O PROBLEMA DE SEQUENCIAMENTO DE CAMINHÕES EM CENTROS DE *CROSSDOCKING* COM MÚLTIPLAS DOCAS

Priscila Mara Cota
Programa de Pós-Graduação em Engenharia de Produção – UFMG
Av. Antônio Carlos, 6627 – Pampulha, Belo Horizonte – MG – CEP 31270-901
priscila.maracota@gmail.com

Elisa Granha Lira
Departamento de Engenharia de Produção – Universidade Federal de Minas Gerais
Av. Antônio Carlos, 6627 – Pampulha, Belo Horizonte – MG – CEP 31270-901
elisa.granha@gmail.com

Martín Gómez Ravetti
Programa de Pós-Graduação em Engenharia de Produção – UFMG
Av. Antônio Carlos, 6627 – Pampulha, Belo Horizonte – MG – CEP 31270-901
martin.ravetti@dep.ufmg.br

RESUMO

Esse trabalho realiza um estudo à respeito do sequenciamento de caminhões em um Centro de *Crossdocking*. Um modelo de programação linear inteira mista indexado no tempo e um algoritmo *Iterated Greedy*, são propostos. O problema é formulado como um problema de sequenciamento do tipo *flowshop* híbrido de dois estágios, sujeito à restrições de *crossdocking*. Essas restrições proíbem os *jobs* do segundo estágio de iniciar seu processamento antes da conclusão de seus *jobs* precedentes. A metaheurística parte de uma solução viável obtida através de uma heurística construtiva polinomial, a partir da qual, buscas locais são propostas a fim de pesquisar o espaço de soluções do problema. Experimentos computacionais foram realizados comparando o modelo, resolvido por um *solver* comercial, ao seu relaxamento linear e à metaheurística. O modelo foi incapaz de resolver grandes instâncias, no entanto, mostrou bons limites inferiores. A metaheurística encontrou soluções de boa qualidade em um tempo computacional razoável.

PALAVRAS CHAVE. Programação Linear Inteira, *Crossdocking*, *Iterated Greedy*.

ABSTRACT

This article undertakes a study of trucks scheduling in a crossdocking station problem. A time-indexed mixed integer linear programming model and an iterated greedy algorithm are proposed. The problem is formulated as a two-stage hybrid flowshop problem, subject to crossdocking constraints. These constraints forbid a job in the second stage to be processed until the conclusion of its precedent jobs. The metaheuristic starts from a feasible solution obtained through a constructive heuristic, from which, local searches are used to search the solution space of the problem. Several computational experiments are performed comparing the model solved through a commercial solver, its linear relaxation and the metaheuristic. The model was incapable to solve large instances, however, it showed good lower bounds. The metaheuristic found good quality solutions in a reasonable computational times.

KEYWORDS. Integer linear programming model, Crossdocking, Iterated Greedy.

1. Introdução

O ambiente competitivo do mercado global tem aumentado a pressão sobre os fabricantes e distribuidores a fim de fornecer rapidamente produto para os clientes. Para este fim, uma rede de distribuição bem definida é muito importante.

Crossdocking é uma abordagem que elimina ou reduz significativamente duas funções dispendiosas dos centros tradicionais de distribuição que são a estocagem e coleta dos produtos. Para isso, um Centro de *Crossdocking* (CCD) funciona com estoque limitado ou, se possível, nulo. Em um CCD a carga recebida por diversos fornecedores é imediatamente preparada para ser transferida para a área de despacho e destinada aos clientes. Os CCD operam recebendo carretas completas de diversos pontos de fornecimento. Dentro do centro, as cargas são descarregadas, separadas, combinadas e recarregadas em carretas de saída, de acordo com os pedidos específicos dos clientes. Essas carretas então deixam a instalação com carga combinada, composta por produtos de diversos fornecedores, dedicada a um cliente ou destino específico.

Esse artigo propõe um modelo de programação linear inteira indexado no tempo, essa formulação é capaz de fornecer melhores limites de Relaxação Linear se comparado com outros tipos de formulações encontradas na literatura. Além de propor também uma metaheurística que parte de uma solução viável obtida através de uma heurística construtiva polinomial propostas por Araújo e Melo (2010). Aplicam-se buscas Locais a partir dessa heurística de forma a analisar o espaço de busca do problema para então propor a metaheurística baseada no *Iterated Greedy Algorithm* de Ruiz e Stutzle (2007). O modelo e a metaheurística são testados computacionalmente e seus resultados avaliados.

2. Fundamentação Teórica de CCD

Existem diferentes abordagens à respeito dos CCD com enfoques nas diversas etapas do processo logístico. Boysen e Fliedner (2010) e Belle et al. (2012) focam na revisão dos trabalhos presentes na literatura sobre o tema *crossdocking*. Segundo os autores, centros de *crossdocking*, assim como unidades de produção, lidam com problemas envolvendo tomada de decisão, desde a concepção até o nível operacional. Boysen e Fliedner (2010) definem os seguintes problemas enfrentados durante o ciclo de vida de um CCD do nível estratégico ao operacional: localização do Centro de *Crossdocking*, *layout*, atribuição de docas, roteamento de veículos, sequenciamento de caminhões, sequenciamento interno de recursos e empacotamento e desempacotamento de cargas. O presente trabalho possui foco principal no sequenciamento dos caminhões tanto de entrada quanto de saída.

Alguns trabalhos científicos cujo foco se encontra em problemas relacionados ao sequenciamento de caminhões em CCD são apresentados na Tabela 1.

Chen e Lee (2009) abordam o problema do sequenciamento dos caminhões de forma análoga ao problema clássico de *flowshop* com duas máquinas ($F2|CD|C_{max}$), cujo objetivo é minimizar o *makespan*. Esse problema considera as restrições de *crossdocking*, isto é, o carregamento de um caminhão só pode ser iniciado se todos os produtos necessários para aquele caminhão tiverem sido descarregados na doca de entrada. Existe apenas uma doca de recebimento e uma doca de despacho, a primeira máquina representa a doca de entrada, a qual realiza a operação de descarregamento dos caminhões; e a segunda máquina representa a doca de saída, aonde são realizadas as operações de montagem de caminhões contendo produtos com um mesmo destino. Os caminhões são representados por dois conjuntos distintos de *jobs*, os quais estão relacionados a cada uma das duas máquinas. Os autores comprovam que se trata de um problema NP-difícil e apresentam

Tabela 1. Histórico de pesquisas relacionadas ao sequenciamento de caminhões em CCD. Fonte: Boysen e Fliedner (2010).

Publicação	Notação do Problema	Complexidade	Contribuição
McWilliams et al. (2005)	$[E\|t_{io}\|C_{max}]$	Desconhecida	HM, S
Boysen (2010)	$[E\|p_j = p, no - wait, t_j = 0\|\sum T_o]$	Desconhecida	M, HM, E
Boysen et al.	$[E2\|p_j = p, change\|C_{max}]$	NP-difícil	M, HS, HI, E, P
Chen e Lee (2009)	$[E2\|t_j = 0\|C_{max}]$	NP-difícil	B, E, P
Chen e Lee (2009)	$[E2\|t_j = 0, p_j = p\|C_{max}]$	NP-difícil	P
Chmielewski (2007)	$[EM\|r_j, d_j, limit, doors, t_io\|*]$	Desconhecida	M, E
Miao et al. (2009)	$[M\|limit, t_{io}\|*]$	NP-difícil	M, HM, P
Boysen (2008)	$[M1\|p_j = p, t_j = 0, change\|\sum S_p]$	NP-difícil	M, HM, E, P
Yu e Egbelu (2008)	$[E2\|change\|C_max]$	Desconhecida	M, HS
Chen e Song (2009)	$[E\|t_j = 0\|C_{max}]$	NP-difícil	M, HS, B
Boysen e Fliedner (2010)	$[E\|t_{io}, fix\|\sum w_s U_s]$	NP-difícil	M, P
Boysen e Fliedner (2010)	$[E\|t_i = 0, fix\|\sum w_s U_s]$	NP-difícil	P

As siglas da coluna "Contribuição"correspondem à: M (Modelo Matemático),B (Programação por *bound*), HI (Procedimento de Melhoria da Heurística), HS (Começa Heurística para solução inicial), HM (Metaheurística), P (Propriedades e.g. complexidade do problema), S (Abordagem por Simulação) e E (Procedimento de resolução exato)

casos especiais em que ele pode ser resolvido polinomialmente. Ao final propõe um algoritmo de *branch and bound* para resolução do problema. Os resultados mostram que o problema pode resolver instâncias com até 60 caminhões em tempo aceitável.

Chen e Song (2009) estendem esse trabalho considerando múltiplas docas de entrada e saída, esse problema é denominando *crossdocking* híbrido de dois estágio, ou seja, com várias máquinas em cada estágio, paralelas e idênticas, uma para cada porta do CCD $(F2(P)|CD|C_{max})$, que é o problema considerado neste artigo. Os autores supracitados propuseram uma modelagem matemática resolvida por programação inteira mista capaz de resolver pequenas instâncias. Além disso, quatro heurísticas construtivas são investigadas e comparadas para analisar sua performance em instâncias moderadas e grandes. Posteriormente com base nesses dois trabalhos científicos, Araújo e Melo (2010) analisam e avaliam heurísticas já existentes na literatura para o problema em questão e propõe outras, objetivando determinar aquelas que obtêm melhores *GAPs* quando comparadas a um limite inferior. A heurística construtiva polinominal que apresentou melhores resultados será utilizada na metaheurística aqui proposta. Essa heurística construtiva viável escolhida (chamada nesse trabalho de HP1) é embasada na heurística NEH, a qual foi proposta por Nawaz et al. (1983), é uma das melhores heurísticas polinomiais para o *flowshop* permutacional. A heurística HP1 ordena os *jobs* do segundo estágio conforme a regra do maior tempo de processsamento *(largest processing time - LPT)*

IG está fortemente relacionado com o *Iterated Local Search (ILS)*, mas ao invés de realizar as iterações em relação a uma busca local como em ILS, IG realiza suas iterações de forma análoga só que usando heurísticas de construção. Ruiz e Stutzle (2007) foram os primeiros a aplicar algoritmo *iterated greedy* com sucesso para um problema de sequenciamento.

Modelos indexados no tempo foram propostos em Sousa e Wolsey (1992) no problema *non-preemptive single machine scheduling*. Nos modelos indexados no tempo, o horizonte de tempo é dividido em períodos e, assim as variáveis de decisão são também indexadas em cada um dos períodos empregados. Esta abordagem leva a modelos grandes normalmente com um número maior de restrições e variáveis de decisão, entretanto, suas

relaxações lineares fornecem limitantes duais muito mais fortes que outras formulações encontradas na literatura.

3. Descrição do problema

O problema será tratado neste estudo analogamente a um sequenciamento comum em linhas de produção de modo que docas recebem e descarregam ou carregam caminhões assim como máquinas processam ordens de produção (*jobs*). Assim o problema consiste na determinação da sequência na qual caminhões são descarregados nas docas de entrada e carregados nas docas de saída, de forma a minimizar o tempo total de conclusão das tarefas (*makespan*).

Esse problema é representado por $F2(P)|CD|C_{max}$. Um problema análogo ao *flowshop* híbrido de dois estágios ("F2 ") com multiprocessadores idênticos em paralelo ("P"), com restrições de *crossdocking* (CD), no qual a função objetivo é minimizar o *makespan* (C_{max}). Segundo Garey e Johnson (1979) esse problema é NP-difícil no sentido estrito, o que justifica a apresentação de uma nova formulação matemática com a finalidade de encontrar limites mais estreitos e o uso de abordagens metaheurísticas para resolução de grandes instâncias.

3.1. Formulação

Para facilitar o entendimento do modelo e da metaheurística, a notação utilizada é sumarizada abaixo.

- i: estágio de processamento, $i \in I = \{1, 2\}$;
- j: *job* a ser processado, $j \in J^i = \{j_1, ...j_{n_i}\}$;
- J^i: Conjunto de *jobs* a serem processados no estágio i.
- t: instante de processamento, $t \in T$;
- T: Horizonte de tempo considerado, uma primeira estimativa é a soma de processamento de todos os *jobs*;
- m_i: número de processadores (docas) no estágio i.
- n_i: número de *jobs* no estágio i.
- p_{ij}: tempo de processamento do *job* j no estágio i;
- S_j: conjunto de subconjuntos precendentes de J^1 correspondente ao *job* $j \in J^2$, $S_j \in S = \{S_1, ..., S_{n_2}\}$.
- C_{max}: tempo que se demora para processar todos os *jobs* nos dois estágios.
- r_j: data de liberação do *job* j, $j \in J^2$.
- $x_{jt} = 1$ se o *job* j começar no instante t, e igual a 0 caso contrário, $j \in J^1$.
- $y_{jt} = 1$ se o *job* j começar no instante t, e igual a 0 caso contrário, $j \in J^2$.

3.2. Modelo Matemático

A partir da notação e variáveis de decisão acima, segue o Modelo de Programação Linear Inteira Indexado no Tempo:

$$min \quad C_{max} \tag{1}$$

$$sujeito \quad a:$$

$$\sum_{t=0}^{T-p_{1j}} x_{jt} = 1; \quad \forall j \in J^1; \tag{2}$$

$$\sum_{t=0}^{T-p_{2j}} y_{jt} = 1; \quad \forall j \in J^2; \tag{3}$$

$$\sum_{j \in J^1} \sum_{s=max\{0,t-p_{1j}+1\}}^{t} x_{js} \leq m_1; \quad \forall t \in T; \tag{4}$$

$$\sum_{j \in J^2} \sum_{s=max\{0,t-p_{2j}+1\}}^{t} y_{js} \leq m_2; \quad \forall t \in T; \tag{5}$$

$$r_j \geq \sum_{t=0}^{T} (t + p_{1k}) * x_{kt}; \forall j \in J^2; \quad \forall k \in S_j, \tag{6}$$

$$\sum_{t=0}^{T-p_{2j}} t y_{jt} \geq r_j; \quad \forall j \in J^2; \tag{7}$$

$$C_{max} \geq p_{2j} + \sum_{t=0}^{T-p_{2j}} t y_{jt}; \quad \forall j \in J^2; \tag{8}$$

$$x_{jt} \in 0, 1; \quad \forall j \in J^1; \quad \forall t \in T, \tag{9}$$

$$y_{jt} \in 0, 1; \quad \forall j \in J^2; \quad \forall t \in T, \tag{10}$$

$$r_j \geq 0; \quad \forall j \in J^2; \tag{11}$$

$$C_{max} \geq 0; \tag{12}$$

O conjunto de restrições (2) garantem que cada *job* $j \in J^1$ deve iniciar seu processamento em uma, e somente uma, data dentro do horizonte de planejamento T, não permitindo que se associem duas datas distintas para o início do processamento de um *job*. O conjunto (3) trabalha com o mesmo raciocínio, porém aplicado aos *jobs* processados nas máquinas do segundo estágio, $j \in J^2$. As restrições indicadas por (4) garantem que um *job* $j \in J^1$ não inicia seu processamento enquanto outro estiver sendo processado, da seguinte forma: para cada data $t \in T$, a restrição verifica se algum *job* $j \in J^1$ começou a ser processado em uma data maior ou igual a $t - p_{1j} + 1$. Caso afirmativo, j ainda está sendo processado, o número de *jobs* que podem ser processados ao mesmo tempo é menor ou igual ao número de máquinas existente naquele estágio. O conjunto de restrições (5) trabalha como o conjunto anterior, no entanto é aplicado aos *jobs* $j \in J^2$. O conjunto (6) trata as precedências dos *jobs*, são assim, as restrições do tipo *crossdocking*. Para cada restrição de precedência, cria-se uma restrição desse conjunto, na qual, a data de liberação de cada *job* do segundo estágio, deve ser maior ou igual a data de conclusão da tarefa precedente. Se o *job* possui mais de um precedente, prevalece aquela relativa ao *job* precedente com maior data de conclusão. Já o conjunto (7) garante que cada *job* do segundo estágio só inicia seu processamento depois da sua data de liberação. O conjunto de restrições (8) indicam que a variável C_{max}, *makespan*, deve ser a maior data de conclusão dos *jobs* do segundo estágio. Por fim os conjuntos (9) à (12) definem o domínio das variáveis do modelo. A função objetivo é dada por (1) e visa minimizar o tempo necessário para o processamento de todos os *jobs (makespan)*.

4. Metaheurística

Para a construção da metaheurística tornou-se necessário definir primeiramente a heurística construtiva polinomial viável a ser utilizada. Utilizamos a que apresentou o melhor resultado no trabalho de Araújo e Melo (2010), a HP1. Posteriormente definimos

a busca local a ser utilizada. E por fim determinamos os parâmetros necessários para a utilização do algoritmo IG.

4.1. Espaços de busca

Foram definidas inicialmente duas estruturas de vizinhança (Movimento de Troca Simples de dois *jobs* e Movimento de Inserção) para serem testadas e comparadas para as estratégias de busca *First Improvement* (FI) e *Best Improvement* (BI), com o objetivo de testar qual a busca local proporciona a maior melhoria no *makespan*. A melhoria é calculada da seguinte forma:

$$Melhoria = \frac{Cmax(AntesBuscaLocal) - Cmax(AposBuscaLocal)}{Cmax(AntesBuscaLocal)} * 100$$

A fim de identificar a melhor estratégia os algoritmos desses quatro tipos de busca local foram implementados de forma a analisar os resultados. Após o estudo estatístico e comparativo a busca local de troca simples de dois *jobs* apresentou melhores resultados que o movimento de inserção para as duas estratégias de busca. Assim, optou-se por utilizar a estrutura de vizinhança de troca simples na metaheurística a ser criada. Comparando a busca local de troca FI com BI, a segunda apresentou melhor resultado. Entretanto, como a BI apresenta um tempo computacional muito maior, optou-se por utilizar a busca local de troca simples *jobs First Improvement* na metaheurística.

4.2. Representação e estrutura de vizinhança: Movimento de troca simples (2, 3, 4, 5 e 6 *jobs*)

Definida a busca local de troca simples *First Improvement*, investigou-se a quantidade ideal de *jobs* a serem trocados durante essa busca local. Assim, a troca simples de 2, 3, 4, 5 e 6 *jobs* foi testada tanto para a distribuição de *jobs* nas máquinas do primeiro estágio quanto para a distribuição de *jobs* nas máquinas do segundo estágio.

Manteve-se o mesmo pseudocódigo de troca simples *First Improvement* da seção anterior com a diferença de que nos casos em que se trocou mais de 2 *jobs*, os outros *jobs* a serem trocados eram sorteados aleatoriamente. As trocas seguiam a ordem de sorteio, e cada *job* sorteado assume a posição do *job* sorteado em seguida. Assim, o último *job* sorteado assumirá a posição do primeiro.

Em relação ao processamento no primeiro estágio, a busca local de troca simples que apresentou melhores resultados na análise geral foi a de 3 *jobs*, embora não exista diferença significativa entre as buscas de trocas testadas. Em relação ao segundo estágio, a busca local FI de troca simples de 5 *jobs* foi escolhida arbitrariamente também para ser usada no algoritmo *Iterated Greedy*, uma vez que as buscas locais apresentaram tempos computacionais semelhantes e essa busca local apresentou melhor desempenho.

4.3. Algoritmo *Iterated Greedy* proposto

A metaheurística proposta, chamada MH1, baseia-se no algoritmo *iterated greedy*. O algoritmo parte de uma solução inicial viável, gera uma sequência de soluções através da iteração contínua das fases de *Construção* e *Destruição*. Em seguida, aplica-se a Busca local troca simples de 3 *jobs First Improvement* no primeiro estágio e a Busca local troca simples de 5 *jobs First Improvement* no segundo estágio.

A fase de *Destruição* remove aleatoriamente *d jobs*, os quais devem ser diferentes entre si, a partir de uma permutação π de *n jobs*. Assim, gera-se a partir da solução inicial π dois conjuntos:

- π_D que representa a solução inicial após a remoção dos *d jobs* e que possui, pois, $n - d$ *jobs*.
- π_R que representa o vetor de *d jobs* sorteados aleatoriamente e eliminados da π, os quais serão reinseridos na etapa de *Construção*.

A fase de *Construção* testa a inserção iterativa em *d* passos dos *jobs* de π_R em todas as posições possíveis de π_D de forma a alocá-los na permutação que produza um menor *makespan*. Assim, $\pi_R^{[1]}$ é o primeiro *job* a ser alocado em π_D na posição que produza um menor *makespan*. π_D é então aumentado em um *job* e o processo se repete até que todos os *jobs* sejam alocados e π_R fique vazio. Após a fase de *Construção* aplicou-se a busca local.

Quando uma nova solução candidata é criada, um critério de aceitação decide se a nova solução deve substituir a atual solução armazenada π'. Objetivando evitar situações de estagnação da busca por diversificação insuficiente, considera-se um critério de aceitação semelhante ao do método *simulated annealing* com uma temperatura constante que depende da instância e é dada pela seguinte fórmula:

$$Temperatura = T * (\sum_{i=1}^{n_1} p_{1e_i})/n_1 + (\sum_{i=2}^{n_2} p_{2e_i})/n_2;$$

A melhoria alcançada através do algoritmo proposto é avaliada pela seguinte fórmula:

$$Melhoria = \frac{Cmax(AntesIG) - Cmax(AposIG)}{Cmax(AntesIG)} * 100$$

O pseudocódigo do algoritmo *iterated greedy* utilizado segue na Figura 1.

O critério de parada escolhido foi um número máximo de iterações ≤ 20, pois, a partir desse número de iterações a melhor solução encontrada (π_b) demorava muito tempo para ser atualizada.

4.3.1. Experimentos preliminares: ANOVA

Para execução do algoritmo *iterated greedy*, fez-se necessário o ajuste de dois parâmetros *d* e *T*, para isso, foi utilizada a abordagem de Planejamento de Experimentos, através da técnica de Análise de Variância (ANOVA), conduzido no Minitab. A análise de variância é uma técnica estatística que permite avaliar afirmações sobre as médias de populações, dessa maneira, vamos verificar se existe uma diferença significativa entre a melhoria média do *makespan* e se os fatores exercem influência nessa melhoria. Ao se utilizar a técnica ANOVA deve-se checar suas três principais suposições: normalidade, homocedasticidade e independência dos resíduos. Essas três premissas foram aceitas na ANOVA realizada. Sugerindo que a suposição do erro aleatório ser normalmente distribuído e a variância constante é razoável para os experimentos conduzidos. Dessa forma, foi conduzido um experimento fatorial completo com 2 fatores, os quais possuíam 4 níveis cada.

- Fator 1: *T* - Níveis de teste: 10, 20, 30 e 40.
- Fator 2: *d* - Níveis de teste: 2, 3, 4 e 5.

Para aplicação da técnica (ANOVA), o número de máquinas (*m*) e o número de *jobs* em cada estágio (*n*) foram considerados fatores não controlados. Analisaram-se os três grupos diferentes de números de máquinas: duas, quatro e dez máquinas nos dois estágios. Dentro de cada um desses grupos, foi análisado dois casos, o primeiro com 30 *jobs* no primeiro estágio e segundo com 60 *jobs* no primeiro estágio. Assim, foram consideradas

Procedimento *IteratedGreedy*

1 $\pi \leftarrow$ NEHtp2LPT_HeuristicaConstrutiva;
2 $\pi \leftarrow$ BuscaLocal_TrocaSimples
3 $\pi_b \leftarrow \pi$;
4 Enquanto (*critério de término não satisfeito*) então
5 $\pi' \leftarrow \pi$;
6 Para i = 1 até *d* faça //Fase de Destruição
7 $\pi' \leftarrow$ remove um *job* aleatoriamente de π' e insere em π_b;
8 Fim para
9 Para i = 1 até *d* faça //Fase de Construção
10 $\pi' \leftarrow$ melhor permutação obtida ao se inserir *job* $\pi_b(i)$ em todas as possíveis posições de π';
11 Fim para
12 $\pi'' \leftarrow$ BuscaLocal_X;
13 Se ($f(\pi'') < f(\pi)$) então //Critério de Aceitação 1 (CA1)
14 $\pi \leftarrow \pi''$;
15 Se ($f(\pi) < f(\pi_b)$) então //atualiza melhor solução encontrada
16 $\pi_b \leftarrow \pi$;
17 Fim se
18 Fim se
19 Se($aleatório \leq exp\{-(f(\pi'') - f(\pi))/Temperatura\}$) então //(CA2)
20 $\pi \leftarrow \pi''$;
21 Fim se
22 Fim enquanto
23 Retorne π_b

Fim *IteratedGreedy*

Figura 1. Pseudocódigo do algoritmo *iterated greedy*.

6 combinações distintas de número de máquinas e de *jobs*. Para cada uma dessas combinações, geraram-se a partir de uma mesma instância 10 distribuições aleatórias de *jobs* nas máquinas de cada estágio. O objetivo era testar em cada uma dessas distribuições aleatórias a melhoria que a busca local iria proporcionar no *makespan*.

A tabela ANOVA gerada indicou que, o fator *Destruição (d)* se mostrou estatisticamente significante (p-valor<0.05), considerando um nível de significância de 5%. A figura 2 contém o Gráfico de efeitos principais para o fator *d* e para o fator *T*. Como pode ser observado nesse gráfico, o nível do fator *Destruição* de 3 *jobs* apresentou um melhor desempenho na média para a variável de resposta melhoria. Já o fator *T* não apresentou diferenças significativas entre seus níveis considerando um nível de significância de 5% (p-valor>0.05), como também ocorreu nos experimentos de Ruiz e Stutzle (2007). De acordo com esses autores isso sugere que o algoritmo IG proposto é robusto em relação à *T*. Como resultado os parâmetros foram ajustados em $d = 3$ e $T = 20$.

5. Resultados Computacionais

Os experimentos computacionais foram realizados em um computador DELL Inspiron 15z com memória de 8GB, 500GB de HD, processador Intel Core i7 e sistema operacional Windows 7 Home Premium. Os programas foram escritos na linguagem C.

5.1. Geração de Instâncias

As instâncias foram criadas utilizando o gerador de números pseudo-aleatórios de *Mersenne Twister*. O pseudocódigo para a geração segue abaixo:

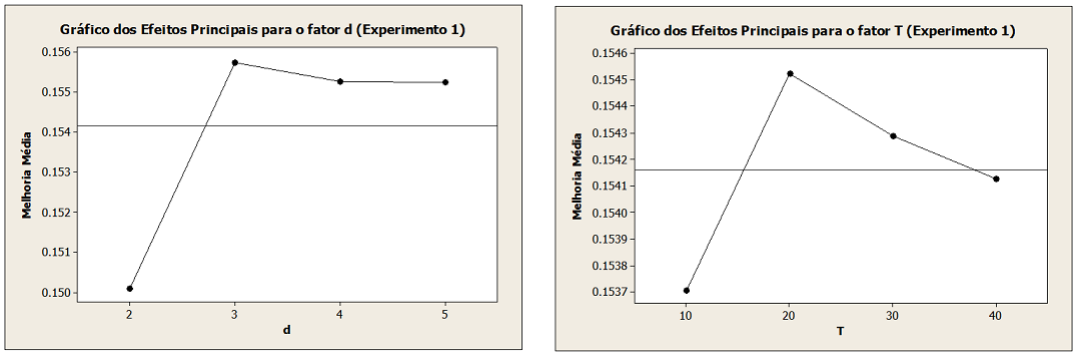

Figura 2. Gráficos dos Efeitos Principais para o fator *d* e *T*.

1. O número de *jobs* no segundo estágio é gerado por uma distribuição uniforme entre 80% e 120% em relação ao número de *jobs* no primeiro estágio: $n_2 = U[0,8n_1, 1, 2n_1]$.
2. O número de máquinas é igual para os dois estágios: $m_1 = m_2$.
3. O tempo de processamento dos *jobs* no primeiro e segundo estágio são gerados seguindo uma distribuição uniforme.
4. A matriz de precedências do tipo binária (0 ou 1) é gerada. Essas matriz possui dimensão n_2 x n_1. Se o valor for igual a 1, significa que o *job* do estágio 1 que se refere a essa coluna é precedente do *job* do segundo estágio que se refere a essa linha. Uma premissa considerada foi que os *jobs* do segundo estágio deveriam ter pelo menos um precedente.

5.2. Pequenas Instâncias

Três grupos foram criados com diferentes números de máquinas em cada estágio: Duas, quatro e seis máquinas nos dois estágios. Dentro de cada um desses grupos, variou-se o número de *jobs* no primeiro estágio ($n_1 = 5, 8, 10, 15$). Assim, foram consideradas 12 combinações distintas de número de máquinas e de *jobs*. Para cada uma dessas combinações 3 instâncias foram consideradas. Com a finalidade de análisar a influência do tempo de processamento dos *jobs* no modelo, dois grupos foram considerados:

- Grupo 1: Tempo de processamento dos *jobs* variando entre 1 e 10;
- Grupo 2: Tempo de processamento dos *jobs* variando entre 10 e 50;

Os resultados alcançados são analisados na Tabela 2. A qual apresenta o valor da função objetivo FO (*makespan*), e o tempo computacional para o modelo completo (MC). Já para a relaxação linear do modelo (RL) e para a metaheurística proposta MH1, são apresentados os valores encontrados para as funções objetivo e o *Gap* com relação ao modelo completo. Na RL as restrições de integralidade das variáveis x_{jt} e y_{jt} não são consideradas. O critério de parada específicado foi o tempo computacional viável ($t \leq 1800$ segundos), assim, alguns valores da tabela não foram encontrados, isso devido à complexidade do problema. O símbolo (*) presente na tabela indica que o valor da função objetivo não foi encontrado por MH1, isso porque o algoritmo IG criado só é resolvido para instâncias mais realisticas, onde o número de *jobs* é maior que o número de máquinas.

A partir da Tabela 2 é possível verificar que à medida que o número de *jobs* aumenta o tempo computacional para a resolução do modelo completo aumenta consideravelmente. É importante ressaltar que foram criadas instâncias com 15 *jobs* no primeiro estágio, que só foram resolvidas pelo MC quando são consideradas 6 máquinas. Além disso, é perceptível que a medida que se aumenta o número de máquinas (docas) o MC passa a ser resolvido mais rápidamente. Com relação ao modelo relaxado todas as instâncias foram resolvidas

quase instantaneamente ($t \leq 1$ s). À medida que o número de máquinas aumenta a RL encontra respostas muito próximas da ótima, encontrada pelo MC. Para os teste com seis máquina quase todos os valores ótimos foram encontrados, o que demonstra a força do modelo de programação linear indexado no tempo proposto. A metaheurística MH1, assim como a RL é resolvida para todas as instâncias em tempo inferior a 1 segundo, e apresenta resultados próximos aos encontrados pelo MC, uma vez que a medida que a instância cresce o resultado melhora. É importante ressaltar que os *Gaps* calculados para MH1 são menores que os calculados para a RL, o que demonstra que o algoritmo IG proposto encontra bons resultados.

Analisando o Grupo 2, é perceptível a maior dificuldade do MC para resolver as intâncias. Isso é devido ao número de variáveis que aumenta consideravelmente, uma vez que, leva em consideração o valor inicial de T (soma dos tempos de processamento). É importante dizer, que os parâmetros em relação aos dois grupos são os mesmos, exceto o tempo de processamento. Apesar da maior dificuldade encontrada pelo MC, a RL e MH1 continuam sendo resolvidos a medida que as instâncias crescem. Sendo que bons resultados continuam sendo alcançados principalmente por MH1.

Tabela 2. Resultados de *makespan* para o Modelo Indexado no tempo Completo, para a relaxação linear do modelo e para MH1

		Grupo 1: p[1;10]						Grupo 2: p[10;50]					
		M. Completo		M. Relaxado		MH1		M. Completo		M. Relaxado		MH1	
m_1	n_1	FO	t(s)	FO	Gap1(%)	FO	Gap2(%)	FO	t(s)	FO	Gap1(%)	FO	Gap2(%)
	5	23	0	17	26	23	0	130	4	103	21	144	11
	5	23	0	20	13	23	0	140	0	111	21	148	6
	5	20	0	15	25	22	10	138	3	93	33	138	0
	8	25	1	19	24	27	8	160	4223	102	36	160	0
	8	36	12	25	31	38	6	-	-	119	-	210	-
2	8	33	4	23	30	35	6	-	-	119	-	177	-
	10	36	158	25	31	38	6	-	-	131	-	267	-
	10	35	6	26	26	38	9	-	-	130	-	220	-
	10	43	570	27	37	43	0	-	-	113	-	187	-
	15	-	-	31	-	60	-	-	-	185	-	354	-
	15	-	-	36	-	72	-	-	-	182	-	408	-
	15	-	-	35	-	83	-	-	-	168	-	420	-
	5	16	0	15	6	16	0	91	0	91	0	91	0
	5	17	1	17	0	*	-	108	1	91	16	*	-
	5	14	0	14	0	14	0	95	1	85	11	100	5
	8	16	0	15	6	17	6	102	7	81	21	103	1
	8	21	1	20	5	22	5	119	1503	99	17	123	3
4	8	20	0	19	5	20	0	105	8	89	15	106	1
	10	22	0	19	14	23	5	-	-	97	-	156	-
	10	22	0	19	14	23	5	-	-	98	-	128	-
	10	26	1	20	23	27	4	-	-	82	-	110	-
	15	-	-	22	-	35	-	-	-	123	-	195	-
	15	-	-	25	-	42	-	-	-	124	-	223	-
	15	-	-	24	-	46	-	-	-	115	-	217	-
	5	15	0	15	0	*	-	91	0	91	0	*	-
	5	17	0	17	0	*	-	91	0	91	0	*	-
	5	14	0	14	0	*	-	85	0	85	0	*	-
	8	15	0	15	0	15	0	81	1	81	0	81	0
6	8	20	0	20	0	20	0	99	1	99	0	99	0
	8	19	0	19	0	19	0	89	1	89	0	89	0
	10	19	0	19	0	19	0	109	121	93	15	116	6
	10	18	0	18	0	18	0	98	4	95	3	100	2
	10	20	0	20	0	20	0	80	0	80	0	88	10
	15	24	29	22	8	26	8	-	-	103	-	143	-
	15	29	87	25	14	31	7	-	-	103	-	159	-
	15	30	972	24	20	33	10	-	-	98	-	157	-

5.3. Médias e Grandes Instâncias

Três grandes grupos foram criados com diferentes números de máquinas: Duas, quatro e dez máquinas nos dois estágios. Dentro de cada um desses grupos, variou-se o número de *jobs* no primeiro estágio (n_1), em um intervalo de 20 em 20, entre os valores de 20 e 80, com tempos de processamento entre 10 e 100. Foram consideradas 12 combinações distintas de número de máquinas e de *jobs*. Para cada uma dessas combinações, geraram-se 20 instâncias, totalizando 240 instâncias. Os resultados alcançados são analisados na Tabela 3.

Tabela 3. Resultados da melhoria no *makespan* após aplicação do *iterated greedy* com critério de parada: # iterações ≤ 20

n_1	n_2	Análise	2maqs	4maqs	10maqs	Média total
20	U[16 24]	Melhoria Média	28.04%	16.49%	15.76%	
		Desvio Padrão	13.58%	8.97%	4.96%	
		IC Melhoria ±	9.71%	6.42%	3.55%	
40	U[32 48]	Melhoria Média	15.04%	10.75%	9.28%	
		Desvio Padrão	3.87%	5.97%	2.27%	
		IC Melhoria ±	2.77%	4.27%	1.62%	
60	U[48 72]	Melhoria Média	14.59%	9.31%	7.60%	
		Desvio Padrão	5.22%	5.09%	3.24%	
		IC Melhoria ±	3.73%	3.64%	2.32%	
80	U[64 96]	Melhoria Média	12.73%	9.52%	6.12%	
		Desvio Padrão	5.14%	6.69%	2.80%	
		IC Melhoria ±	3.68%	4.78%	2.01%	
ANÁLISE	GERAL	Melhoria Média	17.60%	11.52%	9.69%	**12.94%**
		Desvio Padrão	6.95%	6.68%	3.32%	**5.65%**
		IC Melhoria ±	4.97%	4.78%	2.38%	**4.04%**

A partir da Tabela 3 observa-se que o algoritmo proposto causou melhorias importantes no cálculo do *makespan* para instâncias grandes, não resolvidas pelo modelo exato. Como pode ser observado, MH1 fornece uma melhoria média, considerando os três cenários, de 12.94% no *makespan* quando comparado com o *makespan* alcançado pela heurística construtiva polinomial. Para o caso de 2 máquinas em cada estágio e 20 *jobs* no estágio 1, MH1 apresentou uma melhoria média de 28.04% com relação ao algoritmo que aplica apenas a HP1. Outro ponto importante é a rapidez com que os testes são realizados, todos os teste foram resolvidos em tempo computacional viável. Dessa forma, a metaheurística IG criada causou melhorias significativas para as instâncias grandes, e resultados próximos ao ótimo para instâncias menores que foram comparadas ao modelo exato e a relaxação linear do modelo.

6. Conclusões

Neste trabalho, apresentamos um novo modelo de Programação Matemática e uma metaheurística para o Problema de sequenciamento de caminhões em um CCD. O modelo proposto, por empregar indexação no tempo, produz limites de Relaxação Linear fortes, que é resolvido rapidamente e que muitas vezes encontra a solução ótima do problema exato. A metaheurística testada para instâncias pequenas apresentou bons resultados, à medida que o número de *jobs* aumenta ela passa a encontrar resultados mais próximos do ótimo. A melhoria causado no valor do *makespan* para instâncias grandes a partir da metaheurística foi muito significativa como apresenta a Tabela 3.

Como propostas de trabalhos futuros a serem desenvolvidos pretendemos investigar o uso de técnicas como Relaxação Lagrangeana para acelerar a resolução da Relaxação

Linear do modelo apresentado. Ao assim proceder, esperamos torná-lo mais competitivo e capaz de resolver instâncias de maiores dimensões. Já com relação à metaheurística aqui proposta, o algoritmo *iterated greedy* apresentou bons resultados em relação à melhoria gerada no *makespan* após sua aplicação. A simplicidade desse algoritmo também é outro ponto positivo. Entretanto, seria interessante comparar o algoritmo IG com outras metaheurísticas como, por exemplo, *iterated local search*, *simulated annealing*, algoritmos genéticos, entre outras.

7. Agradecimentos

À FAPEMIG (Fundação de Amparo à Pesquisa de Minas Gerais) e ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) pelo apoio financeiro recebido.

Referências

Araújo, D. P. M. e Melo, B. M. R. (2010). Heuristicas construtivas para o sequenciamento de caminhoes em centros de cross docking. Master's thesis, Universidade Federal de Minas Gerais.

Belle, J. V., Valckenaers, P., e Cattrysse, D. (2012). Cross-docking: State of the art. *Omega: The International Journal of Management Science*, 40:827–846.

Boysen, N. (2010). Truck scheduling at zero-inventory cross docking terminals. *Computers and Operations Research,*, 37:32–41.

Boysen, N. e Fliedner, M. (2010). Cross dock scheduling: Classification, literature review and research agenda. *Omega: The International Journal of Management Science*, 38:413–422.

Chen, F. e Lee, C. Y. (2009). Minimizing the makespan in a two-machine cross-docking flow shop problem. *European Journal of Operational Research 2009.*, 193:59–72.

Chen, F. e Song, K. (2009). Minimizing makespan in two-stage hybrid cross docking scheduling problem. *Computers & Operations Research*, 36:2066–2073.

Garey, M. R. e Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co. New York.

McWilliams, D. L., Stanfield, P. M., e Geiger, C. D. (2005). The parcel hub scheduling problem: A simulation-based solution approach. *Computers and Industrial Engineering*, 49:393–412.

Miao, Z., Lim, A., e Ma, H. (2009). Truck dock assignment problem with operational time constraint within crossdocks. *European Journal of Operational Research* ., 192:105–115.

Nawaz, M., Jr., E. E. E., e Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega: The International Journal of Management Science, 11*, 11:91–95.

Ruiz, R. e Stutzle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177:2033–2049.

Sousa, J. P. e Wolsey, L. A. (1992). A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical Programming*, 54:353–367.

Yu, W. e Egbelu, P. J. (2008). Scheduling of inbound and outbound trucks in cross docking systems with temporary storage. *European Journal of Operational Research*, 184:377–396.