

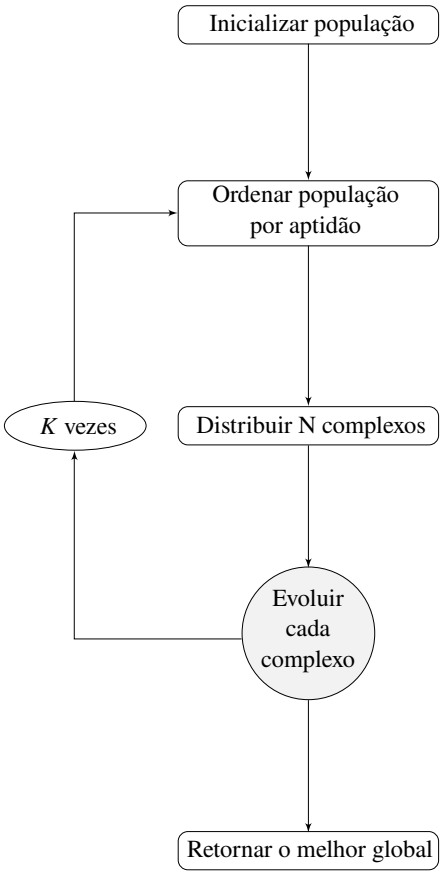


Figura 2: The shuffled complex evolution algorithm.

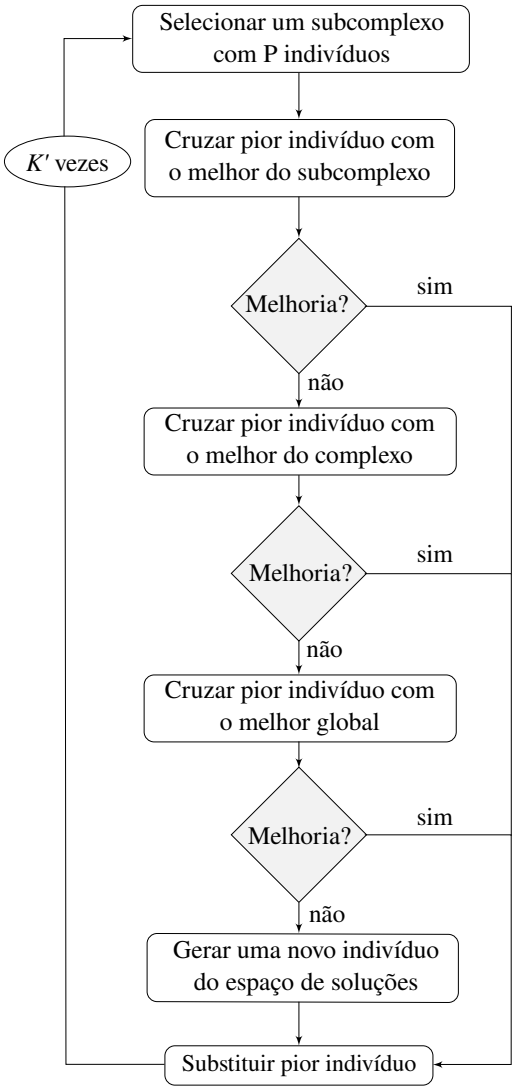


Figura 3: The evolving stage of SCE for a single complex.

A Figura 2 apresenta os passos do algoritmo ECC num fluxograma em que a condição de parada é a execução de uma quantidade fixa de K evoluções.

2.5 O EEC para o MOKP

Como pode-se notar em sua descrição, o EEC é facilmente aplicável a qualquer problema de otimização, inclusive o MOKP, desde que seja possível (a) definir um procedimento de criação de uma nova solução alteatória viável e (b) definir um procedimento de cruzamento entre duas soluções. Esses dois procedimentos para o MOKP são descritos no Algoritmo 5 e Algoritmo 6.

Para se contruir uma nova solução aleatória para o MOKP (Algoritmo 5) os índices dos n itens são primeiramente permutados em ordem aleatória e armazenados em uma lista v (linha 2). Uma nova solução vazia é definida (linha 3). O algoritmo tenta de

Algoritmo 5: Contrução de solução aleatória para o MOKP.

```
1 begin
2     v ← shuffle(1, 2, ..., n);
3     s ← 0⃗;                                          ▷ solução vazia
4     for i ← 1 : n do
5         s[v_i] ← 1;                                  ▷ inserção de item
6         if w(s) > W then
7             s[v_i] ← 0;                              ▷ verificação de viabilidade
8     retorna s;
```

forma iterativa preencher a solução com um item retirado da lista de índices (linhas 4-9). A viabilidade da solução é então verificada: se o item inserido tornar a solução inviável (linha 6) ele é então retirado da solução (linha 7). Após testar a inserção de todos os itens a solução contruida é retornada.

Algoritmo 6: Procedimento de cruzamento entre duas soluções do MOKP.

```
   input: s : pior indivíduo, b : melhor indivíduo, c
1  begin
2      v ← shuffle(1, 2, ..., n);
3      for i ← 1 : c do
4          s[v_i] ← b[v_i];
5      if w(s) > W then
6          reparar s;
7      Computar aptidão de s;
8      return s;
```

O procedimento de cruzamento (Figura 6) recebe como entrada s o pior indivíduo vindo do subcomplexo selecionado, b um indivíduo com maior aptidão que s e parâmetro c sendo o número de genes a serem carregados de b. O parâmetro c controla o quão similar o novo indivíduo será do melhor indivíduo dado como entrada. Primeiramente os índices dos n itens são permutados em uma ordem aleatória e armazenados numa lista (linha 2). Os c genes escolhidos aleatoriamente é então carregado do melhor indivíduo para o pior indivíduo (linhas 3-5). Em seguida a viabilidade da solução é verificada (linha 6) e então reparada se necessário (linha 7). Finalmente a aptidão da solução gerada é atualizada (linha 9) e então retornada (linha 10).

Explicar adaptação da heurística para caso multi-objetivo (uso de archive).

3 A k-d tree

3.1 Verificação de dominância e a busca de faixa

Como dito anteriormente, solucionar um problema multi-objetivo significa apresentar o seu conjunto Pareto, ou seja, o conjunto de soluções não dominadas. Geralmente o processo de solução compreende-se em contruir este conjunto de soluções de forma incremental. Por este motivo, um dos procedimentos mais executados durante o processo é verificar se uma determinada solução é dominada por alguma outra solução existente num conjunto candidato. Um algoritmo pode até mesmo necessitar de extrair um cojunto cobertura, ou seja, extrair todas as soluções não-dominadas de um grande conjuto de soluções.

Esta operação deve demandar exforço quadrático sobre o número total de soluções se implementada como uma comparação par-a-par. Entretanto, se as soluções forem interpretadas como pontos em um espaço multi-dimensional, podemos deduzir da Equação 2.1 que esta operação corresponde em verificar a existência um ponto numa determinada região do espaço. O mesmo pode ser considerado no caso mais específico para a dominância da mochila, segundo a Equação ???. Formalmente:

$$x \underline{\Delta} y \iff pnt(y) \in R(x)$$

where

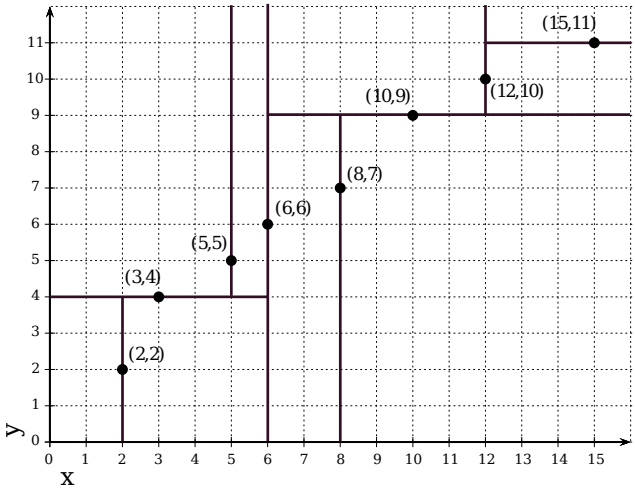
$$pnt(x) = \Big(f_1(x), \ldots, f_m(x), w(x)\Big)$$
$$R(x) = \Big\{a \in \mathbb{R}^{m+1} \;\Big|\; a_{m+1} \leqslant w(x) \text{ and } a_i \geqslant f_i(x),\ i \in \{1, \ldots, m\}\Big\}$$

> Colocar referência para a equação de dominância da mochila (texto acima).

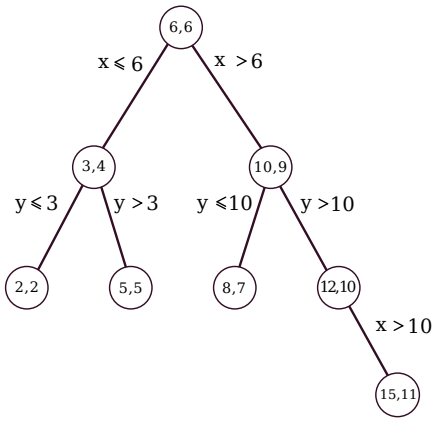
O problema de determinar a existência de um ponto numa determinada região do espaço é já bem conhecido da computação e chama-se problema da *busca de baixa* (ou *range search* em ingês) (AGARWAL; ERICKSON et al., 1999). O problema de busca de faixa é largamente aplicado, por exemplo, na área da computação gráfica e jogos, onde é necessário se verificar a colisão entre pontos e polígonos. Para se ter uma solução eficiente deste problema evitando o esforço computacional quadrático, os pontos devem ser indexados multidimensional.....

> Melhorar parágrafo acima e conferir texto inicial do parágrafo abaixo.

Devido a sua simplicidade e eficiência, a estrutura de dados mais utilizada para o problema é a *k-d tree* (PREPARATA; SHAMOS, 2012). Proposta por Jon Louis Bentley



(a) Pontos dispostos num plano bi-dimensional.



(b) Pontos indexados por uma 2-d tree.

Figura 4: Exemplo de pontos indexados por uma k-d tree.

em 1957 (BENTLEY, 1975), a k-d tree é um tipo de árvore binária de construção simples e baixa utilização de memória. Apesar de sua simplicidade, além da operação de buca de faixa, a k-d tree suporta outras operações como busca de vizinho mais próximo. Por este motivo é também bastante utilizada em algoritmos de clasterização (KANUNGO et al., 2002; INDYK; MOTWANI, 1998) e renderização gráfica (OWENS et al., 2007).

Como uma árvore binária comum, a cada nível recursivo da árvore a k-d tree subdivide os dados em duas partes. Porém, diferentemente de uma árvore de busca binária comum, as quais utilizam apenas uma *chave* em todos os níveis da árvore, a k-d tree utiliza um total de k chaves fazendo um revezamento circular entre as chaves à medida que caminha nos níveis da árvore.

A Figura 4 apresenta (a) um conjunto de pontos dispostos num espaço bi-dimensional e indexados por uma (b) 2-d tree. O primeiro e terceiro nível da 2-d tree indexa a componente x dos pontos, enquanto o segundo nível indexa a componente y. Cada ponto indexado pela árvore subdivide o espaço em dois de acordo com o valor da componente que está sendo indexada. A subdivisão do espaço é representada na figura por uma linha mais grossa.

A Figura 5 apresenta um exemplo de operação de verificação de dominância utilizando uma 2-d tree como estrutura de indexação. A área acinzentada não tem intersecção com a área dominada pelo ponto x (área hachurada), portanto as soluções dentro da área acinzentada não são avaliadas.

Propor passo-a-passo da operação.

Com relação à eficiência da k-d tree é importante considerar que não é recomendável escalar de forma arbitrária o número k de dimensões indexadas pela k-d tree, esperando assim escalar também sua eficiência. Mesmo que o dado possua todas estas dimensões.

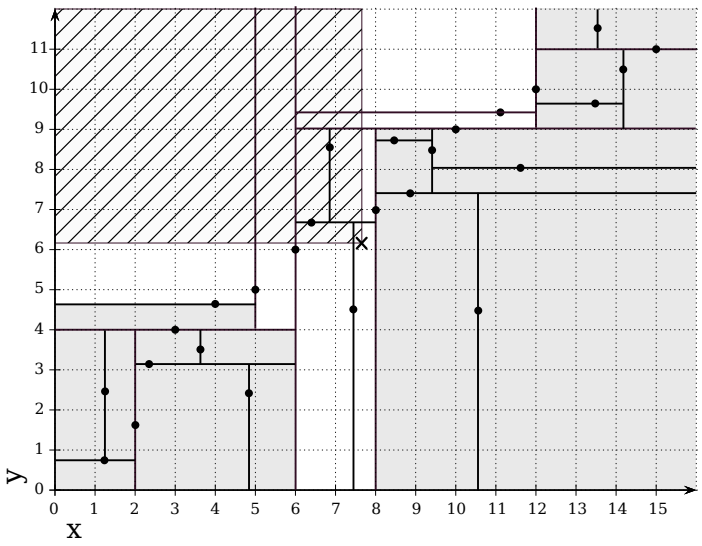


Figura 5: Exemplo de operação de verificação de dominância utilizando a k-d tree.

Como regra geral considera-se que um k-d tree é adequada para indexar um conjunto com n pontos se n não for muito maior que 2^k(TOTH; O'ROURKE; GOODMAN, 2004), caso contrário, a performance da k-d tree se assemelhará a de uma busca linear exaustiva.

Espera-se que a k-d tree auxilie as operações de verificação de dominância *podando* uma grande quantidade de soluções, demandando um menor número de comparações entre soluções, melhorando assim a performance dos algoritmos.

4 Experimentos computacionais

Vários experimentos computacionais foram realizados com o objetivo de veirficar a eficiência da indexação multi-dimensional, especialmente em instâncias com mais de duas dimensões.

Introduzir comentário sobre as instâncias da Bazgan.

Quatro tipos de instâncias bi-objetivo são consideradas:

A) Aleatórias: $p_i^j \in [1, 1000], w_i \in [1, 1000]$.

B) Não-conflitantes: $p_i^1 \in [111, 1000]$,
$p_i^2 \in [p_i^1 - 100, p_i^1 + 100]$,
$w_i \in [1, 1000]$.

C) Conflitantes: $p_i^1 \in [1, 1000]$,
$p_i^2 \in [max\{900 - p_i^1; 1\}, min\{1100 - p_i^1, 1000\}]$,
$w_i \in [1, 1000]$.

D) Conflitantes com pesos correlacionados: $p_i^1 \in [1, 1000]$,
$p_i^2 \in [max\{900 - p_i^1; 1\}, min\{1100 - p_i^1, 1000\}]$,
$w_i \in [p_i^1 + p_i^2 - 200, p_i^1 + p_i^2 + 200]$.

onde $\in [a, b]$ denota uma distribuição uniforme aleatória no intervalo $[b]$.

Para os experimentos com 3-objetivo considerou-se a generalização introduzida por (BAZGAN; HUGOT; VANDERPOOTEN, 2009) para os tipos A e C e também duas propostas de generalização para os tipo B e D:

A) Aleatórias: $p_i^j \in [1, 1000]$
$w_i \in [1, 1000]$

B) Não-conflitantes: $p_i^1 \in [111, 1000]$,
$p_i^2 \in [p_i^1 - 100, p_i^1 + 100]$,
$p_i^3 \in [p_i^1 - 100, p_i^1 + 100]$,
$w_i \in [1, 1000]$.

C) Conflitantes: $p_i^1 \in [1, 1000]$, $p_i^2 \in [1, 1001 - p_i^1]$
$p_i^3 \in [max\{900 - p_i^1 - p_i^2; 1\}, min\{1100 - p_i^1 - p_i^2, 1001 - p_i^1\}]$
$w_i \in [1, 1000]$.

D) Conflitantes com pesos correlacionados: $p_i^1 \in [1, 1000]$
$p_i^2 \in [1, 1001 - p_i^1]$
$p_i^3 \in [max\{900 - p_i^1 - p_i^2; 1\}, min\{1100 - p_i^1 - p_i^2, 1001 - p_i^1\}]$
$w_i \in [p_i^1 + p_i^2 + p_i^3 - 200, p_i^1 + p_i^2 + p_i^3 + 200]$.

Instâncias do tipo B são consideradas mais fáceis enquanto instâncias do tipo D são consideradas as mais difíceis. Para todas as instâncias, atribui-se $W = \frac{1}{2}\left\lfloor \sum_{k=1}^{n} w^k \right\rfloor$. Para cada tipo e valor de n dez instâncias foram geradas.

5 Conclusão

References

AGARWAL, P. K.; ERICKSON, J. et al. Geometric range searching and its relatives. *Contemporary Mathematics*, Providence, RI: American Mathematical Society, v. 223, p. 1–56, 1999.

BARONI, M. D. V.; VAREJÃO, F. M. A shuffled complex evolution algorithm for the multidimensional knapsack problem. In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. [S.l.]: Springer, 2015. p. 768–775.

BARONI, M. D. V.; VAREJÃO, F. M. A shuffled complex evolution algorithm for the multidimensional knapsack problem using core concept. In: IEEE. *Evolutionary Computation (CEC), 2016 IEEE Congress on*. [S.l.], 2016. p. 2718–2723.

BAZGAN, C.; HUGOT, H.; VANDERPOOTEN, D. Solving efficiently the 0–1 multi-objective knapsack problem. *Computers & Operations Research*, Elsevier, v. 36, n. 1, p. 260–279, 2009.

BENTLEY, J. L. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, ACM, v. 18, n. 9, p. 509–517, 1975.

BHATTACHARJEE, K. K.; SARMAH, S. P. Shuffled frog leaping algorithm and its application to 0/1 knapsack problem. *Applied Soft Computing*, Elsevier, v. 19, p. 252–263, 2014.

DUAN, Q.; SOROOSHIAN, S.; GUPTA, V. Effective and efficient global optimization for conceptual rainfall-runoff models. *Water resources research*, Wiley Online Library, v. 28, n. 4, p. 1015–1031, 1992.

EHRGOTT, M. *Multicriteria optimization*. [S.l.]: Springer Science & Business Media, 2013.

EHRGOTT, M.; GANDIBLEUX, X. Bound sets for biobjective combinatorial optimization problems. *Computers & Operations Research*, Elsevier, v. 34, n. 9, p. 2674–2694, 2007.

ELBELTAGI, E.; HEGAZY, T.; GRIERSON, D. A modified shuffled frog-leaping optimization algorithm: applications to project management. *Structure and Infrastructure Engineering*, Taylor & Francis, v. 3, n. 1, p. 53–60, 2007.

INDYK, P.; MOTWANI, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In: ACM. *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. [S.l.], 1998. p. 604–613.

ISHIBUCHI, H.; AKEDO, N.; NOJIMA, Y. Behavior of multiobjective evolutionary algorithms on many-objective knapsack problems. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 19, n. 2, p. 264–283, 2015.

KANUNGO, T. et al. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 24, n. 7, p. 881–892, 2002.

MARTELLO, S.; TOTH, P. *Knapsack Problems, J.* [S.l.]: Wiley & Sons, Chichester, 1990.

NEMHAUSER, G. L.; ULLMANN, Z. Discrete dynamic programming and capital allocation. *Management Science*, INFORMS, v. 15, n. 9, p. 494–505, 1969.

OWENS, J. D. et al. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, Blackwell Publishing Ltd, v. 26, n. 1, p. 80–113, 2007. ISSN 1467-8659. Disponível em: <http://dx.doi.org/10.1111/j.1467-8659.2007.01012.x>.

PREPARATA, F. P.; SHAMOS, M. *Computational geometry: an introduction.* [S.l.]: Springer Science & Business Media, 2012.

ROSENBLATT, M. J.; SINUANY-STERN, Z. Generating the discrete efficient frontier to the capital budgeting problem. *Operations Research*, INFORMS, v. 37, n. 3, p. 384–394, 1989.

TENG, J.-Y.; TZENG, G.-H. A multiobjective programming approach for selecting non-independent transportation investment alternatives. *Transportation Research Part B: Methodological*, Elsevier, v. 30, n. 4, p. 291–307, 1996.

TOTH, C. D.; O'ROURKE, J.; GOODMAN, J. E. *Handbook of discrete and computational geometry.* [S.l.]: CRC press, 2004.

WEINGARTNER, H. M.; NESS, D. N. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, INFORMS, v. 15, n. 1, p. 83–103, 1967.

ZHAO, F. et al. A shuffled complex evolution algorithm with opposition-based learning for a permutation flow shop scheduling problem. *International Journal of Computer Integrated Manufacturing*, Taylor & Francis, v. 28, n. 11, p. 1220–1235, 2015.