LIII Simpósio Brasileiro de Pesquisa Operacional
João Pessoa - PB, 3 a 5 de novembro de 2021

Heurísticas de decomposição para o problema de dimensionamento de lotes com múltiplas plantas

Fernanda Yuka Ueno
Maristela Oliveira Santos
Instituto de Ciências Matemáticas e de Computação (ICMC), Universidade de São Paulo
Av. Trabalhador São-carlense, 400 - Centro, CEP 13566-590, São Carlos - SP.
fernanda.ueno@usp.br
mari@icmc.usp.br

Marcos Furlan
Faculdade de Ciências Exatas e Tecnologias, Universidade Federal da Grande Dourados
Rod. Dourados - Itahum, KM 12, CEP 79804-970, Dourados - MS
marcosfurlan@ufgd.edu.br

RESUMO

Neste trabalho abordamos o problema de dimensionamento de lotes com múltiplas plantas distintas e com limitações de capacidade para produção de diversos itens, os quais são utilizados para o atendimento da demanda sem atrasos. Para lidar com o problema, são propostas heurísticas baseadas na partição do conjunto de variáveis do modelo matemático. Essas heurísticas são do tipo *fix-and-optimize* com partição clássica por período e ADN (*Automatically designed neighborhoods*), que constrói uma vizinhança de forma automática utilizando aprendizado de máquina não supervisionado. Neste caso, usamos um algoritmo de agrupamento. Para realizar a comparação entre as heurísticas, foram utilizadas instâncias da literatura e as soluções são comparadas com as soluções obtidas por um otimizador comercial.

PALAVRAS-CHAVE. Problema de dimensionamento de lotes, Heurística *fix-and-optimize*, Aprendizado de máquina não supervisionado.

Tópicos: MH – Metaheurísticas, OC – Otimização Combinatória

ABSTRACT

In this paper, we deal with the capacitated multi-plant lot-sizing problem with multiple periods and items. We proposed heuristics based on the partition of the set of variables of the mathematical model of the problem. These heuristics are the classic fix-and-optimize and ADN (Automatically designed neighborhoods) that builds a neighborhood automatically using unsupervised machine learning. In this case, using clustering algorithms. To perform the comparison between the heuristics, instances of the literature were used, and the solutions are compared with the solutions obtained by a commercial optimizer.

KEYWORDS. Lot sizing problem, Fix-and-optimize, Unsupervised machine learning.

Paper topics: MH – Metaheuristics, OC – Combinatorial optimization

1. Introdução

O problema de dimensionamento de lotes consiste em determinar as quantidades de itens a serem produzidas em cada período ao longo de um horizonte de planejamento finito, de forma que atenda as demandas e utilize os recursos disponíveis da melhor maneira possível. Neste trabalho foi considerado o problema de dimensionamento de lotes múltiplas plantas com restrição de capacidade com múltiplos períodos e itens, conhecido como MPCLSP (*capacitated multi-plant lot sizing problem*). O objetivo deste problema é determinar um plano de produção que minimize os custos de produção, de estoque, de preparação para produção dos itens e de transferência de itens entre plantas, visando o atendimento da demanda de cada planta em cada período de tempo. Podemos encontrar diversos trabalhos que lidam com este problema como, por exemplo, Melega et al. [2013] investigaram formulações fortes a partir da formulação proposta por Sambasivan e Schmidt [2002], Nascimento et al. [2010] desenvolveram uma heurística GRASP (Greedy Randomized Adaptive Search), Carvalho e Nascimento [2016] usaram heurística lagrangiana e Ghiani et al. [2015] propuseram uma heurística baseada em vizinhança.

Na literatura, para problemas de dimensionamento de lotes, existem diversas abordagens exatas e heurísticas que visam a obtenção de planos de produção factíveis. Dentre as abordagens heurísticas, uma das mais utilizadas são as heurísticas baseadas no modelo de programação inteira mista, denominadas MIP-based heurísticas. Dentre elas, umas das mais conhecidas são as construtivas *relax-and-fix* e de melhoria *fix-and-optimize* (FO), as quais são baseadas na partição do conjunto de variáveis do modelo matemático. Normalmente são escolhidas as variáveis binárias que são particionadas considerando períodos, itens ou recursos produtivos. Podemos observar a utilização dessa abordagem, por exemplo, em Furlan [2011] e em Toledo et al. [2015]. Recentemente, Ghiani et al. [2015] utilizaram um algoritmo de agrupamento para determinar partições de forma não intuitiva em um procedimento de melhoria baseado em modelagem semelhante ao procedimento do tipo FO.

Atualmente, o aprendizado de máquina está sendo aplicado em diversas áreas com sucesso e vem sendo usado na área de otimização com resultados promissores. Por isso investigamos, neste artigo, as heurísticas que combinam técnicas de aprendizado de máquina, para auxiliar na determinação de partições não intuitivas das variáveis binárias utilizando os algoritmos de agrupamento. Estas novas estratégias são adaptações e/ou implementações das propostas em Ghiani et al. [2015]. Para analisar a qualidade das soluções obtidas utilizando estas estratégias de partição, propomos heurísticas, a partir de uma solução inicial factível, utilizando diferentes estratégias de partição das variáveis, bem como diferentes formas de utilização das partições, com o objetivo de melhorar as soluções. As soluções iniciais são obtidas a partir de um resolvedor de otimização como feito em Ghiani et al. [2015]. Para analisar o desempenho destes procedimentos consideramos o problema o MPCLSP, problema clássico da literatura e com base de dados disponíveis.

Este trabalho apresenta, na Seção 2, a formulação matemática do MPCLSP, na Seção 3, as abordagens de solução desenvolvidas, na Seção 4, os testes computacionais e por último, na Seção 5, as conclusões e as perspectivas.

2. Formulação matemática

A seguir será apresentado o modelo matemático proposto em Sambasivan e Schmidt [2002] para o MPCLSP. Neste problema considerou-se que todos os itens podem ser produzidos em todas as plantas, mas pode ocorrer transferência de produção entre fábricas para o atendimento da demanda. Os custos de preparação de máquina, produção de itens e estoque variam conforme os itens, a planta e o período de planejamento. Já os custos de transferência entre plantas variam conforme as plantas envolvidas e o período.

Conjuntos de índices :

- N: conjunto de índices dos itens;
- T: conjunto de índices dos períodos;
- M: conjunto de índices das plantas.

Parâmetros :

- cap_{jt}: capacidade produtiva da planta j no período t, com $j \in M$ e $t \in T$;
- b_{ijt}: tempo de produção do item i na planta j no período t, com $i \in N$, $j \in M$ e $t \in T$;
- f_{ijt}: tempo de preparação do item i na planta j no período t, com $i \in N$, $j \in M$ e $t \in T$;
- s_{ijt}: custo de preparação do item i na planta j no período t, com $i \in N$, $j \in M$ e $t \in T$;
- c_{ijt}: custo de produção do item i na planta j no período t, com $i \in N$, $j \in M$ e $t \in T$;
- h_{ijt}: custo de estoque do item i na planta j no período t, com $i \in N$, $j \in M$ e $t \in T$;
- d_{ijt}: demanda do item i na planta j no período t, com $i \in N$, $j \in M$ e $t \in T$;
- $r_{jj't}$: custo de transferência de uma unidade de um item da planta j para a planta j' no período t, com $j, j' \in M$ e $t \in T$;

Variáveis de decisão :

- X_{ijt}: quantidade do item i produzida na planta j no período t, com $i \in N$, $j \in M$ e $t \in T$;
- I_{ijt}: quantidade do item i em estoque na planta j no período t, com $i \in N$, $j \in M$ e $t \in T$;
- $Q_{ijj't}$: quantidade do item i transferido da planta j para a planta j' no período t, com $i \in N$, $j, j' \in M$ e $t \in T$;
- Y_{ijt}: Variável binária no qual é igual a 1 se o item i é produzido na planta j e no período t, caso contrário é igual a 0, com $i \in N$, $j \in M$ e $t \in T$.

Função objetivo e restrições :

$$\text{Min} \quad \sum_{i\in N}\sum_{j\in M}\sum_{t\in T}(c_{ijt}X_{ijt} + s_{ijt}Y_{ijt} + h_{ijt}I_{ijt} + \sum_{j'\in J, j'\neq j} r_{jj't}Q_{ijj't}) \tag{1}$$

$$\text{s.a.} \quad I_{i,j,t-1} + X_{ijt} - (\sum_{j'\in M, j'\neq j} Q_{ijj't}) + (\sum_{j'\in M, j'\neq j} Q_{ij'jt}) - I_{ijt} = d_{ijt}, \qquad \forall(i,j,t), \tag{2}$$

$$X_{ijl} \leq (\sum_{l, l\geq t} d_{ijt})Y_{ijt}, \qquad \forall(i,j,t), \tag{3}$$

$$\sum_{i\in N}(b_{ijt}X_{ijt} + f_{ijt}Y_{ijt}) \leq cap_{jt}, \qquad \forall(j,t), \tag{4}$$

$$I_{ij0} = 0, \qquad \forall(i,j), \tag{5}$$

$$X_{ijt}, I_{ijt}, Q_{ijj't} \geq 0, \qquad \forall(i,j,j',t), \tag{6}$$

$$Y_{ijt} \in \{0,1\}, \qquad \forall(i,j,t), \tag{7}$$

A função objetivo (1) visa a minimização dos custos de produção, de preparação, de estoque e dos custos das transferências entre plantas. Deve-se produzir todos os itens necessários nas plantas para atender a demanda ao longo do horizonte de planejamento, respeitando às restrições dadas por (2) a (7).

As restrições (2) são responsáveis pelo balanceamento do estoque de cada item i em cada período t e planta j para garantir que a demanda (d_{ijt}) seja atendida considerando a quantidade estocada no período anterior ($I_{i,j,t-1}$), a quantidade do item i produzida (X_{ijt}), a quantidade do item i transferida da planta j para outras plantas ($\sum_{j'\in M, j'\neq j} Q_{ijj't}$), a quantidade do item i transferida de outras plantas para a planta j ($\sum_{j'\in M, j'\neq j} Q_{ij'jt}$) e a quantidade do item i que será estocado no fim do período atual (I_{ijt}).

As restrições (3) garantem que se houver produção do item i na planta j e no período t, a variável binária (Y_{ijt}) será igual a 1, assim, a preparação do item i na planta j no período t é considerada. As restrições (4) garantem que a solução não ultrapasse o limite da capacidade de produção da planta j no período t. As restrições (5) indicam que não têm estoques iniciais ($t = 0$). As restrições (6) e (7) definem o domínio das variáveis.

3. Abordagens da solução

As abordagens heurísticas propostas para lidar com o MPCLSP consistem de duas etapas: construção (obtenção) de uma solução inicial e melhoria. A solução inicial utilizada é a primeira solução obtida através de um resolvedor de otimização. Os procedimentos de melhoria são baseados na decomposição do conjunto de variáveis binárias e na resolução de subproblemas inteiros menores utilizando um resolvedor de otimização. Como procedimentos de melhoria, consideramos a heurística FO, na sua versão clássica, bastante usada na literatura para diversos problemas de planejamento da produção. A outra é a heurística ADN, que também é uma heurística do tipo

FO, porém, constrói uma vizinhança de forma automática utilizando aprendizado de máquina não supervisionado, a qual é utilizada para particionar os conjuntos de variáveis binárias. Nas próximas subseções serão apresentadas as heurísticas de melhoria com mais detalhes.

3.1. *Fix-and-Optimize* (FO)

A heurística FO foi inicialmente proposta para problemas de planejamento da produção em Pochet e Wolsey [2006] com o nome de *exchange* e, posteriormente, foi renomeada em Helber e Sahling [2010]. A ideia principal por trás dos métodos FO é que a partir do particionamento das variáveis do problema, podemos resolver subproblemas mais simples, de forma iterativa, gerando melhorias na solução incumbente. Portanto, partindo de uma solução inicial, o objetivo do método é gerar melhorias através da fixação de parte das variáveis e otimização dos subproblemas resultantes. Para aplicar a heurística, o conjunto de variáveis inteiras e/ou binárias Q são particionadas em R subconjuntos $Q_1, ..., Q_R$.

Na heurística, em cada iteração r, com $1 \leq r \leq R$, é definido um sub-PIM (subproblema inteiro e misto), em que o conjunto de variáveis em Q_r são mantidas no domínio original e o restante fixadas com a solução incumbente. O sub-PIM obtido é resolvido e caso tenha obtido uma solução melhor do que a atual, essa solução passa ser a nova solução incumbente, caso não encontre uma solução de melhor qualidade, após um certo número de iterações ou após um limite de tempo, o método é interrompido.

Neste trabalho foi implementado a versão do FO, baseado em Furlan [2011], em que as variáveis são particionadas por período. Nesta heurística, pode-se particionar as variáveis por períodos, itens, recursos ou fábricas, porém, neste trabalho, consideramos apenas a partição clássica das variáveis por período. O particionamento por período foi escolhido para comparar diretamente com os particionamentos usados na heurística ADN, também por período. Desta forma, pode-se comparar o desempenho das heurísticas do tipo FO na sua versão clássica e com particionamentos obtidos através do aprendizado de máquina não supervisionado.

3.2. ADN

Ghiani et al. [2015] desenvolveram a heurística ADN, que consiste em extrair informações da estrutura do modelo e de uma solução corrente para serem utilizados por um algoritmo de aprendizado não supervisionado para auxiliar na definição da vizinhança.

O primeiro passo da heurística consiste em definir um conjunto de índices que estão relacionados com as variáveis binárias e/ou inteiras, chamado conjunto de entidades fundamentais E que podem ser divididas em subconjuntos disjuntos. No problema MPCLSP, foco de estudo deste trabalho, temos apenas as variáveis Y_{ijt} com domínio de valores binários. Essas variáveis estão relacionadas a três índices, contidos nos conjuntos N, M e T, respectivamente. Desta forma, o conjunto de entidades fundamentais é definido como $E = E_1 \cup E_2 \cup E_3$, nos quais $E_1 = N$, $E_2 = M$ e $E_3 = T$.

Para extrair as informações da estrutura do modelo e da solução corrente ($x^{(h)}$, solução atual) é construído um grafo de adjacência de entidades separadamente para cada subconjunto de entidades, em que as arestas são as entidades e existe um arco entre as arestas se as entidades são adjacentes, ou seja, possui um peso entre eles indicando se possuem alguma similaridade. Dados $e, e' \in E$, o peso entre a entidade e e e' é definido por duas medidas de similaridade, sendo a primeira baseada na solução corrente ($S_{ee'}(x^{(h)})$) e a segunda baseada na estrutura do modelo ($M_{ee'}$). Assim o peso é igual a 1 se $S_{ee'}(x^{(h)}) > 0$; igual a $|E|$ se $S_{ee'}(x^{(h)}) = 0$ e $M_{ee'} > 0$; caso contrário não existe peso entre essas entidades (para mais detalhes ver em Ghiani et al. [2015]).

A partir desse grafo é construído a matriz de dissimilaridade em que para cada par de entidade é calculado a menor distância entre eles sobre o grafo. Observa-se que as entidades são

mais dissimilares entre si quanto maior sua distância. Neste trabalho para obter essa matriz foi utilizado o algoritmo de *Floyd-Warshall* implementado na biblioteca *SciPy* do *Python*. Essa matriz é utilizada pelo algoritmo não supervisionado para agrupar as entidades de cada subconjunto de entidades separadamente. Neste trabalho foi escolhido o algoritmo de agrupamento clássico *k-means*, que também foi modificado de forma que obtenha os agrupamentos balanceados, ou seja, obtenha em todos os agrupamentos com aproximadamente a mesma quantidade de entidades.

Neste trabalho foi implementado o *k-means* baseado em Jain e Dubes [1988] e o algoritmo é descrito a seguir:

***K-means*:** dado k (número de agrupamentos)

- Passo 1: define um conjunto de centroides iniciais, ou seja, seleciona aleatoriamente k entidades dentre o subconjunto de entidades a ser agrupado.
- Passo 2: designa as entidades que estão mais próximas dentre os centroides e assim formamos k agrupamentos.
- Passo 3: calcula novos centroides que são as médias das posições das entidades de cada agrupamento.
- Passo 4: repete o passo 2 e 3 até que os centroides não variem com uma certa tolerância ou atinjam o número máximo de iterações.

Para o *k-means* balanceado, a diferença é apenas no passo 2 do algoritmo *k-means*. O algoritmo designa os elementos que estão mais próximos dos centroides dos agrupamentos que não atingiram o número máximo de elementos. Assim obtemos agrupamentos que possuem aproximadamente a mesma quantidade de elementos, como feita em Ghiani et al. [2015] que utilizam apenas o *k-means* balanceado.

Após agrupar cada subconjunto de entidades, é feita a combinação desses agrupamentos. Por exemplo, para o MPCLSP, Ghiani et al. [2015] agruparam apenas o subconjunto E_3 (agrupamento por períodos) em 6 agrupamentos $E_3^{(1)}, E_3^{(2)}, ..., E_3^{(6)}$ e assim temos 6 combinações de agrupamentos $(E_1 \cup E_2 \cup E_3^{(1)})$, $(E_1 \cup E_2 \cup E_3^{(2)})$, ..., $(E_1 \cup E_2 \cup E_3^{(6)})$. Ou seja, os autores utilizaram apenas decomposição por períodos, deste modo, neste artigo utilizaremos apenas este tipo de decomposição.

A partir dessas combinações de agrupamentos, em Ghiani et al. [2015] foram definidos dois tipos de vizinhança, uma é chamada vizinhança básica formada por todas as soluções de sub-PIMs, nos quais cada sub-PIM está relacionado com uma combinação de agrupamentos de forma que todas as variáveis que possuem os índices nessa combinação de agrupamentos são mantidas no domínio original e o restante fixadas com a solução corrente. E a outra é a vizinhança de intensificação definida de forma igual à básica, mas antes é feita a redução do efeito de borda com o objetivo de melhorar a solução localmente.

Para reduzir o efeito de borda, primeiro é definido o conjunto de entidades de borda que são entidades adjacentes que pertencem a agrupamentos diferentes. Por exemplo, dado $e, e' \in E$, com $e \in E_3^{(3)}$ e $e' \in E_3^{(4)}$, caso exista aresta entre eles no grafo, então eles são adjacentes e consequentemente pertencem ao conjunto de entidades de borda. Depois é feito o reagrupamento de cada subconjunto considerando o conjunto de entidades de borda para a escolha dos centroides iniciais no algoritmo de agrupamento, assim a ideia é diminuir o número de entidades no conjunto de borda, ou seja, reduzir o efeito de borda.

Resumidamente, no algoritmo ADN apresentado em Ghiani et al. [2015] é dado uma solução inicial e um tempo limite. O algoritmo é dividido em duas fases, na primeira fase utiliza a metade do tempo limite para explorar o espaço de busca considerando a vizinhança básica e na segunda fase utiliza o tempo limite restante para intensificar a busca considerando a vizinhança de intensificação.

Baseado em Ghiani et al. [2015] foram implementadas três versões do ADN e em todas é dado uma solução inicial e um tempo limite:

ADN1:

Fase 1: define a vizinhança básica com a solução inicial utilizando metade do tempo limite.

Fase 2: define a vizinhança de intensificação com a melhor solução encontrada na fase 1 utilizando o restante do tempo limite.

ADN2:

Fase 1:

- **Passo 1.1:** agrupa os subconjuntos de entidades.
- **Passo 1.2:** utiliza a solução incumbente para definir a vizinhança básica utilizando a combinação de agrupamentos encontrados no passo 1.1.
- **Passo 1.3:** repete o passo 1.2 até que a solução incumbente não melhore ou atinja o restante da metade do tempo limite.

Fase 2:

- **Passo 2.1:** reagrupa os subconjuntos de entidades reduzindo o efeito de borda.
- **Passo 2.2:** utiliza a solução incumbente para definir a vizinhança de intensificação utilizando a combinação de agrupamentos encontrados no passo 2.1.
- **Passo 2.3:** repete o passo 2.2 até que a solução não melhore ou atinja o resto do tempo limite.

ADN3:

- **Passo 1.1:** agrupa os subconjuntos de entidades.
- **Passo 1.2:** utiliza a solução incumbente para definir a vizinhança básica utilizando a combinação de agrupamentos encontrados no passo 1.1. Ao resolver um sub-PIM e a solução dele for melhor que a incumbente, então a solução incumbente é atualizada.
- **Passo 1.3:** agrupa os subconjuntos de entidades considerando a solução incumbente.
- **Passo 1.4:** repete o passo 1.2 e 1.3 até que a solução não melhore ou atinja o tempo limite.

4. Testes computacionais

4.1. Ambiente de testes e instâncias

Nesta seção vamos apresentar alguns resultados dos experimentos computacionais que foram realizados utilizando o FO e as três versões do ADN (ADN1, ADN2 e ADN3) utilizando o

k-means e o *k-means* balanceado para resolver o MPCLSP apresentado na Seção 2. As heurísticas foram implementadas em *Python 3* e o resolvedor comercial utilizado é o *Gurobi 9.0.1*, na forma padrão. Os experimentos foram realizados em um computador *Intel Core* i7-7700 com $3,60$ GHz, $15,6$ GB de memória RAM e sistema operacional Ubuntu 18.04.5.

As heurísticas foram testadas nas instâncias de Carvalho e Nascimento [2016], as quais possuem diferentes parâmetros indicando o nível de capacidade disponível (apertada ou normal), custo de preparação (alto ou baixo) e tempo de preparação (alto ou baixo). Essas instâncias são divididas em 8 tipos e para cada instância, os parâmetros foram gerados a partir de intervalos definidos. Dentre todas as instâncias, foram escolhidas aquelas que apresentaram a maior dificuldade para a resolução utilizando o resolvedor *Gurobi* em um tempo limite de 900 segundos. Foram escolhidas 30 instâncias que atingiram o tempo limite e para as quais não foi possível obter solução ótima. Essas instâncias possuem número de períodos igual a 12 e número de plantas igual a 20, dentre elas, 10 instâncias com 70 itens, 10 instâncias com 80 itens e 10 instâncias com 90 itens.

4.2. Análise das soluções: Modelo Matemático

Como base para a comparação dos resultados das heurísticas, foram utilizadas as soluções do *Gurobi* com o tempo limite de 900 segundos. A Tabela 1 apresenta os resultados obtidos pelo *Gurobi* para instâncias com 70, 80 e 90 itens, em que para cada instância foi calculado o Gap obtido pelo resolvedor que é igual a $(\frac{|(z_P - z_D)|}{|z_P|} \cdot 100)\%$, sendo z_P o limitante superior e z_D o limitante inferior. Nessa tabela é apresentado o menor Gap (Gap_{min}), o maior Gap (Gap_{max}) e o Gap médio (Gap_{med}) entre as dez instâncias de cada grupo. Na Figura 1 é apresentado o gráfico *Box plot* (Diagrama de caixa) do Gap obtido pelo *Gurobi* para instâncias com 70, 80 e 90 itens. Todas as instâncias obtiveram os valores de Gap bem próximos de zero e atingiram o tempo limite de 900 segundos, mas o solver não provou a otimalidade das soluções encontradas.

Tabela 1: Gap entre o limitante inferior e superior obtido pelo *Gurobi* com tempo limite de 900 segundos para instâncias com 70, 80 e 90 itens.

N de itens	Gap_{min} (%)	Gap_{max} (%)	Gap_{med} (%)
70	0,07	0,14	0,11
80	0,08	0,18	0,11
90	0,09	0,15	0,11

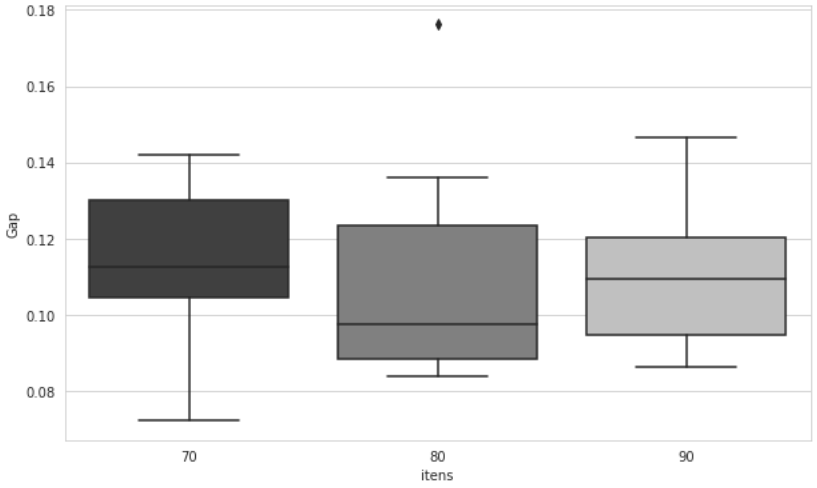
4.3. Análise das soluções: Heurísticas

Para as heurísticas FO e ADN foram utilizadas o tempo limite igual a 900 segundos e para resolver os sub-PIMs foi utilizado o *Gurobi*. A solução inicial utilizada é a primeira solução factível encontrada pelo método *branch-and-cut* do resolvedor de otimização *Gurobi* .

Na FO, o número máximo de iterações é igual a 1000, o tempo para resolver cada sub-PIM é igual ao número de variáveis livres multiplicado por $0,1$ segundos e escolhido o conjunto de índices dos períodos T para ser particionado de forma sequencial, com o tamanho das partições igual a $w = 1$, $w = 2$ e $w = 3$. Por exemplo, para o conjunto $T = \{1, 2, ..., 12\}$ e $w = 3$ teremos partições da seguinte forma, $P_1 = \{1, 2, 3\}$, $P_2 = \{4, 5, 6\}$, $P_3 = \{7, 8, 9\}$ e $P_4 = \{10, 11, 12\}$. Então teremos os subconjuntos de variáveis binárias Q_r relacionados com os índices pertencentes a P_r, com $1 \leq r \leq 4$. Assim, as heurísticas FO resolvem os sub-PIMs de forma sequencial, ou seja, primeiro o sub-PIM relacionado a Q_1, depois Q_2, e assim por diante. Definimos FO1 como sendo o algoritmo FO com $w = 1$, FO2 com $w = 2$ e FO3 com $w = 3$.

Na Tabela 2 são apresentados os resultados obtidos pelas heurísticas FO para instâncias com 70, 80 e 90 itens, em que temos o GAP igual a $(\frac{objValFO - objValGurobi}{objValFO} \cdot 100)\%$, sendo

Figura 1: *Box plot* do Gap entre o limitante inferior e superior obtido pelo *Gurobi* com tempo limite de 900 para instâncias com 70, 80 e 90 itens.



$objValFO$ o valor da função objetivo obtido pelo FO e $objValGurobi$ o valor da função objetivo obtido pelo *Gurobi*. Nessa tabela é apresentado o menor GAP (GAPmin), o maior GAP (GAP$_{max}$) e o GAP médio (GAP$_{med}$) entre as instâncias com 70, 80 e 90 itens. Também é apresentado o tempo mínimo (T$_{min}$), tempo máximo (T$_{max}$) e tempo médio (T$_{med}$) gasto pelas heurísticas. Para facilitar a visualização foi destacado, em negrito, os menores valores em cada coluna do GAP e do tempo (T), exceto quando o tempo limite foi atingido, por exemplo, instâncias com 70 itens, o FO2 obteve o menor GAP$_{min}$, GAP$_{max}$, GAP$_{med}$.

Observando a Tabela 2 para as instâncias com 70 e 90 itens temos que o FO2 obteve valores do GAP$_{med}$ menores e para instâncias com 80 itens temos o F03 obteve valores do GAP$_{med}$ menores. Em relação ao T$_{med}$, para instâncias com 70 e 80 itens, o FO1 obteve valores menores do GAP. Em alguns casos, as heurísticas FO1, FO2 e FO3 conseguiram encontrar uma solução melhor do que a solução encontrada pelo *Gurobi*, por isso, temos GAP com valores negativos.

Tabela 2: Resultados obtidos pelas heurísticas FO1, FO2 e FO3, com o tempo limite igual a 900 para instâncias com 70, 80 e 90 itens. O GAP é em relação ao valor da função objetivo obtido pela heurística e o valor da função objetivo obtido pelo *Gurobi*.

FO	itens	GAP$_{min}$ (%)	GAP$_{max}$ (%)	GAP$_{med}$ (%)	T$_{min}$ (s)	T$_{max}$ (s)	T$_{med}$ (s)
FO1	70	-0,0217	0,0438	0,0075	**838,19**	900,00	**893,86**
	80	-0,0110	0,0344	0,0112	**727,69**	900,00	**865,95**
	90	-0,0095	0,0333	0,0156	900,00	900,00	900,00
FO2	70	**-0,0390**	**0,0163**	**-0,0027**	900,00	900,00	900,00
	80	-0,0526	0,0328	-0,0030	900,00	900,00	900,00
	90	**-0,0284**	0,0299	**0,0031**	900,00	900,00	900,00
FO3	70	-0,0378	0,0297	0,0032	900,00	900,00	900,00
	80	**-0,0645**	**0,0252**	**-0,0054**	900,00	900,00	900,00
	90	-0,0093	0,0396	0,0152	900,00	900,00	900,00

Para os algoritmos *k-means* e o *k-means* balanceado, utilizados pelo ADN, foi considerado

o número máximo de iterações igual a 300, a tolerância igual a 10^{-4}. Como proposto em Ghiani et al. [2015], as variáveis foram agrupadas apenas por períodos T e o número de agrupamentos é igual a $k = 6$. Definimos ADN1K como sendo o algoritmo ADN1 que usou o *k-means* (K), ADN1KB como sendo a heurística que usou o *k-means* balanceado (KB) e o mesmo vale para ADN2K, ADN2KB, ADN3K e ADN3KB.

Na Tabela 3 são apresentados os resultados obtidos pelas heurísticas ADN, para instâncias com 70, 80 e 90 itens, em que temos o GAP (calculado da mesma forma para o FO) igual a $(\frac{objValADN - objValGurobi}{objValADN} \cdot 100)\%$, com $objValADN$ o valor da função objetivo obtido pelo ADN e $objValGurobi$ o valor da função objetivo obtido pelo *Gurobi*. Nesta tabela temos o menor GAP (GAPmin), o maior GAP (GAP$_{max}$) e o GAP médio (GAP$_{med}$) entre as instâncias com 70, 80 e 90 itens. Também temos o tempo mínimo (T$_{min}$), tempo máximo (T$_{max}$) e tempo médio (T$_{med}$) gasto pelas heurísticas. Para facilitar a visualização foi destacado, em negrito, os melhores valores em cada coluna do GAP e do tempo (T) para cada grupo de instâncias, assim como foi feito para a Tabela 2.

Observando a Tabela 3, para todas as instâncias temos que o ADN3K obteve valores de GAP$_{med}$ menores e, em relação ao tempo, para todas as instâncias o ADN1KB obteve valores de T$_{med}$ menores. Aqui também houve alguns casos em que ADN2K, ADN3K e ADN3KB conseguiram encontrar uma solução melhor do que a solução encontrada pelo *Gurobi* por isso, temos GAP com valores negativos. Observamos também que as heurísticas ADN1 tiveram a maior variação no GAP e gastou menos tempo computacional, já que explorou as vizinhanças, em cada fase, apenas uma vez. Os ADN3 obtiveram as menores variações de GAPs e utilizou maior tempo computacional, na maioria. Observamos que as heurísticas ADN que utilizam o K (k-means) possuem menos variações de GAPs em comparação com o que utilizam KB (k-means balanceado).

Para comparar os dois tipos de heurísticas (FO e ADN), na Figura 2 temos o *Boxplot* do GAP de todas as heurísticas para instâncias com 70, 80 e 90 itens. Podemos observar que os GAPs das heurísticas FO são bem próximos entre si e tiveram GAPs menores em comparação com as heurísticas ADN. Mas, por outro lado, observando os valores do tempo médio gasto pelas heurísticas na Tabela 2 e na Tabela 3, as heurísticas ADN gastaram menos tempo, em média, quando comparado a versão clássica FO. Isso indica que o número de iterações pode ser aumentado, gerando melhorias nas heurísticas ADN.

5. Conclusões e perspectivas futuras

Neste trabalho foi abordado o MPCLSP, utilizando 30 instâncias mais difíceis de Carvalho e Nascimento [2016]. Para a abordagem de solução foram implementadas três versões da heurística FO: FO1, FO2 e FO3; e seis versões da heurística ADN: ADN1K, ADN1KB, ADN2K, ADN2KB, ADN3K, ADN3KB. Observamos dentre as heurísticas FO, comparando o GAP médio, o FO2 obteve melhores resultados e dentre as versões da heurística ADN, o ADN3K obteve resultados melhores. Porém, comparando os valores do GAP médio, de forma geral observamos as heurísticas FO obtiveram resultados melhores do que as heurísticas ADN, por outro lado, em relação ao tempo médio, as heurísticas ADN gastaram menos tempo computacional.

Como trabalhos futuros, considera-se a implementação de agrupamentos com outros conjuntos de índices para definir outras vizinhanças para a heurística ADN. Desta forma, investigará se é possível obter soluções com qualidade melhores em comparação as heurísticas de particionamento clássicas, como o FO. Outros problemas de planejamento da produção também podem ser considerados para verificar o desempenho do método em outras condições.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — Brasil (CAPES). Apoio do CNPq (Conselho Nacional de Desenvolvi-

Tabela 3: Resultados obtidos pelas heurísticas ADN1K, ADN1KB, ADN2K, ADN2KB, ADN3K, ADN3KB com o tempo limite igual a 900 para instâncias com 70, 80 e 90 itens. O GAP é em relação ao valor da função objetivo obtido pela heurística e o valor da função objetivo obtido pelo *Gurobi*.

ADN	itens	$GAP_{min}(\%)$	$GAP_{max}(\%)$	$GAP_{med}(\%)$	T_{min}(s)	T_{max}(s)	T_{med}(s)
ADN1K	70	0,0300	0,3061	0,1295	322,46	423,67	354,00
	80	0,0502	0,4329	0,2008	351,36	456,80	384,76
	90	**0,0063**	0,2782	0,1083	330,88	430,87	390,80
ADN1KB	70	0,0513	0,1695	0,1077	**315,71**	**322,66**	**318,72**
	80	0,0648	0,4294	0,1924	**331,63**	**343,02**	**337,32**
	90	0,0896	0,4642	0,1986	**317,78**	**334,98**	**327,54**
ADN2K	70	0,0063	**0,0794**	0,0418	630,65	855,04	776,93
	80	-0,0074	0,2386	0,0865	600,19	900,00	765,64
	90	0,0176	0,3164	0,1011	642,19	900,00	767,67
ADN2KB	70	0,0192	0,2735	0,1050	577,84	764,60	691,09
	80	0,0042	0,4821	0,2545	642,77	829,96	738,24
	90	0,0462	0,4016	0,1731	604,89	900,00	715,89
ADN3K	70	-0,0069	0,0837	**0,0305**	541,72	823,54	705,11
	80	**-0,0239**	0,2169	**0,0465**	442,49	856,79	645,24
	90	0,0112	0,0876	**0,0480**	504,11	811,52	643,85
ADN3KB	70	**-0,0012**	0,0893	0,0432	501,88	730,37	616,28
	80	0,0026	**0,1757**	0,0758	512,49	810,98	649,84
	90	0,0118	**0,0776**	0,0547	537,53	763,79	637,15

mento Científico e Tecnológico) e do Centro de Ciências Matemáticas Aplicadas à Indústria (CeMEAI), financiados pela FAPESP (proc. 2013/07375-0).

Referências

Carvalho, D. M. e Nascimento, M. C. (2016). Lagrangian heuristics for the capacitated multi-plant lot sizing problem with multiple periods and items. *Computers & Operations Research*, 71: 137–148.

Furlan, M. M. (2011). *Métodos heurísticos para o problema de dimensionamento de lotes multi-estágio com limitação de capacidade*. PhD thesis, Universidade de São Paulo.

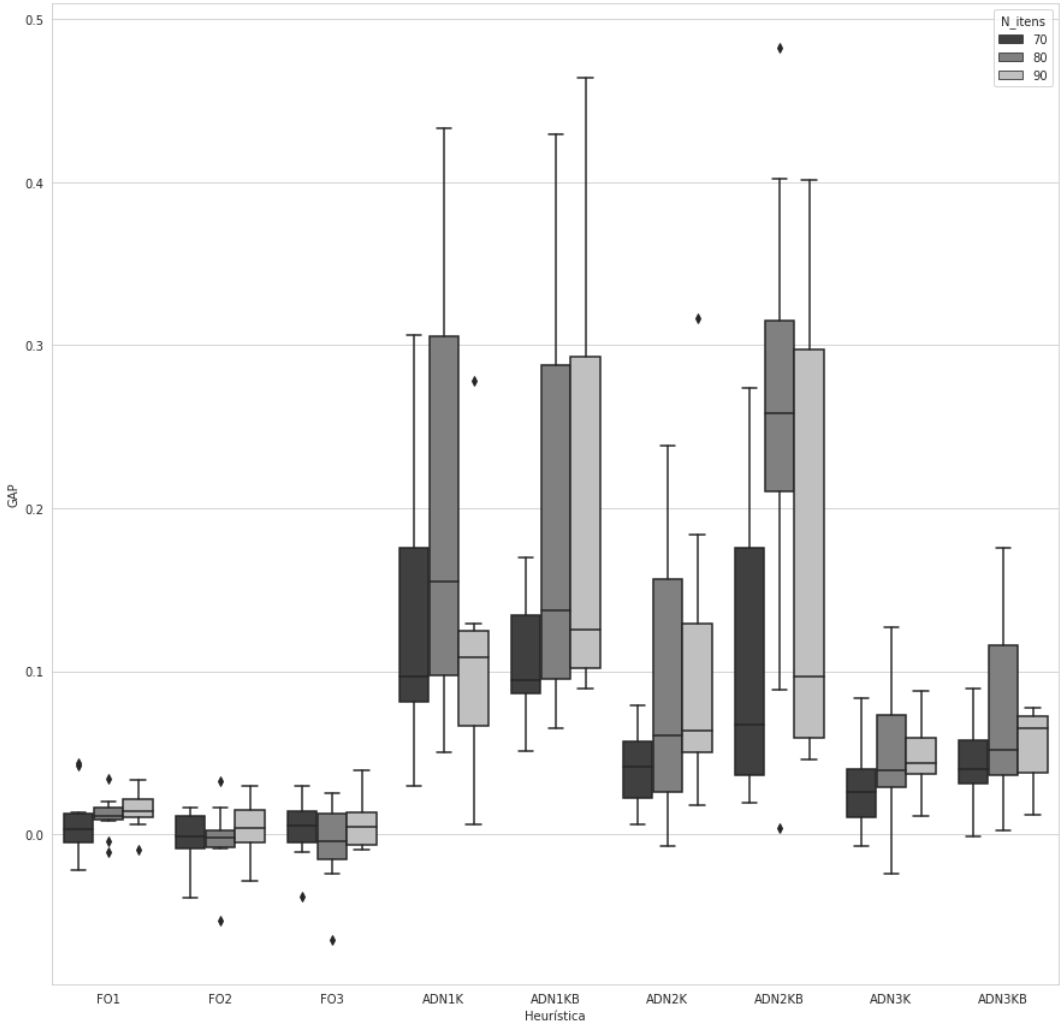
Ghiani, G., Laporte, G., e Manni, E. (2015). Model-based automatic neighborhood design by unsupervised learning. *Computers & Operations Research*, 54:108–116.

Helber, S. e Sahling, F. (2010). A fix-and-optimize approach for the multi-level capacitated lot sizing problem. *International Journal of Production Economics*, 123(2):247–256.

Jain, A. K. e Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice-Hall, Inc.

Melega, G. M., Fiorotto, D. J., e de Araujo, S. A. (2013). Formulações fortes para o problema de dimensionamento de lotes com várias plantas. *TEMA (São Carlos)*, 14(3):305–318.

Figura 2: *Boxplot* do GAP de todas das heurísticas FO e ADN para instâncias com 70, 80 e 90 itens.



Nascimento, M. C., Resende, M. G., e Toledo, F. M. (2010). Grasp heuristic with path-relinking for the multi-plant capacitated lot sizing problem. *European Journal of Operational Research*, 200 (3):747–754.

Pochet, Y. e Wolsey, L. A. (2006). *Production planning by mixed integer programming*. Springer Science & Business Media.

Sambasivan, M. e Schmidt, C. P. (2002). A heuristic procedure for solving multi-plant, multi-item, multi-period capacitated lot-sizing problems. *Asia Pacific Journal of Operational Research*, 19 (1):87–106.

Toledo, C. F. M., da Silva Arantes, M., Hossomi, M. Y. B., França, P. M., e Akartunalı, K. (2015). A relax-and-fix with fix-and-optimize heuristic applied to multi-level lot-sizing problems. *Journal of Heuristics*, 21(5):687–717.