UM ESTUDO SOBRE FORMULAÇÕES DE PROGRAMAÇÃO INTEIRA MISTA PARA O PROBLEMA DE *JOB SHOP* FLEXÍVEL

Ernesto G. Birgin[1], **Paulo Feofiloff**[1], **Cristina G. Fernandes**[1],
Everton L. de Melo[2], **Marcio T. I. Oshiro**[1], **Débora P. Ronconi**[2]

[1] Universidade de São Paulo – Instituto de Matemática e Estatística
Rua do Matão, 1010, Cidade Universitária – São Paulo-SP – Brasil
egbirgin@ime.usp.br, pf@ime.usp.br, cris@ime.usp.br, oshiro@ime.usp.br

[2] Universidade de São Paulo – Escola Politécnica
Av. Almeida Prado, 128, Cidade Universitária – São Paulo-SP – Brasil
everton.melo@usp.br, dronconi@usp.br

RESUMO

O ambiente de produção abordado neste trabalho é o *Job Shop* Flexível (JSF), uma generalização do *Job Shop* (JS) que é um problema NP-difícil. O JS é composto por um conjunto independente de tarefas, cada qual constituída por uma sequência ordenada de operações. Cada operação deve ser processada individualmente em uma única máquina, enquanto que no JSF cada operação possui um subconjunto de máquinas capazes de processá-la. A medida de desempenho considerada será a minimização do instante de término da última tarefa (*makespan*). São apresentadas duas formulações de programação inteira mista para o JSF. Elas são comparadas com uma formulação disponível na literatura, em diversos problemas através de um software de programação matemática. Os resultados mostram que as formulações propostas apresentam, em geral, melhor desempenho.

PALAVRAS CHAVE. programação de tarefas, *job shop*, formulações inteiras mistas, *makespan*.

Área principal: Programação Matemática.

ABSTRACT

The production environment addressed in this work is a generalization of the NP-hard Job Shop (JS) problem known as Flexible Job Shop (FJS). In the machine scheduling JS problem, one is given a set of independent jobs, each consisting of a sequence of operations. Each operation must be processed individually by a specific machine, while in the FJS each operation can be processed by a subset of the machines. The performance measure considered is the minimization of the maximum completion time of the jobs (makespan). This work proposes two mixed integer linear programming formulations for the FJS. These formulations are compared against a formulation available in the literature in several problems using a mathematical programming software. The results indicate that the proposed formulations achieve, in general, best performance.

KEYWORDS. machine scheduling, job shop, mixed integer formulations, makespan.

Main area: Mathematical Programming.

1. Introdução

O problema de programação de tarefas *Job Shop* (JS), classificado por Garey *et al.* (1976) como NP-difícil, pode ser enunciado como segue. Considere um conjunto de máquinas e um conjunto independente de tarefas, cada qual constituída por uma sequência ordenada de operações. Cada operação deve ser processada individualmente em uma máquina específica do conjunto sem interrupção. As máquinas nas quais duas operações de uma mesma tarefa devem ser processadas são diferentes. Deve ser obtida a sequência de processamentos de modo que o instante de término da última operação processada da última tarefa finalizada, o *makespan*, seja o menor possível. Além do *makespan*, outros critérios de otimização também podem ser considerados, como atraso total (*tardiness*), *lateness* ou a carga de trabalho nas máquinas.

Um modo de representar o JS é através de um grafo, chamado grafo disjuntivo, como proposto por Balas (1969). Nesse grafo, cada operação de cada tarefa é um vértice cujo peso é o tempo de processamento da respectiva operação. Além desses, há mais dois vértices, para representar o início e o término das operações. Arcos ligam o vértice de início à primeira operação de cada tarefa e também a operação final de cada tarefa ao vértice de término. Também há arcos que ligam as operações consecutivas de cada tarefa, determinando a ordem em que elas devem ser processadas. Finalmente são acrescentados arcos sem orientação, chamados arcos disjuntivos, que conectam operações a serem executadas numa mesma máquina. A Figura 1 ilustra o grafo disjuntivo de um JS com 3 tarefas e 4 máquinas.

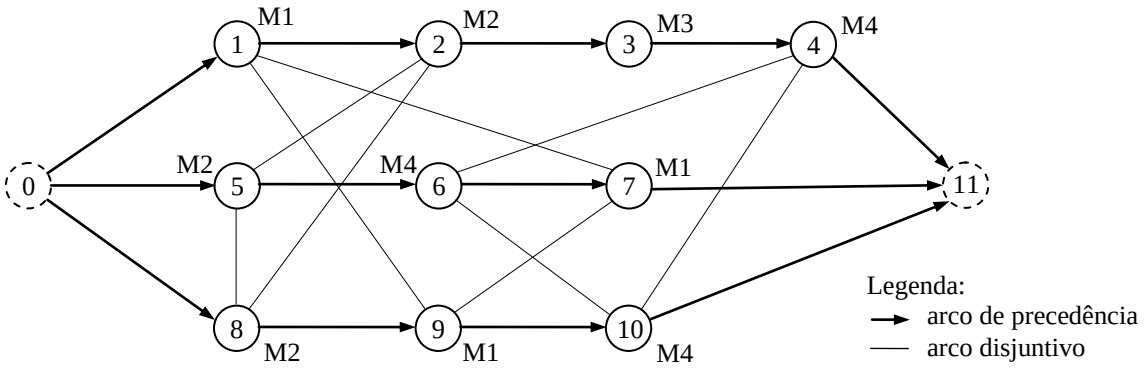

Figura 1: Grafo disjuntivo.

No grafo disjuntivo acima os vértices 0 e 11, tracejados, representam, respectivamente, o início e o término das operações. A primeira tarefa é composta pelas operações 1, 2, 3 e 4, nessa ordem; a segunda, pelas operações 5, 6 e 7; e a terceira, pelas operações 8, 9 e 10. Arcos disjuntivos unem as operações 1, 7 e 9, indicando que elas devem ser processadas na mesma máquina, M1. As operações 2, 5 e 8 possuem arcos disjuntivos entre si pois todas demandam processamento em M2. A operação 3 é a única a ser executada em M3. Já as operações 4, 6 e 10 precisam ser executadas na máquina M4. A resolução do problema corresponde à definição da direção dos arcos disjuntivos de modo que seja determinada a ordem de processamento das operações a serem processadas em cada uma das máquinas. Assim se obtém um grafo orientado que, se for acíclico, representa uma solução viável do JS. Um caminho de peso máximo nesse grafo orientado é denominado caminho crítico e seu peso corresponde ao *makespan*.

O *Job Shop* Flexível (JSF) é uma generalização do JS na qual pode haver mais de uma máquina capaz de executar uma mesma operação, sendo que as máquinas capazes de processar cada operação podem ser idênticas ou não. Essa flexibilidade torna possível uma tarefa ser processada através de diversos caminhos alternativos no grafo disjuntivo, o que aumenta significativamente o número de soluções viáveis. Na resolução do JSF são empregadas diferentes abordagens e métodos de resolução, majoritariamente heurísticos.

As abordagens de resolução se distinguem pela decomposição ou não do JSF em problemas menos complexos que podem ser atacados isoladamente. O JSF pode ser visto como um problema constituído por duas fases. A primeira delas é a atribuição de cada uma das

operações a uma das máquinas habilitadas a realizar seu processamento. Essa etapa, por vezes, é denominada roteamento, já que as atribuições especificam os caminhos que as tarefas percorrerão através das máquinas. Definidas tais atribuições, vem a segunda fase. Ela é constituída pelo sequenciamento das operações nas máquinas, ou seja, pela resolução do JS resultante após as atribuições da primeira fase. Esse tipo de resolução é denominada abordagem hierárquica. Em contraste com a abordagem hierárquica existe a abordagem integrada, pela qual um determinado método de resolução busca obter soluções para o JSF tratando, ao mesmo tempo, atribuições e sequenciamentos.

Mishra e Pandey (1989) e Brucker *et al*. (1990) estão entre os primeiros trabalhos a mencionar o JSF, embora façam uso de uma terminologia distinta para se referir ao problema. Mishra e Pandey (1989) utilizam simulação computacional para comparar um método heurístico proposto com diferentes regras de prioridade que visam minimizar o *makespan* e a carga de trabalho média das máquinas. Brucker *et al*. (1990) propõem um algoritmo de complexidade polinomial para o JSF com duas tarefas. Com relação a problemas mais realistas, com quantidades maiores de tarefas, usualmente são utilizadas técnicas alternativas. Brandimarte (1993) foi o primeiro trabalho a utilizar a abordagem hierárquica. Para tanto foram empregadas regras de prioridade para estabelecer as rotas das tarefas pelas máquinas às quais suas operações eram designadas. Na fase seguinte, as sequências foram obtidas através da heurística Busca Tabu (BT). Evidentemente, caso a abordagem hierárquica seja escolhida, a resolução ótima de ambos os subproblemas do JSF ainda não garante a solução ótima do problema composto.

Outro aspecto relevante a ser considerado nos problemas de JSF se refere ao grau de sua flexibilidade. Kacem *et al*. (2002) classificam os problemas de JSF em JSF-Total e JSF-Parcial. No JSF-Total cada uma das operações pode ser executada em qualquer uma das máquinas. Já no JSF-Parcial cada uma das operações está associada a um subconjunto de máquinas aptas a realizar seu processamento. Naturalmente, podem haver inúmeros graus de flexibilidade entre o JSF-Total e o JS.

O fato do JS ser um dos mais difíceis problemas de otimização combinatória, segundo Lawer *et al*. (1993), e do JSF ser, no mínimo, tão difícil quanto o JS, conduz muitos pesquisadores a aplicar e desenvolver métodos heurísticos. Kacem *et al*. (2002) utilizam a abordagem hierárquica associada a um método evolutivo. Zhang e Gen (2005) propõem um Algoritmo Genético (AG) baseado numa representação multiestágio no qual as operações de uma tarefa descrevem um caminho através das máquinas. Os experimentos envolvem JSF-Total e JSF-Parcial tendo como objetivo a minimização do *makespan* e da carga de trabalho total e média nas máquinas. Seus resultados são superiores aos de AGs que usam outras representações e aos do método heurístico baseado no *Shortest Processing Time* (SPT). Chan *et al*. (2006) utilizam um AG de dois níveis para resolver o JSF-Total através da abordagem hierárquica. Ho *et al*. (2006) desenvolveram uma arquitetura que funde AG e aprendizagem de máquina para uma aplicação do JSF na indústria eletrônica de circuito impresso cujo objetivo é minimizar o *makespan*. Gutiérrez e García-Magariño (2011) buscam minimizar o *makespan* usando AG aliado a heurísticas de reparação. Dauzère-Pérès e Paulli (1997) apresentam uma versão estendida do grafo disjuntivo proposto por Balas (1969) e utilizam Busca Tabu (BT) em uma vizinhança que não distingue alterações nas atribuições ou nos sequenciamentos. Zhang *et al*. (2009) hibridizam a metaheurística *Particle Swarm Optimization* (PSO) com BT num método multicritério de abordagem integrada. Li *et al*. (2010) desenvolvem um método híbrido multiobjetivo que envolve BT e *Variable Neighborhood Search* (VNS).

Em contraste com o grande volume de estudos que evocam técnicas heurísticas, a quantidade de trabalhos que procuram resolver o JSF através de métodos exatos é bastante pequena. Nesse campo, um dos artigos mais relevantes é Fattahi *et al*. (2007), onde um modelo de programação inteira mista é proposto para o JSF. Além da modelagem, Fattahi *et al*. (2007) propõem um conjunto de 20 problemas de teste específicos do JSF, sendo 10 de pequeno porte e 10 de médio e grande porte. O modelo de programação inteira mista elaborado é utilizado para resolver tais problemas de teste com o tempo de execução computacional limitado a uma hora. As soluções ótimas, obtidas para os problemas de pequeno porte, e os limitantes superiores e

inferiores, obtidos nos problemas de médio e grande porte, são utilizados em comparações com resultados alcançados por métodos heurísticos. Os métodos heurísticos utilizados por Fattahi *et al.* (2007) são BT e *Simulated Annealing* (SA). O trabalho explora a abordagem integrada aplicando BT e SA isoladamente. Já a abordagem hierárquica usa BT e SA isolados e também combinados. Neste último caso uma metaheurística gera as atribuições e a outra, os sequenciamentos.

Özgüven *et al*. (2010) elaboram outro modelo de programação inteira mista para o JSF a partir de modificações do modelo de Manne (1960) que permitem incorporar a flexibilidade das máquinas. Esse modelo, sendo consideravelmente mais conciso, é confrontado com o proposto por Fattahi *et al.* (2007) por meio da resolução de seus 20 problemas de teste. Os resultados indicam que o modelo de Özgüven *et al*. (2010) teve melhor desempenho. Ele obtém solução ótima em todos os 10 problemas de pequeno porte, assim como seu concorrente, porém em menor tempo. Özgüven *et al*. (2010) alcançam solução ótima em 5 dos 10 problemas de médio porte, enquanto Fattahi *et al.* (2007) não conseguem nenhum ótimo nesse conjunto. Nos problemas de teste em que nenhum dos modelos atinge uma solução ótima, Özgüven *et al*. (2010) fornecem melhores limitantes.

Evidentemente, problemas com a complexidade do JS ou do JSF exigem técnicas sofisticadas para que resultados razoáveis possam ser obtidos em um tempo aceitável. Por tal razão, a exploração de métodos não-exatos é observada com tão alta frequência. Muitas vezes é preferível obter rapidamente resultados não-ótimos a esperar dias ou semanas para se conseguir uma solução ótima. Contudo, é necessário que os resultados oriundos desses métodos não-exatos, especialmente heurísticos, possam ser avaliados de modo que determinados níveis de qualidade de solução sejam atendidos. Por esse motivo, o desenvolvimento e o aprimoramento de modelos que permitam a resolução exata é fundamental. Modelos de programação inteira mista bem elaborados, aliados à grande evolução dos recentes bons pacotes comerciais baseados em métodos como o *branch-and-bound* e o *branch-and-cut*, podem ser capazes de fornecer, senão uma solução ótima, limitantes satisfatórios para muitos problemas. Esses limitantes então podem ser utilizados para balizar a avaliação de métodos heurísticos, distinguindo com maior clareza bons e maus resultados.

Dessa forma, o objetivo deste trabalho é contribuir com o desenvolvimento de modelos que permitam a resolução exata do JSF, tornando possível a obtenção de resultados de melhor qualidade que devem servir como parâmetro para a avaliação de outras técnicas de resolução. Em seguida são apresentados os modelos estudados e propostos, seguidos pelos experimentos efetuados.

2. Modelos

Nesta seção são descritos três modelos para o JSF. O primeiro deles é o apresentado em Özgüven *et al*. (2010); o segundo é uma adaptação do primeiro para incorporar relações de precedências mais gerais; e o terceiro é uma nova proposta de modelagem para o JSF.

2.1. Modelo de Özgüven *et al*. (2010)

O primeiro modelo é o proposto por Özgüven *et al*. (2010), visto que se mostrou o melhor disponível na literatura. Ele deriva da formulação de Manne (1960) e trabalha com menos variáveis e menos restrições do que o modelo de Fattahi *et al*. (2007). O objetivo é minimizar o *makespan* obedecendo as demais restrições do JSF.

Índices e conjuntos:

J	Conjunto de tarefas;
M	Conjunto de máquinas;
O	Conjunto que contém todas as operações de todos os *jobs*;
i, i'	Tarefas ($i, i' \in J$);
j, j'	Operações ($j, j' \in O$);
k	Máquina ($k \in M$);

O_i Conjunto ordenado das operações da tarefa i, $O_i = \{O_{if(i)}, O_{if(i)+1}, \ldots, O_{il(i)}\}$, onde $O_{if(i)}$ é o primeiro elemento de O_i e $O_{il(i)}$ o último ($O_i \subseteq O$);
M_j Conjunto de máquinas alternativas onde a operação j pode ser processada ($M_j \subseteq M$);
$M_j \cap M_{j'}$ Conjunto de máquinas onde ambas as operações j e j' podem ser processadas.

Parâmetros:
t_{ijk} Tempo de processamento da operação O_{ij} na máquina k;
L Número grande.

Variáveis:
X_{ijk} Assume 1 se a máquina k é selecionada para a operação O_{ij} e 0, caso contrário;
S_{ijk} Instante de início de processamento da operação O_{ij} na máquina k;
C_{ijk} Instante de término de processamento da operação O_{ij} na máquina k;
$Y_{iji'j'k}$ Assume 1 se a operação O_{ij} precede a operação $O_{i'j'}$ na máquina k e 0, caso contrário;
C_i Instante de término de processamento da tarefa i;
$C_{\max}$ Máximo instante de término de processamento dentre todas as tarefas, o *makespan*.

Modelo de programação inteira mista:

Função objetivo:

Minimizar $C_{\max}$

Sujeito a:

$$\sum_{k \in M_j} X_{ijk} = 1 \qquad \forall i \in J, \forall j \in O_i \qquad (1)$$

$$S_{ijk} + C_{ijk} \leq (X_{ijk}) \cdot 2L \qquad \forall i \in J, \forall j \in O_i, \forall k \in M_j \qquad (2)$$

$$C_{ijk} \geq S_{ijk} + t_{ijk} - (1 - X_{ijk}) \cdot L \qquad \forall i \in J, \forall j \in O_i, \forall k \in M_j \qquad (3)$$

$$S_{ijk} \geq C_{i'j'k} - (Y_{iji'j'k}) \cdot L \qquad \forall i < i', \forall j \in O_i, \forall j' \in O_{i'}, \forall k \in M_j \cap M_{j'} \qquad (4)$$

$$S_{i'j'k} \geq C_{ijk} - (1 - Y_{iji'j'k}) \cdot L \qquad \forall i < i', \forall j \in O_i, \forall j' \in O_{i'}, \forall k \in M_j \cap M_{j'} \qquad (5)$$

$$\sum_{k \in M_j} S_{ijk} \geq \sum_{k \in M_{j-1}} C_{i,j-1,k} \qquad \forall i \in J, \forall j \in O_i - \{O_{if(i)}\} \qquad (6)$$

$$C_i \geq \sum_{k \in M_{O_{il(i)}}} C_{i,O_{il(i)},k} \qquad \forall i \in J \qquad (7)$$

$$C_{\max} \geq C_i \qquad \forall i \in J \qquad (8)$$

$$X_{ijk} \in \{0,1\} \qquad \forall i \in J, \forall j \in O_i, \forall k \in M_j$$

$$S_{ijk} \geq 0 \qquad \forall i \in J, \forall j \in O_i, \forall k \in M_j$$

$$C_{ijk} \geq 0 \qquad \forall i \in J, \forall j \in O_i, \forall k \in M_j$$

$$Y_{iji'j'k} \in \{0,1\} \qquad \forall i < i', \forall j \in O_i, \forall j' \in O_{i'}, \forall k \in M_j \cap M_{j'}$$

$$C_i \geq 0 \qquad \forall i \in J$$

A função objetivo é minimizar o máximo dos instantes de término dentre todas as tarefas, que é exatamente o *makespan*; O conjunto de restrições do tipo (1) garante que cada operação seja designada a exatamente uma máquina dentre as habilitadas para seu processamento; As restrições do tipo (2) asseguram que, caso uma determinada operação não seja atribuída a certa máquina, seus instantes de início e término de processamento nessa máquina sejam nulos; As restrições (3) fazem com que, caso uma determinada operação seja atribuída a certa máquina, seu instante de término seja, no mínimo, a soma de seu instante de início de processamento e de seu tempo de processamento naquela máquina; As restrições (4) e (5) evitam que operações alocadas a uma mesma máquina sejam colocadas em execução concorrentemente; As restrições do tipo (6) exigem que as operações sejam processadas na ordem determinada por

cada tarefa; As restrições (7) forçam que o instante de término de cada tarefa seja maior ou igual ao instante de término de sua última operação; As restrições (8) definem o *makespan*. O modelo é apresentado com correções tipográficas nos somatórios das restrições (6) e (7) com relação ao mostrado em Özgüven *et al*. (2010).

2.2. Modelo adaptado

Nesta seção são descritas adaptações e aprimoramentos do modelo original de Özgüven *et al*. (2010). Esse modelo trabalha com uma lista ordenada de operações para cada tarefa. Isso fica especialmente evidenciado nas restrições (6) e (7). Porém, a representação clássica do JS e do JSF envolve arcos de precedência. As adaptações têm o intuito de fazer com que o modelo trabalhe com as precedências declaradas explicitamente. Para compor o modelo adaptado devem ser considerados:

Índices e conjuntos:

l	Tarefa ($l \in J$);
m	Operação ($m \in O$);
w	Máquina ($w \in M$);
P_{ij}	Conjunto de pares ordenados nos quais a operação O_{ij} é o último elemento do arco de precedência ($P_{ij} \subseteq O \times \{O_{ij}\}$);
P	União dos conjuntos P_{ij}, para todo $i \in J$ e $j \in O_i$.

A primeira alteração consiste no seguinte ajuste no modelo proposto por Özgüven *et al*. (2010). Substituir o conjunto de restrições do tipo (2) pelos dois conjuntos de restrições abaixo:

$$S_{ijk} \leq X_{ijk} \cdot L \qquad \forall i \in J, \forall j \in O_i, \forall k \in M_j \qquad (9)$$

$$C_{ijk} \leq X_{ijk} \cdot L \qquad \forall i \in J, \forall j \in O_i, \forall k \in M_j \qquad (10)$$

A soma das restrições (9) e (10) é equivalente às restrições (2). Logo, essa alteração deixa o modelo mais justo no sentido de que qualquer solução viável do modelo adaptado é viável no modelo original.

Para que o modelo possa lidar com os arcos de precedência, as restrições (6) e (7) devem ser substituídas, respectivamente, pelas restrições a seguir:

$$\sum_{k \in M_j} S_{ijk} \geq \sum_{w \in M_m} C_{imw} \qquad \forall i \in J, \forall j \in O_i, \forall m : (O_{im}, O_{ij}) \in P_{ij} \qquad (11)$$

$$C_i \geq \sum_{k \in M_j} C_{ijk} \qquad \forall i \in J, \forall j \in O_i \qquad (12)$$

Desse modo, em vez do modelo considerar que cada operação de uma tarefa deve ser iniciada após o término da operação listada anteriormente na série, como nas restrições (6), o conjunto das restrições do tipo (11) exige que cada operação inicie seu processamento após a finalização das operações que a precedem. As restrições (12), em vez de basear o instante de término das tarefas na última operação, como em (7), verifica todas operações da tarefa.

2.3. Modelo novo

Esta seção descreve um modelo diferente para o JSF. Neste modelo não é feita a distinção entre operações de acordo com a tarefa a que pertencem. São considerados apenas o conjunto das operações de todas as tarefas e suas relações de precedência. O objetivo é minimizar o *makespan* obedecendo as restrições do JSF.

Índices e conjuntos:

O	Conjunto que contém todas as operações de todos os *jobs*;
M	Conjunto de máquinas;
j, j'	Operações ($j, j' \in O$);
k	Máquina ($k \in M$);

O_k Conjunto de operações que podem ser processadas na máquina k ($O_k \subseteq O$).
E_k Conjunto de pares ordenados de elementos distintos de O_k ($E_k \subseteq O_k \times O_k$);
E União dos conjuntos E_k, para todo $k \in M$;
P Conjunto pares ordenados correspondentes aos arcos de precedência ($P \subseteq O \times O$).

Parâmetros:

t_{jk} Tempo de processamento da operação j na máquina k (para simplificação do modelo, considera-se $t_{jk} > 0$);
L Número grande.

Variáveis:

$Y_{jj'}$ Assume 1 se j precede j' e ambas são processadas na mesma máquina, 0 caso contrário;
X_{jk} Assume 1 se j é processada na máquina k, 0 caso contrário;
S_j Instante de início de processamento da operação j;
c_j Tempo que a operação j levou para ser processada na máquina a que foi atribuída;
$C_{\max}$ *Makespan*.

Modelo de programação inteira mista:

Função objetivo:

Minimizar $C_{\max}$

Sujeito a:

$$\sum_{k \in M} X_{jk} = 1 \qquad \forall j \in O \qquad (13)$$

$$X_{jk} = 0 \qquad \forall k \in M, \forall j \notin O_k \qquad (14)$$

$$c_j = \sum_{k \in M} t_{jk} \cdot X_{jk} \qquad \forall j \in O \qquad (15)$$

$$S_j + c_j \leq C_{\max} \qquad \forall j \in O \qquad (16)$$

$$Y_{jj'} + Y_{j'j} \geq X_{jk} + X_{j'k} - 1 \qquad \forall k \in M, \forall (j, j') \in E_k \qquad (17)$$

$$S_j + c_j - L \cdot (1 - Y_{jj'}) \leq S_{j'} \qquad \forall (j, j') \in E \qquad (18)$$

$$S_j + c_j \leq S_{j'} \qquad \forall (j, j') \in P \qquad (19)$$

$$S_j \geq 0 \qquad \forall j \in O \qquad (20)$$

$$X_{jk} \in \{0,1\} \qquad \forall k \in M, \forall j \in O$$

$$Y_{jj'} \in \{0,1\} \qquad \forall (j, j') \in E$$

A atribuição de máquinas para as operações é válida devido às restrições (13) e (14). As restrições do tipo (15) garantem que c_j é o tempo de processamento da operação j na máquina à qual ela foi atribuída. As restrições (16) e (20), juntamente com a função objetivo, garantem que $C_{\max}$ tenha o valor do *makespan* do escalonamento encontrado. Se as operações j e j' são processadas na mesma máquina, então o conjunto das restrições do tipo (17) faz com que pelo menos uma dentre as variáveis $Y_{jj'}$ e $Y_{j'j}$ tenha valor 1. Para serem interpretadas como relação de precedência de operações na mesma máquina, $Y_{jj'}$ e $Y_{j'j}$ não podem ter valor 1 simultaneamente. Isso é garantido pelas restrições do tipo (18) que também garantem que não há sobreposição no processamento das operações. Pelas restrições (18) e (19) temos que as relações de precedência não formam um ciclo.

É preciso notar que as restrições permitem soluções viáveis com uma variável $Y_{jj'} = 1$ mesmo que j e j' não sejam processadas na mesma máquina. No entanto, isso não causa problema, pois tais soluções viáveis nunca são melhores que uma solução ótima. Isso porque atribuir o valor 1 a uma variável $Y_{jj'}$ equivale a acrescentar um arco de j para j' no grafo disjuntivo, o que não diminui o tamanho do caminho crítico do grafo.

2. Resultados computacionais

Nesta seção são apresentados os resultados e comparações entre os modelos estudados. Os três modelos foram implementados em CPLEX versão 12.1. Os experimentos foram realizados em um servidor com processador Intel Xeon E5440 de 2,83GHz utilizando somente um núcleo, 32GB de RAM e sistema operacional Linux. Foram utilizados os problemas de teste propostos por Fattahi *et al*. (2007). Na Tabela 1 são apresentadas as dimensões desses problemas de teste em termos do número de tarefas, *i*; do número de operações por tarefa, *j*; e do número de máquinas, *k*. Para cada um desses problemas, o número de operações por tarefa é o mesmo. Ainda na Tabela 1 são listadas as quantidades de variáveis inteiras, de variáveis não-inteiras e de restrições de cada modelo. Os problemas de teste SFJS são consideradas de pequeno porte e as MFJS são consideradas de médio a grande porte. Por conveniência, será denotado por MO o modelo de Özgüven *et al*. (2010), MA o modelo adaptado de Özgüven *et al*. (2010) e MN o modelo novo.

Tabela 1: Número de variáveis inteiras, não-inteiras e número de restrições de cada modelo.

Prob.	Tam. (*i*,*j*,*k*)	MO			MA			MN		
		#var. int.	#var. não-int.	#rest.	#var. int.	#var. não-int.	#rest.	#var. int.	#var. não-int.	#rest.
SFJS1	2, 2, 2	16	19	42	16	19	52	24	9	74
SFJS2	2, 2, 2	10	15	30	10	15	38	24	9	50
SFJS3	3, 2, 2	26	24	67	26	24	80	48	13	131
SFJS4	3, 2, 2	26	24	67	26	24	80	48	13	137
SFJS5	3, 2, 2	36	28	87	36	28	102	48	13	171
SFJS6	3, 3, 2	39	34	99	39	34	120	108	19	219
SFJS7	3, 3, 5	36	40	93	36	40	117	126	19	200
SFJS8	3, 3, 4	45	40	111	45	40	135	117	19	239
SFJS9	3, 3, 3	55	40	131	55	40	155	108	19	306
SFJS10	4, 3, 5	48	45	124	48	45	152	204	25	260
MFJS1	5, 3, 6	99	70	241	99	70	275	315	31	530
MFJS2	5, 3, 7	128	84	291	128	84	340	330	31	639
MFJS3	6, 3, 7	190	103	422	190	103	482	450	37	958
MFJS4	7, 3, 7	250	120	549	250	120	619	588	43	1262
MFJS5	7, 3, 7	243	118	535	243	118	604	588	43	1255
MFJS6	8, 3, 7	307	133	670	307	133	748	744	49	1596
MFJS7	8, 4, 7	475	165	1022	475	165	1124	1248	65	2612
MFJS8	9, 4, 8	519	182	1119	519	182	1232	1584	73	2889
MFJS9	11, 4, 8	751	218	1601	751	218	1737	2288	89	4172
MFJS10	12, 4, 8	899	237	1906	899	237	2054	2688	97	5006

Como esperado, é possível verificar que o MO e o MA utilizam o mesmo número de variáveis. Já o MN usa uma quantidade razoavelmente superior, em alguns casos, quase o triplo que os demais. Quanto às restrições, o MA trabalha com uma quantidade ligeiramente superior que o MO devido ao desmembramento das restrições do tipo (2) e restrições extras do tipo (12). O MN, por sua vez, impõe mais restrições. No maior problema de teste o número de restrições do MN é superior ao dobro das restrições do MA. Contudo, não é possível afirmar que menores quantidades de variáveis ou de restrições fazem um modelo melhor que outro. Pelo contrário, mais restrições podem, por exemplo, restringir mais o espaço de busca de soluções, facilitando o trabalho de métodos de enumeração implícita, como o *branch-and-bound*.

Os três modelos apresentados utilizam um parâmetro *L*, mencionado como um número grande. Para que os modelos sejam válidos, *L* deve ser uma delimitação superior do *makespan* ótimo. Este trabalho utiliza a seguinte delimitação como *L*. Para cada operação, toma-se o maior

tempo de processamento dentre todas as máquinas habilitadas a realizar sua execução e, então, soma-se esses tempos máximos de todas as operações. Essa soma foi o valor adotado de L para os três modelos nos experimentos apresentados.

No artigo de Özgüven *et al*. (2010), as restrições do tipo (2) aparecem sem o fator 2 multiplicando o L. Manter tais restrições sem esse fator 2 forçaria a adoção de um valor de L para tal modelo duas vezes maior para manter sua validade, já que se pode deduzir que, para as restrições (2) não eliminarem nenhuma solução ótima, o valor de L deve ser basicamente pelo menos duas vezes o *makespan* ótimo. O acréscimo do fator 2 foi feito nas restrições (2) para permitir o uso do mesmo valor de L para os três modelos. Sem a multiplicação pelo fator 2, a utilização de um valor de L menor que duas vezes o *makespan* ótimo pode resultar em um modelo incorreto para o problema, com valor ótimo maior que o verdadeiro valor ótimo do problema. De fato, no estudo experimental apresentado por Özgüven *et al*. (2010), o valor ótimo estabelecido para um dos problemas de teste está incorreto, como será discutido mais adiante. Possivelmente o valor de L usado no experimento não era suficientemente grande.

As comparações mais importantes entre os modelos podem ser feitas pela Tabela 2. Ela apresenta, para cada um dos três modelos testados e para cada problema discriminado na primeira coluna: o *makespan*, citado como C_{max}; o *gap*, indicando a diferença relativa entre os limitantes superior e inferior de C_{max}; e o tempo de CPU, em segundos.

Tabela 2: Resultados dos 20 problemas de teste de Fattahi *et al*. (2007).

Prob.	MO			MA			MN		
	C_{max}	*gap* (%)	Tempo de CPU(s)	C_{max}	*gap* (%)	Tempo de CPU(s)	C_{max}	*gap* (%)	Tempo de CPU(s)
SFJS1	66	0,00	0,00	66	0,00	0,00	66	0,00	0,00
SFJS2	107	0,00	0,01	107	0,00	0,00	107	0,00	0,00
SFJS3	221	0,00	0,00	221	0,00	0,01	221	0,00	0,02
SFJS4	355	0,00	0,00	355	0,00	0,00	355	0,00	0,01
SFJS5	119	0,00	0,01	119	0,00	0,01	119	0,00	0,01
SFJS6	320	0,00	0,01	320	0,00	0,00	320	0,00	0,01
SFJS7	397	0,00	0,01	397	0,00	0,01	397	0,00	0,00
SFJS8	253	0,00	0,03	253	0,00	0,02	253	0,00	0,03
SFJS9	210	0,00	0,01	210	0,00	0,02	210	0,00	0,02
SFJS10	516	0,00	0,02	516	0,00	0,00	516	0,00	0,01
MFJS1	468	0,00	0,29	468	0,00	0,32	468	0,00	0,31
MFJS2	446	0,00	0,60	446	0,00	1,06	446	0,00	0,45
MFJS3	466	0,00	2,47	466	0,00	11,84	466	0,00	1,41
MFJS4	554	0,00	24,06	554	0,00	110,97	554	0,00	16,57
MFJS5	514	0,00	10,66	514	0,00	3,39	514	0,00	2,46
MFJS6	634	0,00	44,67	634	0,00	38,79	634	0,00	35,20
MFJS7	[764,00; 881]	13,28	3600	[784,00; 881]	11,01	3600	879	0,00	1368,28
MFJS8	[764,00; 899]	15,02	3600	[764,62; 895]	14,57	3600	[764,00; 893]	14,45	3600
MFJS9	[808,00; 1098]	26,41	3600	[835,67; 1097]	23,82	3600	[818,12; 1088]	24,81	3600
MFJS10	[1000,00; 1237]	19,16	3600	[970,20; 1225]	20,80	3600	[944,40; 1318]	28,35	3600
Média		18,468[a]	216,638[b]		17,550[a]	221,555[b]		16,899[a]	83,811[b]

[a] *gap* médio relativo aos problemas MFJS7 a MFJS10;
[b] tempo médio relativo aos problemas SFJS1 a MFJS7.

Conforme a Tabela 2, em uma hora de execução todos os modelos foram capazes de resolver de modo ótimo os problemas de SFJS1 a SFJS10 e de MFJS1 a MFJS6. Assim, para esses 16 problemas de teste, o *gap* sempre foi igual a zero. Apenas o MN alcançou uma solução ótima para MFJS7 em no máximo uma hora. Nos três demais problemas de teste, mesmo após uma hora de processamento computacional, os três modelos não conseguiram zerar o *gap*, não indicando, portanto, uma solução ótima. Nesses casos, cada coluna C_{max} exibe, para cada problema, os limitantes inferior e superior encontrados por cada modelo. Por exemplo, no problema de teste MFJS7 o MO encontrou 764,00 como limitante inferior e 881 como limitante superior. Ao lado desses valores consta o *gap* entre eles.

Com o problema MFJS7, a formulação desenvolvida, o MN, encontrou uma solução ótima, enquanto MO e MA tiveram *gap* de 13,28% e 11,01%, nessa ordem. Com o problema MFJS8 o MN também obteve o melhor *gap*, 14,45%, seguido do MA com 14,57% e do MO com 15,02%. Já com o problema de teste MFJS9 as adaptações incorporadas no MA conduziram a melhor resultado. Seu *gap* foi igual a 23,82% enquanto o MN alcançou 24,81%. Por sua vez o MO chegou a 26,41%. O problema MFJS10 foi o único em que o MO chegou ao menor *gap*, 19,16% contra 20,80% do MA e 28,35% do MN. A Tabela 2 traz as médias do *gap* para cada modelo considerando apenas os problemas de teste nas quais nem todos os modelos obtiveram *gap* zero. Esses valores médios sugerem a superioridade do MN frente aos demais. Para os últimos 4 problemas ele obteve *gap* médio igual a 16,899. O MA, com segunda melhor média, ficou em 17,550% enquanto a média de *gap* do MO foi 18,468%.

Outro aspecto relevante a ser observado é o tempo de resolução gasto pelos modelos para encontrar uma solução ótima. Pela Tabela 2 se nota que, nos problemas de menor porte, SFJS1 a SFJS10, as diferenças não são significativas. Assim, melhores observações podem ser feitas a partir de MFJS1. A Tabela 2 traz a média dos tempos de execução, para cada modelo, dos problemas de teste para as quais ao menos um modelo obteve solução ótima. Com média igual a 83,811s, o MN foi o mais rápido, seguido do MO, com 216,638s e do MA com média 221,555s.

Cabe ainda observar que o *makespan* ótimo estabelecido no artigo de Özgüven *et al*. (2010) para o problema de teste MFJS4 é 564. No entanto, em nosso estudo experimental encontramos, para esse problema, o escalonamento apresentado na Figura 2 cujo *makespan* é 554. Essa diferença pode ocorrer devido à associação das restrições (2) sem o fator de multiplicação 2 a um valor não suficientemente grande de *L*, o que pode fazer o modelo original eliminar a solução ótima com *makespan* igual a 554, resultando no valor 564.

A Figura 2 exibe um escalonamento viável do problema MFJS4 com *makespan* 554. As máquinas são denotadas por *M*1, *M*2, …, *M*7. A sequência de operações de uma tarefa *i* é (3*i* - *2*, 3*i* -1, 3*i*) sendo indicada pelas setas na Figura 2. A Tabela 3 apresenta os instantes de início e término das operações nas máquinas.

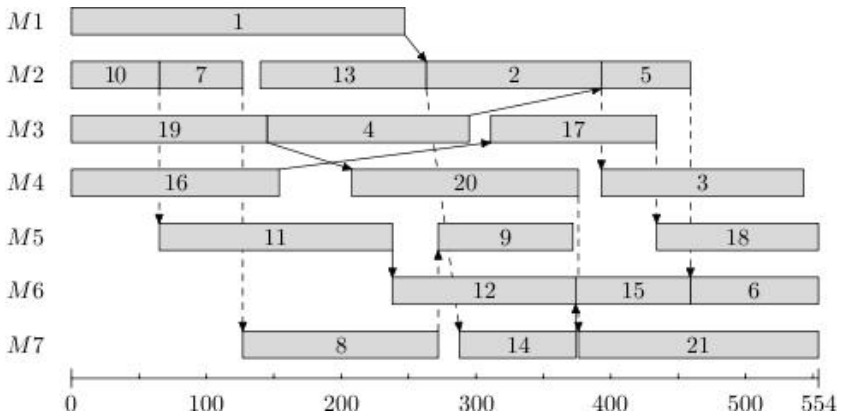

Figura 2: Uma solução do problema MFJS4 com *makespan* 554.

Tabela 3: Instantes de início e término das operações na solução da Figura 2.

Operação	Instante de início	Instante de término	Máquina
1	0	247	1
2	263	393	2
3	393	543	4
4	145	295	3
5	393	459	2
6	459	554	6
7	65	127	2
8	127	272	7
9	272	372	5
10	0	65	2
11	65	238	5
12	238	374	6
13	140	263	2
14	288	374	7
15	374	459	6
16	0	154	4
17	311	434	3
18	434	554	5
19	0	145	3
20	208	376	4
21	376	554	7

3. Conclusões e trabalhos futuros

Neste trabalho foram estudados diferentes modelos do JSF com o intuito de identificar a formulação mais adequada para resoluções através de métodos exatos. Os resultados obtidos sugerem que o modelo novo, em geral, obteve melhores soluções em menor tempo computacional. Apesar disso, nesses experimentos, o MA obteve menor *gap* em dois problemas de teste, enquanto o MO, em uma delas. No entanto, este é um estudo preliminar e mais experimentos serão feitos. Portanto, um dos próximos passos será criar mais problemas de teste para avaliar os modelos. Outros trabalhos futuros consistem em estudar formas de melhorar os modelos; estudar a influência do valor de L na resolução dos problemas; explorar variações do JSF e métodos alternativos de resolução.

Agradecimentos: Este trabalho foi financiado parcialmente por CAPES, CNPq e pelo *GOLD Advanced Research grant* da Hewlett-Packard Company, patrocinado por HP Labs e HP Brazil R&D.

Referências

Balas, E. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm, *Operations Research*, v. 17, n. 6, pp. 941-957, 1969.

Brandimarte, P. Routing and scheduling in a flexible job shop by tabu search, *Annals of Operations Research*, v. 41, n. 3, 157-183, 1993.

Brucker, P. e Schile, R. Job-shop scheduling with multi-purpose machines, *Computing*, v. 45, n. 4, pp. 369-375, 1990.

Chan, F.; Wong, T e Chan, L. Flexible job-shop scheduling problem under resource constraints, *International Journal of Production Research*, v. 44, n. 11, pp. 2071-2089, 2006.

Dauzère-Pérès, S. e Paulli, J. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search, *Annals of Operations Research*, v. 70, pp. 281-306, 1997.

Fattahi, P.; Mehrabad, M. e Jolai, F. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems, *Journal of Intelligent Manufacturing*, v. 18, pp. 331-342, 2007.

Garey, M; Johnson, D. e Sethi, R. The Complexity of Flowshop and Jobshop Scheduling, *Mathematics of Operations Research*, v. 1, n. 2, pp. 117-129, 1976.

Gutiérrez, C. e García-Magariño, I. Modular design of a hybrid genetic algorithm for a flexible job–shop scheduling problem, *Knowledge-Based Systems*, v. 24, pp. 102-112, 2011.

Ho, N.; Tay, J. e Lai, E. An effective architecture for learning and evolving flexible job-shop schedules, *European Journal of Operational Research*, v. 179, pp. 316–333, 2006.

Kacem, I.; Hammadi, S. e Borne, P. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems, *IEEE Transactions on Systems, Man, and Cybernetics*, v. 32, n. 1, pp. 1-13, 2002.

Lawer, E.; Lenstra, J.; Rinnooy Kan, A. e Shmoys, D. Sequencing and scheduling: algorithms and complexity em S. Graves, A. Rinnooy Kane P. Zipkin (Eds.), *Logistics of production and inventory*, North-Holland, 1993.

Li, J.; Pan, Q. e Liang, Y. An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems, *Computers & Industrial Engineering*, v. 59, pp.647-662, 2010.

Manne, A. On the job-shop scheduling problem, *Operations Research*, v. 8, pp. 219-223, 1960.

Mishra, P. e Pandey, P. Simulation modeling of batch job shop type flexible manufacturing systems, *Journal of Mechanical Working Technology*, v. 20, 441-450, 1989.

Özgüven, C.; Özbakır, L. e Yavuz, Y. Mathematical models for job-shop scheduling problems with routing and process plan flexibility, *Applied Mathematical Modelling*, v. 34, pp. 1539-1548, 2010.

Vilcot, G. e Billaut, J. A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem, *European Journal of Operational Research*, v. 190, 398-411, 2008.

Zhang, G.; Shao, X.; Li, P. e Gao, L. An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem, *Computers & Industrial Engineering*, v. 56, pp. 1309–1318, 2009.

Zhang, H. e Gen, M. Multistage-based genetic algorithm for flexible job-shop scheduling problem, *Complexity International*, v. 11, pp. 223-232, 2005.