Um algoritmo exato para uma classe de problemas de programação linear-fracionária

Rian Gabriel S. Pinheiro, Ivan César Martins, Fábio Protti,
Luiz Satoru Ochi, Luidi Gelabert Simonetti
Instituto de Computação, Universidade Federal Fluminense
Rua Passo da Patria, 156 - Bloco E - 3^{o}andar, São Domingos, Niterói - RJ
{rgpinheiro,imartins,fabio,satoru,luidi}@ic.uff.br

RESUMO

Neste artigo é proposto um algoritmo exato para a solução de uma classe de Problemas de Programação Linear-fracionária (PPLF). Esta classe consiste nos problemas que possuem na função objetivo uma fração na qual seu numerador e denominador possuem como contradomínio o conjunto dos números inteiros. O algoritmo proposto se baseia em limites superiores e inferiores que são atualizados iterativamente através de chamadas de subproblemas parametrizados. Como aplicação validamos o algoritmo em dois PPLFs conhecidos.
PALAVRAS CHAVE. Programação Matemática. Otimização Combinatória. Células de Manufatura.
Área principal OC - Otimização Combinatória.

ABSTRACT

In this paper we propose an exact algorithm for solving a class of Linear-Fractional Programming Problems (LFPP). This class consists of problems whose objective function is a fraction in which both the numerator and denominator are integers. The proposed algorithm is based on upper and lower bounds which are updated iteratively through parameterized subproblems. As an application, we validated the algorithm in two well-known LFPPs.
KEYWORDS. Mathematical Programming. Combinatorial Optimization. Cellular Manufacturing.
Main area CO - Combinatorial Optimization.

1 Introdução

Uma instância $\mathcal{P}$ de um Problema de Programação Linear-Fracionária (PPLF) consiste na especificação de um conjunto $\mathcal{S} \subset \mathbb{Z}$ também chamado de *domínio* ou *estrutura* do problema. Além disso, têm-se também duas funções $g : \mathcal{S} \to \mathbb{R}$ e $h : \mathcal{S} \to \mathbb{R}$. Define-se $\mathcal{P}$ como

$$\mathcal{P} : \max f(x) = \frac{g(x)}{h(x)}, \qquad \text{para } x \in \mathcal{S}.$$

A literatura assume que $g(x) > 0$ para algum x e que $h(x) > 0$ para todo $x \in \mathcal{S}$. Este caso é chamado de caso geral [6]. Para resolvê-lo, aplicam-se principalmente técnicas de busca baseadas em gradientes, apesar de haver técnicas que transformam o problema em um Problema de Programação Linear (PPL). Os principais métodos utilizados na literatura são:

- método de Newton [6];
- busca de Megiddo [6];
- transformação de Charnes-Cooper [1].

Os dois primeiros são métodos de gradientes, já o último é uma linearização do PPLF, sendo sua principal desvantagem o fato do número de variáveis aumentar consideravelmente.

Neste artigo é proposto um algoritmo para o caso em que as funções g e h possuem o conjunto dos inteiros como seus contradomínios, ou seja, $g : \mathcal{S} \to \mathbb{Z}$ e $h : \mathcal{S} \to \mathbb{Z}_+$. Este algoritmo consiste em uma generalização do algoritmo proposto por Pinheiro [5].

2 Proposta

O algoritmo proposto consiste em particionar o espaço de busca e iterativamente executar um algoritmo exato em cada uma das partes, atualizando o limite inferior e superior do problema principal até que eles se encontrem, provando assim sua otimalidade.

Considere o PPLF denotado $\mathcal{P}$. Podemos criar o seguinte PPL $\mathcal{P}'$.

$$\mathcal{P}' : \min h(x) - g(x), \qquad \text{para } x \in \mathcal{S}.$$

Veja que a solução ótima de $\mathcal{P}'$ não é necessariamente ótima para $\mathcal{P}$.

Sejam k^* o valor da solução ótima de $\mathcal{P}'$ e x^* uma solução ótima de $\mathcal{P}'$; dessa forma $k^* = h(x^*) - g(x^*)$. Para um $k \in \mathbb{Z}$ fixo, tem-se que $k = h - g$, onde h e g representam os valores de $h(x)$ e $g(x)$ para todo $x \in \{y \mid h(y) - g(y) = k\}$. Dessa forma $f(x)$ pode ser calculada em função de h.

$$f(h) = \frac{h - k}{h}.$$

Calculando a derivada,

$$f'(h) = \frac{h - (h - k)}{h^2} = \frac{k}{h^2}.$$

Se $k > 0$ a derivada é positiva. Assim, quanto maior h, maior será o valor de $f(h)$.

Seja então $\mathcal{P}(k)$ a versão parametrizada de $\mathcal{P}$, ou seja, uma versão em que se procura, dentre todas as soluções de $\mathcal{P}$, a que possui o maior valor da função objetivo e atende a restrição $h(x) - g(x) = k$. A formulação de $\mathcal{P}(k)$ para o caso em que $k > 0$ é definida como

$$\begin{aligned} \mathcal{P}(k): \quad & \max h(x) \\ \text{s. a} \quad & h(x) - g(x) = k \\ & x \in \mathcal{S}. \end{aligned}$$

Para os casos em que $k < 0$ a formulação será definida como

$$\begin{aligned} \mathcal{P}(k): \quad & \max g(x) \\ \text{s. a} \quad & h(x) - g(x) = k \\ & x \in \mathcal{S}. \end{aligned}$$

Além disso, seja $\mathcal{S}(k)$ o domínio de $\mathcal{P}(k)$. Ou seja, a divisão de $\mathcal{S}$ em todos os possíveis $\mathcal{S}(k)$ formam uma partição.

Seja x_k^* a solução de $\mathcal{P}(k)$. Pode-se concluir que ela é um limite inferior para o problema geral. Isso é fácil de ver pois se x_k^* é solução de $\mathcal{P}(k)$, também será uma solução de $\mathcal{P}$. Para calcular um limite superior do problema, pode-se utilizar a seguinte estratégia. Para os casos em que $k > 0$, seja $\ell' \leq g(x)$ o menor valor possível para g, $\forall x \in \mathcal{S}$. Define-se $\ell = \max(\ell', 1)$ e x_ℓ a solução na qual $g(x_\ell) = \ell$. Então,

$$f(x_\ell) = \frac{g(x_\ell)}{h(x_\ell)} = \frac{\ell}{k + \ell}$$

será um limite superior para a parte $\mathcal{S}(k)$ e também para qualquer parte $\mathcal{S}(\kappa)$ com $\kappa > k$.

Para os casos em que $k < 0$, tem-se que $h(x) \geq 1$, já que h possui o contradomínio discreto e por definição $h(x) > 0$ para todo x. Nesses casos, o objetivo é maximizar g e consequentemente minimizar h. Como $h(x_\ell) = 1$ é um limite para h, temos então que $f(x_\ell) = \frac{1-k}{1}$ é um limite superior para a parte $\mathcal{S}(k)$. Da mesma forma que no caso anterior, o limite decai até quando k for igual a zero. A Figura 1 mostra o comportamento dos limites superiores de acordo com o valor de k.

Com os limites e os subproblemas já definidos, pode-se então apresentar um pseudo-código do algoritmo ExIt (Exato Iterativo) representado pelo Algoritmo 1. O Algoritmo ExIt busca a solução ótima do problema $\mathcal{P}$ através de execuções de sua versão parametrizada. Para isso, primeiramente, se obtém k como a solução de $\mathcal{P}'$. A partir deste valor, iterativamente, se obtém o valor de $\mathcal{P}(k)$ e se incrementa k. A cada iteração atualizam-se os valores dos limites superior e inferior. Isso ocorre até que os limites se encontrem, significando que a partir desse ponto não existe nenhuma solução melhor. Logo a seguir, o Teorema 1 prova que o ExIt retorna uma solução ótima do PPLF.

Teorema 1. *O Algoritmo ExIt retorna uma solução ótima do problema $\mathcal{P}$.*

Demonstração. A ideia do ExIt (Algoritmo 1) consiste em buscar a solução ótima iterativamente através de execuções de sua versão parametrizada, tendo como início o número k retornado por $\mathcal{P}'$. A cada iteração novos limites são calculados. Isso ocorre até que os limites se encontrem, significando que a partir desse ponto não existe nenhuma solução melhor.

Sabe-se que as linhas 8, 9, 14, 24, 25 e 29 do Algoritmo 1 realmente calculam os limites inferior e superior. Falta então mostrar que as linhas 16 e 32 calculam de fato um limite superior e que o algoritmo para.

Algorithm 1 Algoritmo ExIt (Exato Iterativo)

procedure ECM(instância i, limite ℓ)
  $x^* \leftarrow \mathcal{P}'(i)$ ▷ Resolve o $\mathcal{P}'$ para a instância i
  if $x^* = \emptyset$ **then**
    return $\emptyset$
  end if
  $k \leftarrow h(x^*) - g(x^*)$
  if $k < 0$ **then**
    $LS_- \leftarrow 1 - k$
    $LI_- \leftarrow f(x^*)$
    while $LS > LI$ **do**
      $x_k^* \leftarrow \mathcal{P}(i, k)$ ▷ Resolve o $\mathcal{P}(k)$ para instância i
      if $f(x_k^*) > f(x^*)$ **then**
        $x^* \leftarrow x_k^*$
        $LI_- \leftarrow f(x^*)$
      end if
      $LS_- \leftarrow 1 - k$
      $k \leftarrow k + 1$
    end while
    if $f(x^*) < 1$ e $\mathcal{P}(i, 0) \neq \emptyset$ **then**
      $x^* \leftarrow \mathcal{P}(i, 0)$ ▷ Resolve o $\mathcal{P}(0)$ para instância i
    end if
    $k \leftarrow 1$
  end if
  $LS_+ \leftarrow \frac{\ell}{k+\ell}$
  $LI_+ \leftarrow f(x^*)$
  while $LS_+ > LI_+$ **do**
    $x_k^* \leftarrow \mathcal{P}(i, k)$ ▷ Resolve o $\mathcal{P}(k)$ para instância i
    if $f(x_k^*) > f(x^*)$ **then**
      $LI_+ \leftarrow f(x_k^*)$
      $x^* \leftarrow x_k^*$
    end if
    $LS_+ \leftarrow \frac{\ell}{\ell+k}$
    $k \leftarrow k + 1$
  end while
  return x^*
end procedure

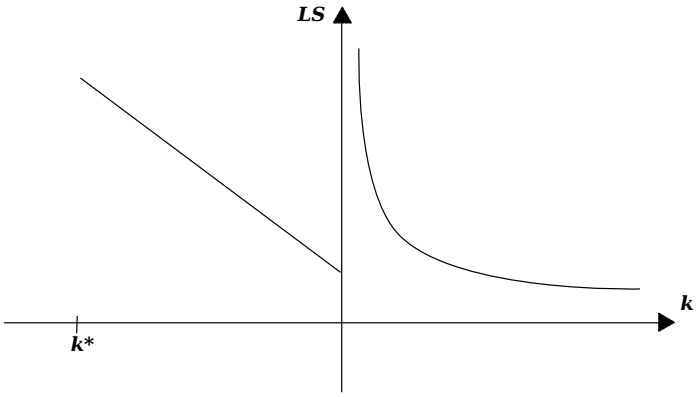

Figura 1: Limite Superior por k

Mesmo já sendo uma boa solução, a solução ótima de $\mathcal{P}'$ não é necessariamente ótima para $\mathcal{P}$. Neste caso, o ótimo de $\mathcal{P}$ pode possuir um k maior que o de $\mathcal{P}'$. Para encontrá-lo, utilizam-se as chamadas de $\mathcal{P}(k)$. Da forma como foi definida a função objetivo de $\mathcal{P}(k)$, obtém-se como retorno a solução com melhor função objetivo f dentre todas as soluções na parte $\mathcal{S}(k)$.

As linhas 16 e 32 calculam de fato um limite superior, pois se existisse uma solução acima deste novo limite e abaixo do antigo, o algoritmo já a teria encontrado em uma iteração anterior através de uma chamada de $\mathcal{P}(k)$. Como o Limite Superior (LS) diminui a cada iteração conclui-se que o algoritmo então converge. □

3 Aplicações

Nesta seção serão apresentados dois PPLFs e como aplicar o algoritmo ExIt nestes problemas. Após isso, na última parte, será apresentado um corte que melhorará o desempenho do método proposto. O primeiro PPLF é o Problema de Formação de Células de Manufatura (PFCM), para o qual foi criada a primeira versão do algoritmo em Pinheiro [5]. O segundo PPLF é uma generalização do Problema de Coloração em Grafos (PCG), chamada de Problema de Alocação de Canais com Prêmios (PACG) e apresentada primeiramente neste trabalho apenas como exemplificação para o método proposto.

3.1 O Problema de Formação de Células de Manufatura

Pode-se representar um sistema de manufatura por uma matriz binária parte–máquina em que seus elementos possuem valor 1 se a parte é fabricada pela máquina e 0 caso contrário. O objetivo do PFCM é formar células nas quais são agrupadas máquinas dedicadas à produção de uma família de partes. Para tal, deseja-se obter células em que as máquinas possuem um alto nível de similaridade entre si, isto é, produzem em comum o mesmo conjunto de partes destinadas à célula. Como consequência disso, obtém-se na prática uma redução no número de deslocamentos de partes entre células e um melhor aproveitamento das máquinas na célula, reduzindo consideravelmente os custos de fabricação [4, 7].

Para se obter essas células, fazemos permutações entre as linhas e colunas da matriz a fim de formar submatrizes de 1's na diagonal principal e submatrizes de 0's fora. A Figura 2 mostra um exemplo de uma célula de manufatura à esquerda e a clusterização formada pelas permutações à direita.

Existem diversas funções objetivo para este problema. A abordagem utilizada nos trabalhos

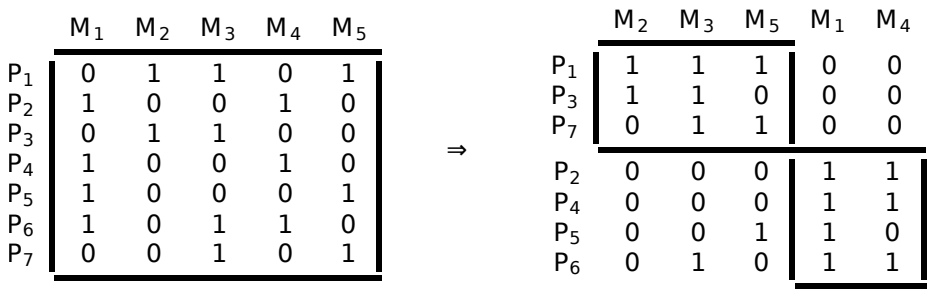

Figura 2: Exemplo PFCM

atuais é a eficácia de agrupamento μ, proposta por [3]. Pode-se defini-la como:

$$\mu = \frac{N_1 - N_1^{out}}{N_1 + N_0^{in}}.$$

em que:

- N_1 é o número de 1's na matriz original;
- N_1^{out} é o número de 1's fora das células;
- N_0^{in} é o número de 0's dentro das células.

Observe que o objetivo é maximizar a eficácia; assim, quanto mais próximo de 1 for este índice, melhor será o agrupamento. Segundo Pinheiro [5], uma formulação matemática para o problema pode ser definida como algo do tipo:

$$\begin{aligned} \mathcal{PFCM}: \quad & \max \quad \frac{m - \sum_{+(ij)} (1 - y_{ij})}{m + \sum_{-(ij)} y_{ij}} \\ & \text{s.a} \quad y \in \mathcal{S}. \end{aligned}$$

A versão linear $h(x) - g(x)$ é definida por:

$$\begin{aligned} \mathcal{PFCM}': \quad & \min \sum_{+(ij)} (1 - y_{ij}) + \sum_{-(ij)} y_{ij} \\ & \text{s.a} \quad y \in \mathcal{S}. \end{aligned}$$

E a versão parametrizada por:

$$\begin{aligned} \mathcal{PFCM}(k): \quad & \max \quad m + \sum_{-(ij)} y_{ij} \\ & \text{s.a} \sum_{+(ij)} (1 - y_{ij}) + \sum_{-(ij)} y_{ij} = k \\ & y \in \mathcal{S}. \end{aligned}$$

Para calcular ℓ basta ver que o valor mínimo é o próprio m, assim o limite superior para cada iteração passa a ser:

$$LS = \frac{m}{m + k},$$

já que k sempre será positivo.

3.2 Problema de Alocação de Canais com Prêmios

Propomos a seguir um novo PPLF chamado de Problema de Alocação de Canais com Prêmios (PACG) para exemplificar a forma em que o algoritmo ExIt pode ser aplicado.

Suponha que em uma cidade há uma certa demanda por serviços de telefonia móvel. Para atender esta demanda existe um conjunto de torres de transmissão que opera em um determinado conjunto de frequências. Devemos então atribuir para cada torre uma ou mais frequências de modo que torres dentro do raio de cobertura operem em frequências diferentes. Esta restrição deve ser considerada para que se evite a interferência entre as torres. Além disso, considere que existe um custo associado à contratação de uma frequência e, portanto, deseja-se minimizar o número de frequências contratadas. Em contrapartida, existe um benefício associado ao se utilizar um maior número de frequências em uma determinada torre, já que diminui-se a interferência entre os usuários na comunicação com esta torre.

Podemos considerar o PACG como uma generalização do Problema de Coloração em Grafos (PCG), um problema $\mathcal{NP}$-difícil e bem conhecido na literatura, definido a seguir. Seja $G(V, E)$ um grafo e $\mathcal{C}_i$ um conjunto de cores. O objetivo do problema é encontrar uma k-coloração em que $k \in \mathcal{N}^+$ seja o menor possível. Analogamente, no PACG temos que cada frequência será uma cor, e portanto, minimizar o número de cores utilizadas equivale a minimizar o número de frequências contratadas. Contudo, expandindo esse conceito podemos ainda considerar um custo c_{vi} e um prêmio p_{vi} associado à contratação da frequência i na torre v.

Temos portanto, a seguinte formulação para o problema, considerando o valor 1 a x_{vi} se foi atribuido frequência i à torre v e 0 caso contrário:

$$\begin{aligned} \mathcal{CG}: \quad & \max \frac{\sum_{v\in V}\sum_{i\in\mathcal{C}} c_{vi}\cdot x_{vi}}{\sum_{v\in V}\sum_{i\in\mathcal{C}} b_{vi}\cdot x_{vi}} & \\ \text{s. a} \quad & x_{vi} + x_{wi} \leq 1 & \forall w \in E | w = N(v), \forall i \in \mathcal{C} \\ & x_{vi} \in \{0,1\} & \forall v \in V, \forall i \in \mathcal{C}. \end{aligned}$$

Veja que o problema é difícil uma vez que é uma generalização do PCG. Uma estratégia para resolvê-lo é através da utilização do algoritmo ExIt. Para isso, deve-se encontrar sua versão linear e sua versão parametrizada.

A versão linear é dada por:

$$\begin{aligned} \mathcal{CG}': \quad & \min \sum_{v\in V}\sum_{i\in\mathcal{C}} b_{vi}\cdot x_{vi} - \sum_{v\in V}\sum_{i\in\mathcal{C}} c_{vi}\cdot x_{vi} \\ \text{s. a} \quad & x \in \mathcal{S}. \end{aligned}$$

E a versão parametrizada para $k > 0$ é dada por:

$$\begin{aligned} \mathcal{CG}(k): \quad & \max \sum_{i\in\mathcal{C}} c_{vi}\cdot x_{vi} \\ \text{s. a} \quad & \sum_{v\in V}\sum_{i\in\mathcal{C}} b_{vi}\cdot x_{vi} - \sum_{v\in V}\sum_{i\in\mathcal{C}} c_{vi}\cdot x_{vi} = k \\ & x \in \mathcal{S}. \end{aligned}$$

Quando $k < 0$ será:

$$\begin{aligned} \mathcal{CG}(k): \quad & \max \sum_{i \in \mathcal{C}} b_{vi} \cdot x_{vi} \\ \text{s. a} \quad & \sum_{v \in V} \sum_{i \in \mathcal{C}} b_{vi} \cdot x_{vi} - \sum_{v \in V} \sum_{i \in \mathcal{C}} c_{vi} \cdot x_{vi} = k \\ & x \in \mathcal{S}. \end{aligned}$$

Para que o algoritmo proposto possa ser aplicado, falta apenas obter o valor de ℓ.

$$\begin{aligned} \ell = \quad & \min \sum_{i \in \mathcal{C}} b_{vi} \cdot x_{vi} \\ \text{s. a} \quad & x \in \mathcal{S}. \end{aligned}$$

3.3 Otimização

Como visto na seção anterior, o algoritmo ExIt se baseia em iterações nas quais subproblemas são resolvidos. Uma vez que um dos subproblemas está resolvido, informações podem ser passadas ao subproblema seguinte. Nesta seção serão propostos alguns cortes que melhorarão o desempenho do algoritmo.

Considere que na iteração i o valor de $\mathcal{P}(k_i)$ foi V_i. Considere também que nesta parte do algoritmo temos $k > 0$. O próximo passo do algoritmo é calcular o valor de $\mathcal{P}(k_{i+1})$. Porém, ao calcular $\mathcal{P}(k_i)$, foi obtido o valor V_i, e agora no cálculo de $\mathcal{P}(k_{i+1})$ não interessa um valor menor que V_i. Pois, se o valor V_{i+1} for menor que V_i, então o valor da solução dessa nova iteração aplicado em f será menor que o da iteração anterior. Dessa forma o corte $h(x) \geq V_i + 1$ pode ser aplicado para as iterações seguintes, em que i representa a iteração em que foi encontrada a melhor solução vigente.

4 Resultados

Nesta seção serão apresentados os resultados computacionais referentes ao PFCM. As ferramentas utilizadas nos testes computacionais serão apresentadas na Seção 4.1 e os resultados na Seção 4.2.

4.1 Ferramentas

Como ferramenta utilizou-se o CPLEX, um dos pacotes de *software* de otimização linear mista mais utilizados na literatura. O CPLEX é responsável por gerenciar todo o processo de *Branch-and-Cut* (B&C).

Todas os métodos desenvolvidos foram implementados em C++ e executados em uma máquina Intel Core i7-2600 com 3,40 GHz e 32 GB de memória RAM no sistema operacional Arch Linux 3.3.4.

4.2 Testes computacionais

Para os experimentos, foi aplicado o algoritmo proposto na Seção 2 para a resolução do PFCM e as soluções encontradas comparadas com as obtidas pelos seguintes trabalhos da literatura:

Wu et al. [8]: método de coeficientes de similaridade com função de Boltzmann e operador de mutação;

Pailla et al. [4]: *simulated annealing*;

Elbenani et al. [2]: algoritmo genético híbrido.

A Tabela 1 compara o ótimo obtido pelo algoritmo proposto com o melhor resultado da literatura. Podemos observar que o algoritmo encontrou o ótimo em 26 de 35 instâncias. Em 3 instâncias, o algoritmo encontra soluções ainda desconhecidas na literatura (King1980, Kumar1987 e Stanfel1985_1). Nas demais instâncias, o algoritmo não obteve resultados. Vale salientar que houve instâncias na literatura em que foram encontrados resultados "melhores que os exatos". Isso se deve por conta de alguns erros de digitação das instância por parte de alguns autores, relatados em Pinheiro [5]. Os tamanhos das instâncias variam de 5 x 7 até 40 x 100 partes–máquinas.

Tabela 1: Resultado do Algoritmo Exato.

Instância	Literatura	Exato	Instância	Literatura	Exato
King1982	75	75	Kumar1986	50.81	50.81
Waghodekar1984	69.57	69.57	Carrie1973b	78.40	78.40
Seiffodini1989	80.85	80.85	Boe1991	58.79	58.79
Kusiak1992	79.17	79.17	Chandrasekharan1989_1	100	100
Kusiak1987	60.87	60.87	Chandrasekharan1989_2	85.11	85.11
Boctor1991	70.83	70.83	Chandrasekharan1989_3-4	73.51	73.50
Seiffodini1986	69.44	69.44	Chandrasekharan1989_5	53.29	
Chandrasekaran1986a	85.25	85.25	Chandrasekharan1989_6	48.63	
Chandrasekaran1986b	58.72	58.72	Chandrasekharan1989_7	46.15	
Mosier1985a	75	75	McCormick1972b	54.82	
Chan1982	92	92	Carrie1973c	47.68	
Askin1987	74.24	74.24	Kumar1987[1]	62.86	**63.04**
Stanfel1985	72.86	72.86	Stanfel1985_1[1]	59.66	**59.77**
McCornick1972a	53.33	53.33	Stanfel1985_2	50.83	
Srinivasan1990	69.92	69.92	King1982	47.93	
King1980	57.96	**58.04**	McCormick1972c	61.16	
Carrie1973a	57.73	57.73	Chandrasekharan1987	84.03	84.03
Mosier1985b	43.26				

5 Conclusões

Neste trabalho foi apresentado um novo algoritmo exato para uma classe de PPLFs. Esta classe engloba muitos problemas fracionários estudados na prática. Dentre as principais vantagens da utilização deste método em comparação com os outros métodos da literatura pode-se citar:

- uso menor de memória se comparado com os métodos de linearização;
- foco na estrutura do problema original;
- simplicidade de implementação.

Pode-se destacar o menor uso de memória como a principal vantagem do método. Uma vez que o consumo de memória cresce de forma exponencial em problemas combinatórios, a utilização de um método com menor consumo faz grande diferença entre encontrar a solução ótima ou não. Apesar de conter diversas chamadas de subproblemas, o método proposto pode ter um desempenho computacional competitivo através da utilização de:

- cortes a cada iteração;
- paralelização do próprio algoritmo;
- resolução bem trabalhada dos subproblemas como plano de corte, geração de colunas, etc.

Como trabalhos futuros pretende-se comparar o algoritmo proposto com os algoritmo clássicos da literatura para problemas fracionários.

Referências

[1] **Charnes, A. & Cooper, W. W.** (1962), 'Programming with linear fractional functionals', *Naval Research Logistics Quarterly* **9**, 181–196.

[2] **Elbenani, B., Ferland, J. A. & Bellemare, J.** (2012), 'Genetic algorithm and large neighbourhood search to solve the cell formation problem', *Expert Systems with Applications* **39**(3), 2408–2414.

[3] **Kumar, C. S. & Chandrasekharan, M. P.** (1990), 'Group efficacy: a quantitative criterion for goodness of block diagonal forms of binary matrices in group technology', *International Journal of Production Research* **28**(2), 233–243.

[4] **Pailla, A., Trindade, A. R., Parada, V. & Ochi, L. S.** (2010), 'A numerical comparison between simulated annealing and evolutionary approaches to the cell formation problem', *Expert Syst. Appl.* **37**, 5476–5483.

[5] **Pinheiro, R. G. S.** (2012), Método exato para biclusterização por edição de arestas e aplicação em formação de células de manufatura, Dissertação de Mestrado, Universidade Federal Fluminense.

[6] **Radzik, T.** (1998), *Handbook of Combinatorial Optimization*, Kluwer Academic Publishers, chapter Fractional Combinatorial Optimization, pp. 429–478.

[7] **Trindade, A. R. & Ochi, L. S.** (2006), 'Um algoritmo evolutivo híbrido para a formação de células de manufatura em sistemas de produção', *Pesquisa Operacional* **26**(2), 255–294.

[8] **Wu, T. H., Chang, C. C. & Y., Y. J.** (2009), 'A hybrid heuristic algorithm adopting both boltzmann function and mutation operator for manufacturing cell formation problems', *International Journal of Production Economics* **120**(2), 669–688.