DOI: 10.48011/asba.v2i1.1312

Tradução Automática de Problemas de Escalonamento Job Shop Flexível com Bloqueio para Autômatos Utilizando a TCS

Daniel Sarsur C. * **Patrícia N. Pena** **
Ricardo H. C. Takahashi ***

* *Programa de Pós-Graduação em Engenharia Elétrica - Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brasil (e-mail: danielsc@ufmg.br)*
** *Departamento de Engenharia Eletrônica - Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brasil (e-mail: ppena@ufmg.br).*
*** *Departamento de Matemática - Universidade Federal de Minas Gerais, Belo Horizonte, MG, Brasil (e-mail: taka@mat.ufmg.br).*

Abstract: This work presents an algorithm to automatically translate the search space of a Blocking Flexible Job Shop Scheduling Problem modeling into automata using the Supervisory Control Theory. Some problems of the literature are analyzed in their textual form and the algorithm returns an automaton that implements the closed-loop behavior. A heuristic that minimizes makespan is applied. This methodology faces memory usage boundaries, but it was able to find makespan values near to those in the literature.

Resumo: Este trabalho apresenta um algoritmo para converter automaticamente o espaço de busca de um problema de escalonamento *job shop* flexível com bloqueio em autômatos usando a Teoria de Controle Supervisório. Alguns problemas da literatura são analisados em sua forma textual e o algoritmo retorna o autômato que representa o comportamento em malha fechada de cada um deles. Uma heurística de minimização de *makespan* é aplicada. A metodologia apresenta limitações quanto ao uso de memória, mas encontra soluções com resultados próximos aos da literatura.

Keywords: Supervisory Control Theory; Optimization; Job Shop Scheduling; Blocking; Application;

Palavras-chaves: Teoria de Controle Supervisório, Otimização, Escalonamento *job shop*, Bloqueio, Aplicação.

1. INTRODUÇÃO

O problema do escalonamento de tarefas vem sendo cada vez mais estudado em virtude da sua relevância dentro do contexto industrial. Este problema consiste de um processo de tomada de decisão da alocação dos recursos para otimizar algum objetivo (Baker and Trietsch, 2013). Na formulação do problema, um conjunto de *jobs*, subdivididos em um conjunto de operações, devem ser processados pelas máquinas disponíveis. Diferentes configurações são encontradas em função de como os *jobs* (tarefas) são processados, como *flow shop*, *job shop*, *open shop* e outros (Pinedo, 2016).

O problema de escalonamento *job shop* (JSSP) define que cada *job* tem o seu próprio caminho entre as máquinas. A sua generalização é o problema de escalonamento *job shop* flexível (FJSSP), em que as operações podem ser processadas por mais de uma máquina. Nesse caso, cada *job* pode ter mais de um caminho (Pinedo, 2016). Brucker et al. (1994), Adams et al. (1988) e Van Laarhoven et al. (1992) são alguns trabalhos voltados para a solução de FJSSP, usando diferentes técnicas.

Este trabalho foca no problema do escalonamento *job shop* flexível com bloqueio, em que uma máquina só pode liberar a operação processada se a máquina subsequente estiver disponível. Caso contrário, a máquina permanece bloqueada. Esse cenário é encontrado em sistemas de manufatura em que não existem *buffers* intermediários entre as máquinas (Mascis and Pacciarelli, 2002).

Em geral, as relações de precedência e os tempos de processamento em cada máquina são apresentados em forma de tabela ou texto. As tabelas são mais adequadas para a interpretação visual dos dados, enquanto o formato textual é mais indicado para ser interpretado por um software. Cada conjunto de dados é, também, chamado de instância.

É comum utilizar a abstração de sistemas a eventos discretos (SED) para representar o comportamento de sistemas como os encontrados neste trabalho, em que o estado das máquinas se modifica em função da ocorrência de eventos discretos de início e de fim das operações. A teoria de linguagens e autômatos (Hopcroft et al., 2001) é usada neste trabalho.

A Teoria de Controle Supervisório (TCS) (Ramadge and Wonham, 1989) é utilizada juntamente com a representação de autômatos para restringir o comportamento em malha fechada do sistema de forma que ele seja não-bloqueante e minimamente restritivo. Um supervisor é obtido por meio do comportamento em malha aberta, o modelo das restrições do sistema e alguns procedimentos. Além disso, o supervisor contém todo o espaço de busca do problema.

O principal objetivo deste trabalho é gerar automaticamente os autômatos que modelam um problema de escalonamento *job shop* flexível com bloqueio a partir de instâncias em sua forma textual, como encontradas na literatura. Outros objetivos são: gerar o supervisor que modela o comportamento em malha fechada de cada problema e aplicar uma heurística de minimização de *makespan*. Assim, mostramos que é possível utilizar SED e TCS para resolver problemas práticos de outras áreas, nesse caso, a de Pesquisa Operacional.

Na próxima seção são apresentados os conceitos preliminares para o bom entendimento dos resultados. A seção 3 apresenta os procedimentos para a tradução automática do problema e algumas informações sobre os supervisores encontrados. A seção seguinte mostra os resultados de *makespan* quando aplicada a heurística sobre os supervisores mencionados. A seção 5 discute os resultados e limitações deste trabalho.

2. PRELIMINARES

Conceitos sobre o FJSSP com bloqueio são introduzidos na seção 2.1. Em seguida, são introduzidos os principais conceitos relacionados às linguagens e autômatos (seção 2.2) e à TCS (seção 2.3). Na seção 2.4 algumas considerações sobre o termo "bloqueio" são apresentadas.

2.1 O Escalonamento Job Shop Flexível com Bloqueio

O FJSSP é uma das variações dos problemas de escalonamento. As tarefas são chamadas de *jobs* e os recursos, de máquinas. Um conjunto de n *jobs* $\{J_i\}_{1\leq i\leq n}$, constituído de h operações $\{O_{ij}\}_{1\leq j\leq h}$, deve ser processado em um conjunto de m máquinas $\{M_k\}_{1\leq k\leq m}$ em um tempo de processamento p_{ijk}. Cada máquina pode processar apenas uma única operação por vez.

A Tabela 1 apresenta a atribuição dos tempos de processamento das operações em função das máquinas. Um valor $p \in \mathbb{R}_+$ é atribuído a uma operação se ela pode ser processada pela máquina, caso contrário, o tempo infinito (∞) é atribuído (Behnke and Geiger, 2012).

O FJSSP considera a existência de *buffers* ilimitados entre as máquinas, o que significa que uma máquina fica, imediatamente, disponível para receber o novo *job* assim que finaliza seu processamento. Essa premissa só é válida para fins de simulação, sendo um cenário mais realista o de *buffers* limitados ou inexistentes. Nesse último caso, a máquina faz o papel do *buffer* retendo o *job* até a máquina subsequente estar disponível para recebê-lo, ficando, então, bloqueada para processar o próximo *job*. Quando essa restrição é considerada, o problema é chamado de FJSSP com bloqueio (Mascis and Pacciarelli, 2002).

Tabela 1. Atribuição dos tempos de processamento em um FJSSP com flexibilidade total

$p_{i,j,k}$		M_1	M_k	$\cdots$	M_m
J_1	$O_{1,1}$	$p_{1,1,1}$	$p_{1,1,k}$	$\cdots$	$p_{1,1,m}$
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
	O_{1,h_1}	$p_{1,h_1,1}$	$p_{1,h_1,k}$	$\cdots$	$p_{1,h_1,m}$
J_i	$O_{i,1}$	$p_{i,1,1}$	$p_{i,1,k}$	$\cdots$	$p_{i,1,m}$
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
	O_{i,h_i}	$p_{i,h_i,1}$	$p_{i,h_i,k}$	$\cdots$	$p_{i,h_i,m}$
$\cdots$		$\cdots$	$\cdots$	$\cdots$	$\cdots$
J_n	$O_{n,1}$	$p_{n,1,1}$	$p_{n,1,k}$	$\cdots$	$p_{n,1,m}$
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
	O_{n,h_n}	$p_{n,h_n,1}$	$p_{n,h_n,k}$	$\cdots$	$p_{n,h_n,m}$

Dois casos de bloqueio são considerados: o bloqueio com troca (BWS) e bloqueio sem troca (BNS). Em um ciclo em que duas ou mais máquinas estão bloqueadas, uma aguardando o *job* da outra, o bloqueio com troca permite que cada operação seja admitida, simultaneamente, na máquina subsequente. Já o problema de bloqueio sem troca não permite esse movimento, caracterizando um *deadlock*.

Trabalhos como Pranzo and Pacciarelli (2016) e Louaqad and Kamach (2015) abordam a questão da infactibilidade de soluções FJSSP com bloqueio. Dependendo da forma como o algoritmo é implementado, a solução pode não ser factível. Nesse caso, uma alternativa é descartar a solução, desperdiçando recurso computacional. Outra alternativa é voltar alguns passos e buscar um novo caminho para a solução, o que pode gerar um aumento do custo computacional, pois não há a garantia de que o novo caminho escolhido resulta em uma solução factível, sendo necessário repetir o processo. Para contornar esse problema, outras técnicas podem ser utilizadas, como a escolhida neste trabalho.

2.2 Sistemas a Eventos Discretos

Sistemas a eventos discretos são sistemas dinâmicos regidos pela ocorrência de eventos, modificando seu estado de forma instantânea. Os eventos podem representar o acionamento de um botão ou de um sensor, a finalização de uma tarefa, entre outros.

Seja Σ o conjunto finito e não vazio de eventos, chamado alfabeto. Uma sequência finita de eventos de um alfabeto é chamada de cadeia. O conjunto formado por todas as cadeias possíveis, utilizando esse alfabeto Σ, é chamado de Fechamento Kleene Σ^*, incluindo a cadeia vazia ϵ. O comprimento de uma cadeia é definido por $|\sigma_1\sigma_2...\sigma_i| = i$, onde $i > 0$. O comprimento da cadeia vazia é $|\epsilon| = 0$.

A concatenação de duas cadeias é representada por su, onde $s, u, su \in \Sigma^*$. Uma cadeia s é chamada prefixo de $t \in \Sigma^*$, $s \leq t$, se $\exists u \in \Sigma^*$ tal que $su = t$. Define-se que uma linguagem é um subconjunto $L \subseteq \Sigma^*$ e o prefixo fechamento $\overline{L}$ de uma linguagem $L \subseteq \Sigma^*$ é o conjunto de todos os prefixos de cadeias em L, expresso por $\overline{L} = \{s \in \Sigma^* | s \leq t \text{ para algum } t \in L\}$.

Autômatos são grafos orientados cujos vértices são os estados e as arestas, as transições. Define-se um autômato finito determinístico como uma 5-tupla $G = (Q, \Sigma, \delta, q_0, Q_m)$, sendo Q o conjunto finito de estados, Σ, o alfabeto, $\delta : Q \times \Sigma \to Q$, a função de transição, $q_0 \in Q$, o estado inicial e $Q_m \subseteq Q$, o conjunto de estados

marcados. Um autômato é determinístico se $\delta(q,\sigma) = q_1$ e $\delta(q,\sigma) = q_2$ implica em $q_1 = q_2$.

A função de transição pode ser estendida para cadeias na forma $\delta(q,\sigma s) = q'$ se $\delta(q,\sigma) = x$ e $\delta(x,s) = q'$, em que $s \in \Sigma^*$ e $\sigma \in \Sigma$. A função de eventos ativos $\Gamma : Q \to 2^{\Sigma}$ é dada pelo conjunto de eventos $\sigma \in \Sigma$, em um dado estado q, onde $\delta(q,\sigma)$ está definido, ou seja, $\delta(q,\sigma)!$.

A linguagem gerada por um autômato é dada por $\mathcal{L}(G) = \{s \in \Sigma^* | \delta(q_0,s)!\}$ e representa o conjunto de cadeias de eventos que, partindo do estado inicial, levam a algum estado do autômato G. A linguagem marcada por um autômato é dada por $\mathcal{L}_m(G) = \{s \in \Sigma^* | s \in \mathcal{L}(G)$ e $\delta(q_0,s) \in Q_m\}$ e representa o conjunto de cadeias de eventos que, do estado inicial, levam a algum estado marcado $q \in Q_m$ de G.

Uma operação importante para representar o comportamento conjunto de autômatos é a composição paralela. Essa operação entre dois autômatos $G_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, Q_{m1})$ e $G_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, Q_{m2})$ é dada por $G_{12} = G_1 \parallel G_2 = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{12}, (q_{01}, q_{02}), Q_{m1} \times Q_{m2})$, onde

$$\delta_{12} = \delta((q_1,q_2),\sigma) = \begin{cases} (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma)), & \text{se } \sigma \in \Gamma_1(q_1) \cap \Gamma_2(q_2) \\ (\delta_1(q_1,\sigma), q_2), & \text{se } \sigma \in \Gamma_1(q_1) \backslash \Sigma_2 \\ (q_1, \delta_2(q_2,\sigma)), & \text{se } \sigma \in \Gamma_2(q_2) \backslash \Sigma_1 \\ \text{indefinido}, & \text{caso contrário} \end{cases}$$

onde $\Gamma(q)\backslash\Sigma$ indica a subtração de conjuntos.

2.3 Teoria de Controle Supervisório

A Teoria de Controle Supervisório (TCS) (Ramadge and Wonham, 1989) permite encontrar um supervisor que seja não-bloqueante e minimamente restritivo. Essa teoria leva a uma estrutura de controle que opera desabilitando eventos que levam ao comportamento não-seguro do sistema.

O sistema em malha aberta é chamado de planta e seu comportamento é encontrado a partir da composição paralela de todas as máquinas $G = \|G_i$, onde i é o índice da máquina. O conjunto de restrições, chamadas de especificações, é o resultado da composição paralela de cada especificação $E = \|E_i$.

Para essa modelagem, o alfabeto é particionado em dois subconjuntos disjuntos $\Sigma = \Sigma_c \cup \Sigma_u$, sendo o primeiro, Σ_c, dos eventos controláveis, e o segundo, Σ_u, dos eventos não controláveis, que não podem ser impedidos de ocorrer.

A composição paralela entre a planta e as especificações leva ao comportamento em malha fechada desejado $K = G \parallel E$, de forma que $\mathcal{L}_m(K) \subseteq \mathcal{L}_m(G)$. O autômato K é dito controlável em relação a G se $\overline{K}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{K}$.

A propriedade da controlabilidade estabelece que eventos não-controláveis não podem ser desabilitados. Se o supervisor não for controlável, a máxima linguagem controlável e não-bloqueante $Sup\mathcal{C}(K,G) \subseteq K$ deve ser encontrada. Isso é feito, iterativamente, eliminando todos os estados que desabilitam algum evento $\sigma \in \Sigma_u$ e verificando novamente a controlabilidade, até satisfazer a propriedade.

O supervisor obtido dessa forma contém todas as sequências que respeitam as restrições e pode ser usado como espaço de busca para a otimização, como no trabalho de Rafael and Pena (2018). Qualquer sequência escolhida do comportamento em malha fechada é factível, não havendo gasto computacional para tratar soluções infactíveis. Essa é, portanto, a principal vantagem do uso da TCS para modelar o FJSSP com bloqueio.

2.4 Consideração Sobre o Termo "Bloqueio"

O termo "bloqueio" tem significado diverso dependendo do contexto em que ele aparece.

Quando empregado no âmbito da TCS, o termo representa um cenário em que ocorre um *deadlock* ou um *livelock* em um sistema. O *deadlock* caracteriza uma situação em que o sistema não consegue evoluir e fica bloqueado em um estado não marcado. Esse bloqueio ocorre, pois nenhum dos processos é capaz de completar sua tarefa. O *livelock* ocorre quando o sistema fica bloqueado em um ciclo de ações que modificam o estado do sistema sem gerar nenhum avanço em prol da conclusão da tarefa.

Quando no âmbito dos problemas de escalonamento de tarefas, o bloqueio é uma característica do sistema, pois o *job* tem que permanecer na máquina, bloqueando-a, até que a máquina subsequente possa recebê-lo. Embora um problema de escalonamento possibilite a ocorrência de *deadlocks*, deve ficar clara a diferença do uso desse termo em cada contexto.

3. TRADUÇÃO AUTOMÁTICA

Um conjunto de problemas da literatura foi escolhido para realizar a tradução automática e estão apresentados na Tabela 2. A modelagem foi feita considerando que a troca não é permitida (BNS).

Tabela 2. Instâncias de FJSSP.

Instâncias	Jobs	Máquinas	Referência
mt06, 10, 20	6..20	5..10	Muth et al. (1963)
mk01..10	10..20	4..15	Brandimarte (1993)
01..18a	10..20	5..10	Dauzère-Pérès and Paulli (1997)
la01..40	10..30	5..15	Lawrence (1984)
orb1..10	10	10	Applegate and Cook (1991)
mt10_, setb4_, seti5_	10..15	11..18	Chambers and Barnes (1996)
abz5..9	10..20	10..15	Adams et al. (1988)
car1..8	7..14	4..9	Carlier (1978)
hurink s,e,r,v data	6..30	4..15	Hurink et al. (1994)

3.1 Interpretação das Instâncias

As informações relacionadas à alocação nas máquinas e os tempos de processamento das operações são apresentadas de duas formas: 1) em formato de tabela como apresentado na Tabela 1; 2) em formato textual como na Figura 1.

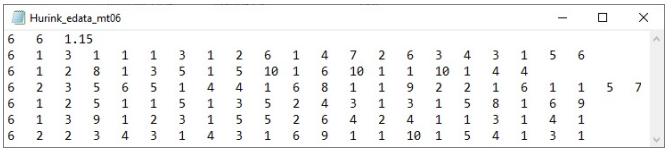


Figura 1. Exemplo de instância.

A primeira linha do arquivo de texto tem os campos: 1) o número de *jobs* n; 2) o número de máquinas m; 3)

(opcional) a média do número de máquinas que processam cada operação, indicando a flexibilidade do problema.

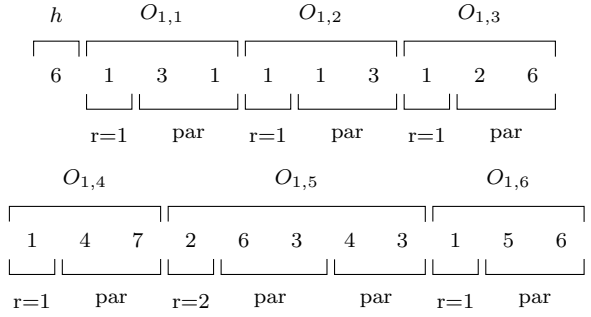
No exemplo da Figura 1 os valores da primeira linha são:

6 6 1.15

o que representa um problema com 6 *jobs*, 6 máquinas e uma média de 1,15 máquinas por operação.

A primeira linha é seguida de n linhas que contêm, cada uma, os dados de um *job*. O primeiro valor da linha indica a quantidade h de operações daquele *job*, seguido de h conjuntos de dados referente a cada operação: o primeiro campo indica a quantidade r de máquinas que podem processar aquela operação, que é acompanhado de r duplas (máquina, tempo de processamento).

A segunda linha da instância contém as informações do *job* J_1, conforme apresentado a seguir:



Nesse *job*, a primeira operação ($O_{1,1}$) é executada na máquina 3 e tem o tempo de processamento de 1 unidade. A segunda operação ($O_{1,2}$) é executada na máquina 1 e com tempo de processamento de 3 unidades. Somente a operação $O_{1,5}$ pode ser processada por duas máquinas, 4 e 6, ambas com tempo de processamento de 3 unidades. Os dados dessa linha levam à Tabela 3.

Tabela 3. Atribuição dos tempos de processamento de um *job*.

$p_{i,j}$		M_1	M_2	M_3	M_4	M_5	M_6
J_1	$O_{1,1}$	∞	∞	1	∞	∞	∞
	$O_{1,2}$	3	∞	∞	∞	∞	∞
	$O_{1,3}$	∞	6	∞	∞	∞	∞
	$O_{1,4}$	∞	∞	∞	7	∞	∞
	$O_{1,5}$	∞	∞	∞	3	∞	3
	$O_{1,6}$	∞	∞	∞	∞	6	∞

3.2 FJSSP com Bloqueio Modelado como Autômato

Neste trabalho propõe-se o desenvolvimento de um método de tradução automática de FJSSP com bloqueio em autômatos. A modelagem do *job* é dada conforme a seguir:

- cada *job* é representado por um autômato;
- cada operação é composta por um par de eventos;
  - · eventos iniciados com a letra a são controláveis e representam o início de uma operação;
  - · eventos iniciados com a letra b são não-controláveis e representam o fim de uma operação;
  - · o índice de cada evento é dado por $ijkl$, onde:
    - i se refere ao *job*;
    - j se refere à operação;
    - k se refere à máquina;
    - l se refere à máquina anterior ($l = 0$ se o evento é relativo à primeira operação);
- a linguagem marcada desse autômato corresponde à sequência das operações do *job*.
- os estados são nomeados como $sd.x$, onde:
  - · d é a profundidade no autômato;
  - · x é um número natural para diferenciar estados de mesma profundidade.

A Figura 2 mostra o autômato do *job*. No exemplo, o *job* tem 6 operações e somente uma delas pode ser processada por mais de uma máquina. Nota-se que no estado $s8.0$ há dois destinos possíveis: o estado $s9.1$ com a execução do evento $a1564$ (*job* 1, 5ª operação, na máquina 6, máquina anterior foi a 4) ou o estado $s9.0$ com o evento $a1544$ (*job* 1, 5ª operação, na máquina 4, máquina anterior foi a 4).

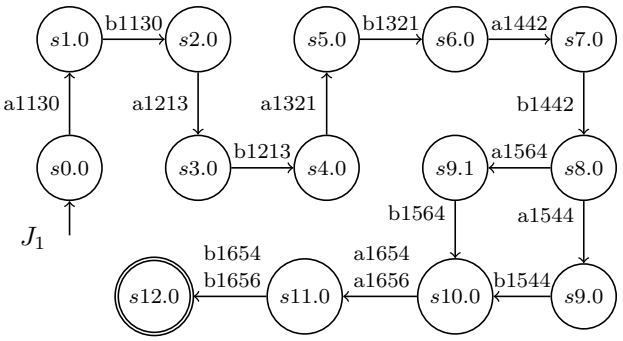


Figura 2. Exemplo de *job* modelado como autômato.

A modelagem das máquinas para o FJSSP sem bloqueio pode ser feita com apenas dois estados: um para a máquina disponível ($s0$) e outro para a máquina trabalhando ($s1$). Com o bloqueio, um estado extra é necessário ($s2$), para a máquina ociosa aguardando ser liberada. A modelagem é apresentada na Figura 3.

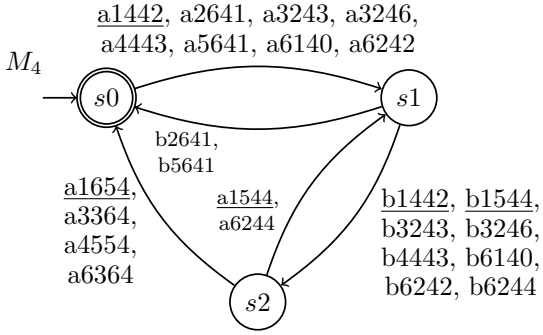


Figura 3. Modelo da máquina M_4 da instância da Figura 1.

Uma máquina no estado inicial $s0$ está disponível. Ao iniciar o processamento de um *job*, vai para o estado $s1$ e, ao finalizá-lo, vai para o estado $s2$. Se a operação subsequente é admitida pela mesma máquina, retorna ao estado $s1$, mas se a operação subsequente é admitida em outra máquina, retorna-se para o estado $s0$. Um caso particular deve ser considerado, quando a máquina processa a última operação de um *job*. Nesse caso, ela retorna diretamente para o estado $s0$, pois o *job* foi totalmente processado, liberando a máquina. Isso é mostrado na Figura 3, na qual a máquina 4 da instância da Figura 1 é modelada.

O nome dos eventos são os mesmos daqueles criados para a modelagem dos *jobs*. Para ilustrar a modelagem

das máquinas são apresentados os eventos da máquina 4 considerando o *job* 1 (Tabela 3). A máquina 4 realiza a operação 4 (a_{1442} e b_{1442}) e a operação 5 (a_{1544} e b_{1544}) do *job* 1. No estado inicial, a máquina M_4 pode fazer o evento a_{1442} que a leva para o estado s_1, do qual sai com o fim dessa operação (b_{1442}) indo para o estado s_2. No caso do *job* 1, a operação seguinte é feita, também, pela máquina 4, de forma que o evento a_{1544} leva de volta para o estado s_1 e, em seguida, para o estado s_2 com o evento b_{1544}. No estado s_2 (máquina está ociosa, porém bloqueada aguardando a próxima máquina admitir o *job*), a máquina só é liberada após o evento a_{1654}. O mesmo processo é feito para cada *job*. Os eventos mencionados acima estão sublinhados para facilitar a observação.

Geralmente, na TCS, o conjunto das máquinas é considerado a planta, conforme apresentado na Subseção 2.3. No caso desse trabalho, no entanto, as máquinas são a especificação, pois, em seus modelos, contêm restrições para execução dos *jobs*.

3.3 Obtenção do Supervisor para FJSSP com Bloqueio

Essa etapa foi feita em um computador com sistema operacional Windows 10, 64 GB de memória RAM e processador Intel Xenon E5-2650/2.00 GHz. Os procedimentos apresentados para a modelagem dos *jobs* e máquinas foram implementados em um programa, apresentados no Algoritmo 1 e os autômatos foram gerados utilizando uma função da biblioteca UltraDES (Alves et al., 2017) (linhas 15 e 17). O supervisor monolítico foi obtido, por outra função da biblioteca, a partir das plantas (modelo dos *jobs*) e das especificações (modelo das máquinas) (linha 18).

Algoritmo 1: Geração do supervisor a partir de uma instância de FJSSP

```
   Entrada: instância_FJSSP
   Saída: supervisor
1  n ← número de jobs
2  m ← número de máquinas
3
4  para cada job_n faça
5      estado ← 0
6      para cada operação_h faça
7          x ← 0
8          para cada máquina_m faça
9              adiciona [estado.0 → a_ijkl → (estado + 1).x] ao job_n
10             adiciona [(estado + 1).x → b_ijkl → (estado + 2).0] ao job_n
11             x ← x + +
12             adiciona [0 → a_ijkl → 1] à maq_m
13             adiciona [1 → b_ijkl → 0] à maq_m
14         estado ← estado + 2
15     G_set ← gera autômato job_n
16 para cada maq_m faça
17     E_set ← gera autômato maq_m
18 supervisor ← gera_supervisor_monolitico (G_set, E_set)
```

A implementação do algoritmo está disponível no sítio eletrônico *github.com/lacsed/FJSSPtranslationToAutomata*.

O número de estados e de transições de cada supervisor, das instâncias para as quais foi possível encontrar uma solução, são apresentados na Tabela 4. O valor $|M_{j,k}|$ indica a média de máquinas por operação (flexibilidade).

Embora o tamanho dos supervisores encontrados seja relativamente grande, eles contêm todo o universo de busca. Significa que qualquer caminho do estado inicial até o estado marcado representa uma sequência factível, livre de bloqueios e que atende às restrições do problema.

Tabela 4. Tamanho dos supervisores obtidos

Instância	Dimensão	$\overline{\lvert M_{j,k}\rvert}$	Estados	Transições
Hurink_sdata_mt06	6x6	1.00	346.328	1.085.803
Hurink_edata_mt06	6x6	1.15	761.494	2.782.216
Hurink_rdata_mt06	6x6	2.00	36.923.146	234.222.896
Hurink_vdata_mt06	6x6	3.00	110.097.999	1.128.024.299
Hurink_sdata_car1	11x5	1.00	121.529.344	394.521.601
Hurink_sdata_car2	13x4	1.00	111.562.752	314.163.201
Hurink_sdata_car3	12x5	1.00	383.021.056	1.253.793.793
Hurink_sdata_car4	14x4	1.00	300.498.944	850.755.585
Hurink_sdata_car5	10x6	1.00	249.672.704	915.118.081
Hurink_sdata_car7	7x7	1.00	10.787.520	40.741.569
Hurink_edata_car7	7x7	1.15	20.944.494	90.622.826
Hurink_sdata_car8	8x8	1.00	235.018.496	1.007.106.049
Hurink_sdata_la01	10x5	1.00	9.709.952	29.609.025
Hurink_edata_la01	10x5	1.15	44.678.169	154.143.493
Hurink_sdata_la02	10x5	1.00	11.018.176	33.899.137
Hurink_edata_la02	10x5	1.15	38.742.965	133.536.014
Hurink_sdata_la03	10x5	1.00	7.694.720	23.422.209
Hurink_edata_la03	10x5	1.15	30.685.423	105.835.602
Hurink_sdata_la04	10x5	1.00	8.495.232	25.624.577
Hurink_edata_la04	10x5	1.15	23.228.017	79.958.694
Hurink_sdata_la05	10x5	1.00	12.766.336	39.199.041
Hurink_edata_la05	10x5	1.15	46.370.480	162.824.604

Para algumas instâncias não foi possível computar o supervisor por falta de memória no computador em que foram processadas. Isso se dá pelo problema da explosão do espaço de estados que pode ser observado, por exemplo, comparando as instâncias "Hurink_sdata_la01" e "Hurink_sdata_car1" em que o aumento de 10% no número de *jobs* provoca um aumento no número de estados de $\approx$ 1.150%. Além disso, um pequeno aumento na flexibilidade leva a um significativo aumento no tamanho do problema, como pode ser observado de "Hurink_edata_mt06" para "Hurink_rdata_mt06".

4. HEURÍSTICA PARA O FJSSP COM BLOQUEIO

Com o espaço de estados definido para algumas instâncias, buscou-se computar uma solução para elas. Uma heurística baseada na Minimização de Tempo Heurístico (MTH), de Alves et al. (2019) foi escolhida, sendo necessárias modificações para adequar ao problema tratado neste trabalho. A heurística prioriza os eventos controláveis em detrimento dos não-controláveis, reduzindo o *branch-factor*. Outro procedimento realizado é que, ao chegar a um estado já visitado, verifica-se aquele que tem o menor tempo, excluindo o caminho de maior tempo.

Na Tabela 5 são apresentados os resultados da aplicação dessa heurística. As duas primeiras colunas identificam a instância. A terceira e a quarta colunas indicam o tamanho do supervisor. As duas colunas seguintes apresentam o tempo de processamento e o valor de *makespan* encontrado ao se aplicar a heurística MTH sobre o supervisor. As duas últimas colunas apresentam valores de *makespan* encontrados na literatura. As células vazias, indicam a inexistência de resultados conhecidos para as respectivas instâncias. Para as instâncias marcadas com * foi utilizado um computador com 256 GB de memória RAM.

Os resultados do MTH foram melhores que os de Mascis and Pacciarelli (2002) e piores que os de Pranzo and Pacciarelli (2016) (melhor solução entre 10 rodadas de 60 segundos de um algoritmo *Iterated Greedy*). Não foi possível comparar problemas maiores, como (10×10), pois a metodologia proposta não foi capaz de computar tais soluções.

O tempo de execução da heurística variou de alguns segundos para instâncias de até 41 milhões de estados,

Tabela 5. Resultados de *makespan* para instâncias de FJSSP com bloqueio

Instância	Dimensão	Estados	Transições	Tempo de execução (s)	Makespan		
					HTM	Pranzo & Pacciarelli (2016)	Mascis & Pacciarelli (2002)
Hurink_sdata_mt06	6x6	346.328	1.085.803	0,11	**69**		74
Hurink_edata_mt06	6x6	761.494	2.782.216	0,04	**68**		
Hurink_rdata_mt06	6x6	36.923.146	234.222.896	24,67	**57**		
Hurink_sdata_car1*	11x5	121.529.344	394.521.601	7.654,70	**7.409**		
Hurink_sdata_car2*	13x4	111.562.752	314.163.201	22.934,44	**7.503**		
Hurink_sdata_car3	12x5	383.021.056	1.253.793.793	Solução não encontrada			
Hurink_sdata_car5*	10x6	249.672.704	915.118.081	1.513,88	**8.218**		
Hurink_sdata_car7	7x7	10.787.520	40.741.569	6,29	**6.788**		
Hurink_sdata_car8*	8x8	235.018.496	1.007.106.049	86,85	**8.585**		
Hurink_sdata_la01	10x5	9.709.952	29.609.025	0,59	965	**881**	1066
Hurink_sdata_la02	10x5	11.018.176	33.899.137	1,16	944	**900**	1077
Hurink_sdata_la03	10x5	7.694.720	23.422.209	2,19	821	**808**	884
Hurink_sdata_la04	10x5	8.495.232	25.624.577	0,50	889	**862**	881
Hurink_sdata_la05	10x5	12.766.336	39.199.041	0,53	803	**742**	995
Hurink_sdata_la06	15x5	Supervisor não encontrado					

até algumas horas para instâncias de cerca de 400 milhões de estados. Isso se justifica pela quantidade de caminhos, existentes no supervisor, que devem ser avaliados.

5. CONCLUSÃO

Este trabalho mostrou que é possível utilizar a TCS para representar FJSSP com bloqueio. Um algoritmo para realizar a tradução automática dessa classe de problemas para autômatos, usando a TCS, foi desenvolvido e os supervisores para algumas instâncias da literatura gerados.

Todo o espaço de busca está contido no supervisor e outras heurísticas podem ser aplicadas sobre ele, com a vantagem de que nenhuma reavaliação da solução é necessária. O modelo proposto pode ser estendido para outras classes dos problemas de escalonamento. Para ilustrar o uso do espaço de busca, foi aplicada uma heurística sobre o supervisor e foi possível encontrar soluções otimizadas para os problemas, próximas de outros trabalhos da literatura.

São propostas de trabalhos futuros a implementação de outras heurísticas que possam levar a resultados melhores e a busca de formas mais eficientes de armazenar as estruturas de estados e transições, a fim de reduzir o consumo de memória e expandir os limites de computação.

REFERÊNCIAS

Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3), 391–401.

Alves, L.V., Pena, P.N., and Takahashi, R.H. (2019). Planejamento da Produção em Sistemas a Eventos Discretos usando Heurística. *XIV Simpósio Brasileiro de Automação Inteligente (SBAI)*, 1733–1738.

Alves, L.V., Martins, L.R., and Pena, P.N. (2017). Ultrades-a library for modeling, analysis and control of discrete event systems. *IFAC*, 50(1), 5831–5836.

Applegate, D. and Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, 3(2), 149–156.

Baker, K.R. and Trietsch, D. (2013). *Principles of sequencing and scheduling*. John Wiley & Sons.

Behnke, D. and Geiger, M.J. (2012). Test instances for the flexible job shop scheduling problem with work centers. *Logistics Management Department, Hamburg,Germany*.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research*, 41(3), 157–183.

Brucker, P., Jurisch, B., and Sievers, B. (1994). A branch and bound algorithm for the job-shop scheduling problem. *Discrete applied mathematics*, 49(1-3), 107–127.

Carlier, J. (1978). Ordonnancements a contraintes disjonctives. *RAIRO-Operations Research*, 12(4), 333–350.

Chambers, J.B. and Barnes, J.W. (1996). Tabu search for the flexible-routing job shop problem. *Graduate program in Operations Research and Industrial Engineering, The University of Texas at Austin, Technical Report Series*.

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

Hopcroft, J.E., Motwani, R., and Ullman, J.D. (2001). Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1), 60–65.

Hurink, J., Jurisch, B., and Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum*, 15, 205–215.

Lawrence, S. (1984). An experimental investigation of heuristic scheduling techniques. *Supplement to resource constrained project scheduling*.

Louaqad, S. and Kamach, O. (2015). Scheduling of blocking and no wait job shop problems in robotic cells.

Mascis, A. and Pacciarelli, D. (2002). Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3), 498–517.

Muth, J.F., Thompson, G.L., and Winters, P.R. (1963). *Industrial scheduling*. Prentice-Hall.

Pinedo, M.L. (2016). *Scheduling: theory, algorithms, and systems*. Springer, New York, New York, 5ª edition.

Pranzo, M. and Pacciarelli, D. (2016). An iterated greedy metaheuristic for the blocking job shop scheduling problem. *Journal of Heuristics*, 22(4), 587–611.

Rafael, G.C. and Pena, P.N. (2018). Using an abstraction of the supervisor to solve a planning problem in manufacturing systems. In *Anais do XXII Congresso Brasileiro de Automática, CBA*.

Ramadge, P.J. and Wonham, W.M. (1989). The control of discrete event systems. *Proceedings of IEEE*, 77, 81–98.

Van Laarhoven, P.J., Aarts, E.H., and Lenstra, J.K. (1992). Job shop scheduling by simulated annealing. *Operations research*, 40(1), 113–125.