

DISSERTAÇÃO DE MESTRADO Nº 1187

**TRADUÇÃO AUTOMÁTICA DE PROBLEMAS DE ESCALONAMENTO JOB
SHOP FLEXÍVEL COM BLOQUEIO PARA AUTÔMATOS UTILIZANDO A
TEORIA DE CONTROLE SUPERVISÓRIO**

Daniel Sarsur Câmara

DATA DA DEFESA: 04/03/2020

Universidade Federal de Minas Gerais
Programa de Pós-Graduação em Engenharia Elétrica
Laboratório de Análise e Controle de Sistemas a Eventos Discretos

**TRADUÇÃO AUTOMÁTICA DE PROBLEMAS DE
ESCALONAMENTO JOB SHOP FLEXÍVEL COM
BLOQUEIO PARA AUTÔMATOS UTILIZANDO A
TEORIA DE CONTROLE SUPERVISÓRIO**

Daniel Sarsur Câmara

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do Título de Mestre em Engenharia Elétrica.

Orientadora: Prof^ª. Patrícia Nascimento Pena
Co-Orientador: Prof. Ricardo Hiroshi Caldeira Takahashi

Belo Horizonte
Março de 2020

C172t	<p>Câmara, Daniel Sarsur. Tradução automática de problemas de escalonamento job shop flexível com bloqueio para autômatos utilizando a teoria de controle supervísório [recurso eletrônico] / Daniel Sarsur Câmara. - 2020. 1 recurso online (80 f. : il., color.) : pdf.</p> <p>Orientadora: Patrícia Nascimento Pena. Coorientador: Ricardo Hiroshi Caldeira Takahashi.</p> <p>Dissertação (mestrado) - Universidade Federal de Minas Gerais, Escola de Engenharia.</p> <p>Anexos e apêndices: f. 55-80.</p> <p>Bibliografia: f. 51-54. Exigências do sistema: Adobe Acrobat Reader.</p> <p>1. Engenharia elétrica - Teses. 2. Otimização - Teses. I. Pena, Patrícia Nascimento. II. Takahashi, Ricardo Hiroshi Caldeira. III. Universidade Federal de Minas Gerais. Escola de Engenharia. IV. Título.</p>
	CDU: 621.3(043)

"Tradução Automática de Problemas de Escalonamento Job Shop Flexível Com Bloqueio Para Autômatos Utilizando A Teoria de Controle Supervisório"

Daniel Sarsur Câmara

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Mestre em Engenharia Elétrica.

Aprovada em 04 de março de 2020.

Por:

Patricia

Prof. Dr. Patrícia Nascimento Pena
DELT (UFMG)

Ricardo

Prof. Dr. Ricardo Hiroshi Caldeira Takahashi
DMAT (UFMG)

Martin

Prof. Dr. Martín Gómez Ravetti
DEP (UFMG)

Vinicius

Prof. Dr. Vinicius Mariano Gonçalves
DEE (UFMG)



Agradecimentos

Agradeço à Patrícia pela sua orientação, antes mesmo de eu ter um tema definido, até a conclusão deste trabalho. Agradeço às inúmeras correções, sempre me incentivando a expressar melhor aquilo que estava na minha cabeça e ao apoio nos momentos em que os resultados demoravam a aparecer.

Agradeço igualmente ao Takahashi pela sua orientação e pelas ponderações e ideias sempre pertinentes ao longo de todo o desenvolvimento da pesquisa. Agradeço por me mostrar que cada passo dado era sempre mais interessante do que parecia ser.

Agradeço à Denize e ao Mauro por me darem todas as condições necessárias para ter chegado até aqui.

Agradeço à Érica pela escuta e incentivo, mesmo que por um longo período fisicamente distante.

Agradeço à Juliana pelo seu otimismo, paciência e companheirismo.

Agradeço aos meus colegas do LACSED pela convivência e oportunidade de auxílio mútuo.

“Talvez a imobilidade das coisas ao nosso redor lhes seja imposta pela nossa certeza de que tais coisas são elas mesmas e não outras, pela imobilidade do nosso pensamento em relação a elas.”
(Marcel Proust)

Resumo

Este trabalho é um estudo exploratório que tem como objetivo desenvolver um algoritmo para converter automaticamente um problema de escalonamento *job shop* flexível com bloqueio em autômatos usando a Teoria de Controle Supervisório. São apresentados o mecanismo para interpretação dos dados de problemas da literatura e a metodologia para a geração dos autômatos que representam o comportamento e as restrições do problema. Além disso, é encontrado o comportamento em malha fechada de alguns sistemas, sobre o qual é aplicada uma heurística de minimização de *makespan*. Os resultados encontrados estão próximos aos da literatura. É, também, desenvolvido um algoritmo para a obtenção de cadeias de eventos que minimizem os tempos de produção em um sistema flexível de manufatura com demandas estocásticas.

Palavras-chaves: Sistemas a Eventos Discretos, Teoria de Controle Supervisório, Escalonamento *job shop*, Bloqueio, Otimização.

Abstract

This work is an exploratory study that aims to develop an algorithm to automatically translate a Blocking Flexible Job Shop Scheduling Problem modeling into automata using the Supervisory Control Theory. The mechanism for interpretation of literature problem data and the methodology for generating automata that represent the behavior and constraints of the problem are presented. In addition, the closed-loop behavior of some systems is found, on which a makespan minimization heuristic is applied. The results found are close to those in the literature. An algorithm for obtaining strings of events that minimize production times in a flexible manufacturing system with stochastic demands is also developed.

Key-words: Discrete Event Systems, Supervisory Control Theory, Job Shop Scheduling, Blocking, Optimization.

Lista de Figuras

2.1	Exemplo de grafo disjuntivo. Fonte: Blażewicz et al. (2000)	6
3.1	Diagrama de um problema de escalonamento <i>job shop</i>	12
3.2	Diagrama de um problema de escalonamento <i>job shop</i> flexível	13
3.3	Exemplo de um autômato determinístico	15
3.4	Operação parte acessível $Ac(G)$	16
3.5	Operação parte coacessível $CoAc(G)$	16
3.6	Operação $Trim(G)$	17
3.7	Funcionamento do supervisor	18
3.8	Produtos do Sistema Flexível de Manufatura	20
3.9	Diagrama do Sistema Flexível de Manufatura	20
3.10	Modelagem das máquinas do Sistema Flexível de Manufatura.	21
3.11	Especificações referentes ao <i>overflow</i> e <i>underflow</i> dos <i>buffers</i> do Sistema Flexível de Manufatura.	21
3.12	Diagrama de simulação de Sistemas a Eventos Discretos	23
4.1	Exemplo de instância	27
4.2	Exemplo da modelagem de um <i>job</i> em autômato	29
4.3	Exemplo da modelagem de uma especificação	30
4.4	Exemplo da modelagem de uma máquina em autômato	31
4.5	Autômatos das máquinas para o exemplo proposto	32
4.6	Autômatos das máquinas para o exemplo proposto com os eventos renomeados	34
4.7	Exemplo da modelagem de um <i>job</i> em autômato com os eventos renomeados	35
4.8	Instância para o problema do SFM	35
4.9	Exemplo da modelagem do job_1 do SFM	35
4.10	Exemplo da modelagem de M_1 do SFM	36
4.11	Diagrama de Gantt da solução do problema Hurink_edata_mt06 com bloqueio	40
4.12	Diagrama de Gantt da solução do problema Hurink_edata_mt06 sem bloqueio	40
5.1	Diagrama ilustrativo do cenário abordado	41
5.2	Valores de <i>makespan</i> das soluções encontradas	46
5.3	Tempo de execução do algoritmo para cada solução encontrada.	47

Lista de Tabelas

2.1	Instâncias de FJSSP	9
3.1	Atribuição dos tempos de processamento em um problema clássico de JSSP	12
3.2	Atribuição dos tempos de processamento em um problema de FJSSP com flexibilidade total	13
3.3	Intervalo de tempo entre os pares de eventos.	22
4.1	Atribuição dos tempos de processamento de um <i>job</i> conforme exemplo . . .	28
4.2	Progressão dos estados em função dos eventos da sequência (a)	33
4.3	Progressão dos estados em função dos eventos da sequência (b)	33
4.4	Tamanho dos supervisores obtidos	37
4.5	Resultados de <i>makespan</i> para instâncias de FJSSP com bloqueio	38
5.1	Cálculo de r_1	45
5.2	Cálculo de r_r	45

Lista de Algoritmos

1	Minimização de Tempo Heurístico	25
2	Geração do supervisor a partir de uma instância de FJSSP	37
3	Minimização de <i>makespan</i> da produção total e da nova demanda	43

Lista de Siglas e Abreviaturas

B&B	Branch and Bound
BNW	Blocking No-Swap
BWS	Blocking With Swap
FJSSP	Flexible Job Shop Scheduling Problem
GA	Genetic Algorithm
IDE	Integrated Development Environment
IG	Iterated Greedy
JSSP	Job Shop Scheduling Problem
MTH	Minimização de Tempo Heurístico
RAM	Random Access Memory
SA	Simulated Annealing
SB	Shifting Bottleneck
SED	Sistemas a Eventos Discretos
SFM	Sistema Flexível de Manufatura
TA	Timed Automata
TCS	Teoria de Controle Supervisório
TS	Tabu Search
TWS	Time-Weighted Systems

Lista de Símbolos

Σ	alfabeto
\approx	aproximadamente
s^*	cadeia otimizada
K	comportamento em malha fechada
$ s $	comprimento da cadeia s
Q	conjunto de estados
Q_m	conjunto de estados marcados
Σ_c	conjunto dos eventos controláveis
Σ_{uc}	conjunto dos eventos não-controláveis
\mathbb{R}_+	conjunto dos números reais não-negativos
\emptyset	conjunto vazio
\subseteq	contido ou igual
q_0	estado inicial
σ	evento
\exists	existe
Σ^*	fechamento Kleene
Γ	função de eventos ativos
δ	função de transição
$=$	igual
∞	infinito
\cap	interseção
$\mathcal{L}(G)$	linguagem gerada de G
$\mathcal{L}_m(G)$	linguagem marcada de G
C_{max}	makespan
$<$	menor

\leq	menor ou igual
n	número de <i>jobs</i>
m	número de máquinas
h	número de operações
$SupC(K, G)$	operação que calcula a máxima sublinguagem controlável de K em relação a G
\parallel	operador de composição paralela
\forall	para todo
$Ac(G)$	parte acessível de G
$Trim(G)$	parte acessível e coacessível de G
$CoAc(G)$	parte coacessível de G
\in	pertence
$\%$	por cento
\bar{L}	prefixo fechamento de L
\times	produto cartesiano
P^{-1}	projeção inversa
P	projeção natural
\setminus	subtração de conjuntos
p	tempo de processamento
\cup	união

Sumário

1	Introdução	1
1.1	Objetivos	2
1.2	Estrutura da Dissertação	3
2	Revisão da Literatura	5
2.1	O Problema do Escalonamento de Tarefas	5
2.2	Técnicas para Solução de FJSSP	5
2.3	Trabalhos de Otimização Utilizando Autômatos	8
2.4	Instâncias de FJSSP	8
3	Preliminares	11
3.1	Problema de Escalonamento de Tarefas	11
3.1.1	Job Shop Scheduling Problem - JSSP	11
3.1.2	Flexible Job Shop Scheduling Problem - FJSSP	12
3.1.3	Job Shop Scheduling Problem com bloqueio	13
3.2	Sistemas a Eventos Discretos	14
3.2.1	Linguagens	14
3.2.2	Autômatos	15
3.3	Teoria de Controle Supervisório	18
3.4	Consideração Sobre o Termo “bloqueio”	19
3.5	Ferramentas auxiliares	19
3.5.1	Sistema Flexível de Manufatura	19
3.5.2	Simulação de Sistemas a Eventos Discretos	21
3.5.3	UltraDES	23
3.5.4	Minimização de Tempo Heurístico	24
3.5.5	Descrição do Computador Utilizado	26
4	Tradução Automática do Problema	27
4.1	Interpretação das Instâncias	27
4.2	FJSSP Modelado como Autômato	28
4.3	FJSSP com Bloqueio Modelado como Autômato	31
4.3.1	Modelagem do SFM	33
4.4	Obtenção do Supervisor para o Problema de FJSSP com Bloqueio	35
4.5	Solução do Problema de FJSSP com uma Heurística	36
4.5.1	Resultados	38

5	Escalonamento de Tarefas com Demandas Estocásticas	41
5.1	Contextualização	41
5.1.1	Definição do Problema	42
5.2	Solução Proposta	42
5.3	Estudo de Caso	45
5.3.1	Simulação	45
6	Conclusão	49
	Referências Bibliográficas	51
	Apêndices	55
A	Modelagem do Problema Hurink_edata_mt06	57
B	Modelagem do Problema Hurink_edata_mt06 com Bloqueio	61
C	Modelagem do Sistema Flexível de Manufatura	65
	Anexos	69
A	Limites Superior e Inferior para Instâncias de FJSSP	71

Capítulo 1

Introdução

O problema do escalonamento de tarefas, ou *scheduling*, vem sendo amplamente estudado por diversas áreas do conhecimento, em virtude do seu relevante impacto, principalmente dentro do contexto industrial. Isso se deve à evolução e maior organização dos sistemas de planejamento e controle de produção ao longo do tempo, constantemente impulsionada pelas demandas das empresas em reduzir os tempos de entrega, a quantidade de material armazenado, a emissão de poluentes no meio ambiente, entre outros (Lustosa et al., 2008).

Outro fator que incentiva essa evolução é a mudança dos padrões de produção ao longo do tempo. Tal mudança passa da produção em massa, para atender à grande demanda de consumo, para a customização em massa, a fim de oferecer maior variedade de produtos, até chegarmos à personalização, que busca atender às demandas individuais de cada consumidor (Hu, 2013). Observando pelo viés dos produtores, isso representa um crescimento significativo do número de combinações possíveis para um mesmo tipo de produto.

Nesse sentido, o escalonamento de tarefas se apresenta como um processo de tomada de decisão da alocação dos recursos disponíveis e necessários para melhorar determinado aspecto da produção. Tais aspectos podem estar relacionados com a redução de *makespan* (tempo para produção), o aumento de *throughput* (taxa de produção), a redução de perdas ao longo do processo, a redução de insumos ou, de maneira geral, o aumento de produtividade (Petrônio & Laugeni, 2005).

Na formulação dos problemas clássicos de escalonamento de tarefas, um determinado número de *jobs* (tarefas) é subdividido em operações que devem ser alocadas nas máquinas disponíveis a fim de minimizar alguma função objetivo. As diferentes configurações que esses problemas podem ter modelam os variados ambientes encontrados na indústria e estão relacionadas à forma como as máquinas processam os *jobs*, tais como máquinas paralelas não relacionadas, *flow shop*, *job shop*, *open shop*, entre outros.

O problema de escalonamento *job shop* (JSSP), em particular, é definido por cada *job* ter uma sequência definida para o processamento de suas operações, ou seja, cada operação pode ser processada em uma máquina apenas. Uma generalização para o JSSP é o problema de escalonamento de tarefas *job shop* flexível (FJSSP) em que uma operação pode ser processada por mais de uma máquina. Nesse caso, um *job* passa a ter mais de uma rota para sua produção, mantendo-se a relação de precedência entre as operações (Pinedo, 2016).

Entre os vários tipos de restrições e características que o FJSSP pode ter, o bloqueio é o que receberá maior enfoque neste trabalho. Nos problemas em que o bloqueio não é considerado,

assume-se que uma máquina, ao finalizar uma tarefa, está disponível para receber outra tarefa. Em outras palavras, considera-se que os *buffers* entre as máquinas têm capacidade ilimitada. Nos problemas com o bloqueio, no entanto, os *buffers* têm capacidade finita ou são inexistentes, o que implica o fato de que uma máquina, após finalizar a sua tarefa, só pode ser liberada caso a máquina subsequente esteja disponível (Mascis & Pacciarelli, 2002).

As informações relativas à precedência entre as operações e os tempos de processamento em cada máquina são, geralmente, apresentados em formato de tabela ou de texto. As tabelas favorecem a interpretação visual dos dados enquanto a forma textual é mais indicada para a interpretação por programas de computador. Cada problema com seu conjunto de dados é também chamado de instância.

No contexto industrial, é comum utilizar a abstração de um sistema a eventos discretos (SED) para modelar o comportamento das máquinas e sistemas. Isso porque, de forma geral, o estado em que eles se encontram se modifica em decorrência de um evento, como o acionamento de um botão ou o sinal de um sensor. Nesse trabalho será utilizada a teoria de linguagens e autômatos (Hopcroft et al., 2001) como a representação de SED.

Junto a essa representação será utilizada a Teoria de Controle Supervisório (TCS) (Ramadge & Wonham, 1989; Cassandras & Lafortune, 2009) que, além de modelar o comportamento das máquinas de um sistema, permite a modelagem das restrições às quais esse sistema está sujeito, chamadas especificações. Assim, como resultado da combinação do modelo das plantas, das especificações e alguns procedimentos que serão apresentados posteriormente, chega-se a um supervisor que contém todas as combinações possíveis de eventos que não ferem a segurança do sistema.

Vislumbra-se, assim, a possibilidade de tratar um problema de escalonamento de tarefas, conhecido e estudado nas áreas de Pesquisa Operacional e de Engenharia de Produção, utilizando a teoria de linguagens e autômatos e a TCS. Estabelece-se, então, uma ponte entre essas áreas, possibilitando o desenvolvimento de novas pesquisas, principalmente para a comunidade de SED que tem buscado aplicar seu recente desenvolvimento teórico a problemas práticos.

Voltado ainda para o contexto industrial, um outro cenário é tratado neste trabalho, e diz respeito ao uso de uma heurística em conjunto com a TCS para otimizar uma sequência de produção com demandas estocásticas. Embora não seja o ponto central desta dissertação, esse conteúdo foi desenvolvido também durante o período do mestrado, o que justifica a sua inclusão no presente trabalho.

1.1 Objetivos

Este trabalho tem como principais objetivos:

- desenvolver um algoritmo que gere, automaticamente, os autômatos que modelam um problema de escalonamento *job shop* flexível com bloqueio, a partir de instâncias em formato textual, como encontrado na literatura;
- computar o supervisor que modela o comportamento em malha fechada de cada problema.

São considerados objetivos secundários os seguintes:

- aplicar uma heurística de minimização de *makespan* utilizando os modelos encontrados;
- desenvolver um algoritmo para resolver o problema de planejamento da produção em um sistema flexível de manufatura com demandas estocásticas.

Em outras palavras, a ideia central deste trabalho é explorar a possibilidade de utilizar a Teoria de Controle Supervisório para abordar um problema conhecido e estudado por outra área de conhecimento. Dessa forma, torna-se possível agregar as vantagens que a TCS oferece, sendo a principal a delimitação de todo o espaço de busca, buscando mitigar algumas dificuldades ou limitações. Sendo isso possível, um passo a mais é dado para se resolver problemas mais complexos, e com mais segurança.

1.2 Estrutura da Dissertação

Esta dissertação é composta por seis capítulos. O presente capítulo apresenta a introdução do trabalho desenvolvido e seus objetivos. O Capítulo 2 reúne a revisão bibliográfica dos assuntos pertinentes ao estudo em questão. O Capítulo 3 apresenta os conceitos introdutórios necessários para o melhor entendimento do leitor. Posteriormente, o Capítulo 4 aponta a metodologia para a tradução automática do problema de FJSSP em diferentes configurações incluindo os resultados encontrados. No mesmo capítulo são apresentados os resultados numéricos da aplicação de uma heurística dessas estruturas. Já o Capítulo 5 propõe uma metodologia para solucionar o problema do escalonamento de tarefas de um sistema flexível de manufatura com demandas estocásticas. Por fim, o Capítulo 6, aponta as conclusões deste trabalho e apresentadas algumas propostas para o desenvolvimento futuro.

Capítulo 2

Revisão da Literatura

2.1 O Problema do Escalonamento de Tarefas

A ideia do planejamento e escalonamento de tarefas é conhecida desde a época da construção das pirâmides (Weaver, 2006), mas somente no fim da II Guerra Mundial as organizações militares começaram a se dedicar mais a fundo à pesquisa e à discussão sobre logística, organização e planejamento das suas atividades (Eccles, 1954).

Com isso, nos anos 50 e 60, observou-se o aparecimento de publicações mais relevantes sobre o problema de *scheduling*, tanto no contexto militar quanto no industrial, entre os quais destaca-se o periódico *Naval Research Logistics Quarterly*, que concentrou boa parte dessas publicações. Exemplos desse tipo de publicação são os trabalhos de Levy et al. (1962), que trata do problema de escalonamento de tarefas visando a redução da mão de obra em estaleiros; de Smith (1956), que trata de um sistema de produção considerando uma máquina e diferentes funções objetivo; e de Johnson (1954), que considera um problema com duas e três máquinas.

Nas décadas seguintes, grande foi o desenvolvimento das pesquisas e crescimento do número de publicações nessa área. Importantes livros foram escritos e serviram de base para os estudos futuros, como Baker (1974), Coffman & Bruno (1976) e French (1982). Diversos modelos determinísticos e estocásticos foram apresentados, assim como igualmente numerosos foram os métodos, exatos ou heurísticos, propostos (Baker & Trietsch, 2013). Por ser o foco desse trabalho, serão apresentadas a seguir algumas abordagens para a solução do problema de *job shop* flexível e, posteriormente, alguns trabalhos de otimização utilizando autômatos.

O desenvolvimento de heurísticas como citado acima deu-se, principalmente, pelo fato de grande parte dos problemas determinísticos de escalonamento não possuírem algoritmos de tempo polinomial. Esses problemas se enquadram na classe denominada NP-Difícil (Garey & Johnson, 1979).

2.2 Técnicas para Solução de FJSSP

É comum encontrarmos na literatura a representação do problema de FJSSP no formato de um grafo disjuntivo. Trata-se de um grafo orientado $G = (V, C \cup D)$, onde V é o conjunto

dos vértices e representam as operações dos *jobs* (tarefas) a serem executados. A esse conjunto são adicionados dois vértices, sendo um que antecede e outro que sucede todas as tarefas, representando o início e o fim do escalonamento. O tempo de processamento desses dois vértices é igual a zero, de forma a não afetar a solução. O conjunto C contém os arcos conjuntivos que relacionam aquelas tarefas de um mesmo *job* que possuem relação de precedência. O conjunto D contém os arcos disjuntivos não direcionados que une as tarefas que são processadas na mesma máquina. O peso dos arcos é definido de acordo com o tempo de processamento de cada tarefa (Błażewicz et al., 2000).

Nessa representação, uma solução para o problema do escalonamento de *job shop* flexível é encontrada transformando cada arco disjuntivo não direcionado em um arco conjuntivo direcionado, de forma que haja um caminho entre o vértice inicial e o final, passando por todos os outros vértices. O grafo resultante deve ser acíclico e, para o problema de minimização do *makespan*, o maior caminho entre os vértices inicial e final deve ser mínimo.

A Figura 2.1 apresenta um exemplo de um grafo disjuntivo para FJSSP. O problema representado é composto pelos *jobs* J_1 , J_2 e J_3 , sendo que J_1 é composto das operações 1, 2 e 3, J_2 é composto de apenas duas operações 4 e 5, e J_3 é composto das operações 6, 7 e 8. Os arcos conjuntivos (linhas cheias) mostram a relação de precedência das operações. Em J_2 , por exemplo, a operação 4 antecede a 5. Os vértices 0 e 9 representam o início e o fim do escalonamento e não estão associados a nenhum *job*.

As cores dos vértices indicam a máquina que processa cada operação, ou seja, as operações coloridas com a cor cinza escura são processada na máquina M_1 (operações 1 e 7), enquanto as preenchidas de branco são processadas na máquina M_2 (operações 2, 5 e 6) e as com a cor cinza claro, na máquina M_3 (operações 3, 4 e 8). Os arcos disjuntivos (linhas tracejadas) ligam as operações que são processadas nas mesmas máquinas. Os tempos de processamento estão associados aos arcos conjuntivos que saem de cada vértice.

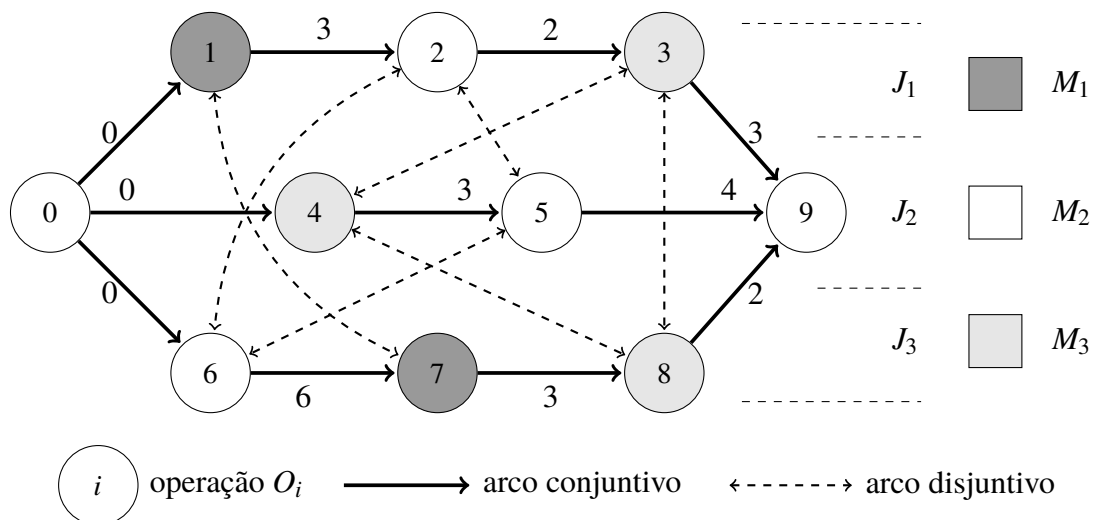


Figura 2.1: Exemplo de grafo disjuntivo. Fonte: Błażewicz et al. (2000)

São apresentados, nesta seção e na próxima, alguns trabalhos existentes na literatura, com o objetivo de ilustrar diferentes técnicas existentes para tratar o problema de FJSSP e de otimização, sem a pretensão de apresentar uma lista exaustiva.

Brucker et al. (1994) usam uma técnica chamada *branch and bound* (B&B) que utiliza a representação de uma árvore para enumerar as possíveis soluções para o FJSSP. Nesse tipo de técnica, cada solução é uma combinação das operações necessárias para a produção de todos os *jobs*, sendo que cada ramificação é uma variação da solução anterior. Para cada ramificação, calculam-se os limites superior e inferior da função objetivo, com os quais se verificam e eliminam aquelas soluções que não contêm a solução ótima. Essa técnica pode encontrar, para problemas pequenos, a solução ótima e, para problemas grandes, os limites superior e inferior da solução ótima.

Adams et al. (1988) utilizam a heurística *shifting bottleneck* (SB) para tratar o problema de minimização de *makespan*. Nessa técnica, cada máquina é sequenciada separadamente, geralmente de forma ótima, resolvendo o problema de *single-machine scheduling*. A máquina escolhida é sempre aquela vista como “gargalo” para minimização do *makespan*, a partir de um ranqueamento quantitativo. Sempre que um sequenciamento é feito, otimizam-se novamente as demais sequências das outras máquinas já sequenciadas sujeitas a melhorias. O critério de parada é atingido quando todos os recursos são otimizados.

Dell’Amico & Trubian (1993) propõem solucionar o problema em questão por meio de uma busca tabu (*tabu search* - TS), uma metaheurística baseada em busca local. Define-se uma estrutura de vizinhança que associa a uma solução um conjunto de outras soluções obtidas por meio de uma modificação parcial nela. Um exemplo de vizinhança é o conjunto de soluções obtidas pela permutação entre dois *jobs*. A nova solução é aquele vizinho que obtiver a melhor avaliação da função objetivo e que não esteja proibido, ou seja, não esteja na lista tabu de soluções já encontradas. Essa lista evita a criação de ciclos, que levariam sempre para as mesmas soluções. A simulação é interrompida após um número de iterações pré-definido ou após uma sequência de iterações sem melhorias.

Van Laarhoven et al. (1992) utilizam a técnica *simulated annealing* (SA), que é inspirada em conceitos estatísticos da física, mais precisamente, da termodinâmica. Embora seja uma busca local movendo-se por uma vizinhança essa técnica permite alguns movimentos que possam piorar a função objetivo, a fim de explorar melhor o espaço de soluções. Cada candidato a solução da vizinhança tem uma probabilidade de ser escolhida em função de um parâmetro de controle que diminui ao longo da execução do algoritmo. Por essas características, as soluções encontradas são, geralmente, próximas do ótimo global.

Pezzella et al. (2008) apresentam uma solução por meio de um algoritmo genético (*genetic algorithm* - GA). Essa abordagem considera uma solução inicial, chamada de pai, com a qual se geram outras soluções, chamadas de filhos, a partir de cruzamentos e mutações. Elas são avaliadas quanto à função objetivo e escolhidas constituindo uma nova população, que serão os pais da geração seguinte. Essas etapas se repetem até que o critério de parada, que é o número de gerações, seja atingido, e o algoritmo retorne o melhor escalonamento encontrado com o seu valor de *makespan*.

Abdeddaim & Maler (2001) usam a teoria de autômatos temporizados (*timed automata* - TA) para abordar o problema. Cada *job* é representado por um autômato temporizado, em que os tempos de processamento estão associados à estrutura de relógio inerente a essa representação. Uma representação global do sistema é obtida pela combinação dos autômatos. A partir dessa representação global, é computado o mínimo *makespan*, obtendo-se o menor caminho no autômato temporizado global por meio da ferramenta de verificação *Kronos*.

É interessante observar a diversidade de técnicas utilizadas para solucionar o mesmo

tipo de problema. Este trabalho se enquadra como uma alternativa para a obtenção de um espaço de busca completo e limitado, de forma a facilitar a obtenção de soluções factíveis. A heurística aplicada neste trabalho para obtenção dos valores de *makespan* de alguns problemas possui algumas características de uma busca *branch and bound*.

2.3 Trabalhos de Otimização Utilizando Autômatos

São encontrados na literatura trabalhos que se dedicam a resolver diversos problemas de otimização, dentro do contexto do escalonamento de tarefas, utilizando a teoria de autômatos. Dentre os trabalhos analisados, no entanto, nenhum utiliza essa representação para resolver o problema de FJSSP.

Jezequel & Fabre (2012), Pinha et al. (2011) e Su et al. (2011) usam *time-weighted systems* (TWS) para a minimização de diferentes funções objetivo como atraso e *makespan*. Essa abordagem é, também, uma derivação da teoria de autômatos, sendo que a principal diferença entre elas diz respeito a uma função de peso que atribui um valor para cada transição em função do tempo de processamento de uma tarefa.

Jo et al. (2010) utilizam autômatos para otimizar as tarefas de robôs de serviços em um restaurante, aumentando a qualidade do serviço oferecido e diminuindo o gasto de energia dos robôs. Alves et al. (2016, 2019) apresentam uma heurística eficiente para a maximização do paralelismo entre as máquinas de um sistema flexível como forma indireta para a minimização de *makespan*.

Malik & Pena (2018) modelam um sistema flexível de manufatura por meio de autômatos e usam verificação formal para encontrar o valor ótimo de *makespan* para a produção de uma quantidade arbitrária de produtos. Costa et al. (2018) utilizam essa mesma modelagem para propor uma metodologia que busca encontrar sequências de produção otimizadas para grandes lotes em sistemas mais complexos com custo computacional reduzido. Hill & Lafortune (2016) realizam a otimização sobre uma abstração de autômatos a fim de reduzir o espaço de busca e, conseqüentemente, o tempo necessário para encontrar a solução.

2.4 Instâncias de FJSSP

São encontradas, na literatura, diferentes instâncias para o problema de *job shop* flexível. Essas instâncias foram geradas pelos próprios pesquisadores, para seus trabalhos nas décadas de 80 e 90, dada a necessidade de testarem os algoritmos de otimização propostos. Os exemplos gerados formam, hoje, uma grande base de dados para outras pesquisas nessa área. A Tabela 2.1 apresenta algumas instâncias amplamente conhecidas. Para garantir a consistência deste trabalho, foram utilizadas algumas dessas instâncias.

Para favorecer a compreensão das informações que se seguem, explicita-se que a dimensão de um problema é definida como $(n \times m)$, onde n é o número de *jobs* e m é o número de máquinas.

As principais características das instâncias da Tabela 2.1 estão apresentadas a seguir:

1. as instâncias mt06, mt10 e mt20 têm dimensões (6×6) , (10×10) e (20×5) , respectivamente. São problemas bastante conhecidos e comumente utilizados como

Tabela 2.1: Instâncias de FJSSP

N°	Instâncias	Jobs	Máquinas	Referência
1	mt06, 10, 20	6..20	5..10	Muth et al. (1963)
2	mk01..10	10..20	4..15	Brandimarte (1993)
3	01..18a	10..20	5..10	Dauzère-Pérès & Paulli (1997)
4	la01..40	10..30	5..15	Lawrence (1984)
5	orb1..10	10	10	Applegate & Cook (1991)
6	mt10_, setb4_, seti5_	10..15	11..18	Chambers & Barnes (1996)
7	abz5..9	10..20	10..15	Adams et al. (1988)
8	car1..8	7..14	4..9	Carlier (1978)
9	hurink s,e,r,v data	6..30	4..15	Hurink et al. (1994)

estado da arte para testar a eficiência de um determinado método de solução. Embora não sejam, atualmente, tão desafiadores do ponto de vista computacional, o problema mt10, por exemplo, levou mais de 20 anos para ser solucionado (Yamada & Nakano, 1997).

- as instâncias mk01..10 tiveram seus tempos de processamento gerados de forma aleatória utilizando uma distribuição uniforme cujos limites são definidos em Brandimarte (1993). Além do número de *jobs* e o número de máquinas, são parâmetros considerados: os limites do número de operações por *job*, o limite máximo de máquinas que podem processar cada operação e os limites dos tempos de processamento.
- as instâncias 01..18a também tiveram seus tempos de processamento gerados por meio de uma distribuição uniforme entre os limites dados. Os problemas são gerados considerando um sistema flexível de manufatura em que o número de operações é maior que o número de máquinas e a flexibilidade é aumentada gradativamente (Dauzère-Pérès & Paulli, 1997).
- as instâncias la01..40 possuem o número de operações igual ao número de máquinas, de forma que cada operação tem uma única opção de máquina para ser processada. As dimensões dos problemas são: la01..05 (10×5), la06..10 (15×5), la11..15 (20×5), la16..20 (10×10), la21..25 (15×10), la26..30 (20×10), la31..35 (30×10), la36..40 (15×15) (Behnke & Geiger, 2012).
- as instâncias orb1..5 foram geradas em 1986 a partir de um desafio de gerar problemas “difíceis”. Mais tarde, para cada uma dessas instâncias, dois conjuntos de tempos de processamento são gerados aleatoriamente, e é mantido aquele que apresenta a maior distância entre o ótimo e o limite inferior padrão. Assim, são geradas as instâncias orb6..10. Todos os problemas desse grupo têm 10 *jobs* e 10 máquinas (Applegate & Cook, 1991).
- as instâncias desse grupo foram geradas com base nas instâncias mt10, la24 e la40. Cada uma foi transformada em outra instância por meio de replicações de determinadas máquinas. O critério para escolha da máquina pode ser o maior tempo de processamento

acumulado ou o maior número de operações do caminho crítico (Chambers & Barnes, 1996).

7. as instâncias abz5..9 também tiveram seus tempos de processamento gerados por meio de uma distribuição uniforme entre os limites estabelecidos. O número de operações é igual ao de máquinas e cada *job* possui sua própria rota (Adams et al., 1988).
8. as instâncias car1..8 se baseiam nas especificidades de uma oficina e os tempos de processamento são os tempos de usinagem das peças em cada máquina. Nos 8 problemas gerados, todos os *jobs* têm a mesma ordem de processamento e cada um tem o número de operações igual ao número de máquinas (Carlier, 1978).
9. as instâncias propostas por Hurink et al. (1994) foram geradas a partir das instâncias de Muth et al. (1963), Lawrence (1984), Adams et al. (1988), Applegate & Cook (1991) e Carlier (1978). Foram geradas outras instâncias por meio do aumento do número de máquinas capazes de processar cada operação a uma certa probabilidade. Com diferentes probabilidades, foram obtidos os seguintes conjuntos de dados:
 - (a) *sdata*: contém as versões originais das instâncias, em que o número de operações de cada *job* é igual ao número de máquinas, ou seja, a média de operações por máquina é igual a um;
 - (b) *edata*: contém instâncias um pouco mais flexíveis com a média de máquinas por operação igual a 1,15;
 - (c) *rdata*: contém instâncias bastante flexíveis, sendo que quase todas as operações podem ser processadas por mais de uma máquina. A média de máquinas por operação é igual a dois;
 - (d) *vdata*: contém instâncias totalmente flexíveis, ou seja, todas as operações podem ser processadas por mais de uma máquina. A média de máquinas por operação é igual à metade do número de máquinas.

A base de dados utilizada para este trabalho foi obtida de uma página da internet¹. Também estão disponíveis os valores de limite superior e inferior para todos os problemas mencionados². Estas tabelas estão apresentadas no Anexo A.

¹Disponível em: <https://cdn.quintiq.com/content/instances.zip> - Acessado em: 04/06/2019

²Disponível em: <https://www.quintiq.es/optimization/flexible-job-shop-scheduling-problem-results.html> - Acessado em: 04/06/2019

Capítulo 3

Preliminares

3.1 Problema de Escalonamento de Tarefas

O problema de escalonamento de tarefas trata da alocação de recursos para a realização de tarefas. Em um sistema de manufatura, por exemplo, as tarefas são chamadas de *jobs*, enquanto os recursos estão associados, geralmente, às 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 dimensão dos problemas é representada por $(n \times m)$.

A caracterização desse problema é dada por uma tripla $\alpha|\beta|\gamma$ (Graham et al., 1979) em que α indica a configuração do problema em relação às máquinas, β informa as características do processo e as restrições às quais ele está submetido e γ determina o objetivo a ser minimizado. Em Pinedo (2016) são apresentadas algumas possíveis entradas para cada um dos três campos mencionados.

3.1.1 Job Shop Scheduling Problem - JSSP

O problema clássico de escalonamento *job shop* considera que cada um dos n *jobs* possui sua própria rota (fixa) entre as m máquinas, ou seja, as operações dos *jobs* são processadas nas máquinas em ordens diferentes. O fato de a rota ser fixa implica que cada operação tem apenas uma opção de máquina para ser processada. Em geral, considera-se que cada *job* possui m operações fazendo com que cada uma delas visite cada máquina apenas uma vez.

O diagrama apresentado na Figura 3.1 ilustra, de forma simplificada, o que é problema de escalonamento *job shop*. Considerando um sistema com cinco máquinas e uma demanda para a produção de dois *jobs*, representados pelas formas geométricas triângulo e pentágono, as setas indicam a ordem das máquinas pelas quais os *jobs* devem passar, ou seja, $\triangle : M_2, M_1, M_4, M_5, M_3 \rightarrow \triangle$ e $\diamond : M_4, M_2, M_3, M_1 \rightarrow \diamond$

Cada operação é processada em uma máquina, ou seja, o *job* triângulo possui cinco operações enquanto o *job* pentágono tem apenas quatro. Os tempos de processamento das operações em uma mesma máquina podem ser diferentes.

A Tabela 3.1 apresenta um exemplo da atribuição dos tempos de processamento das operações em função das máquinas, ou seja, o tempo de processamento em cada máquina varia de acordo com cada *job*. O tempo infinito (∞) é atribuído às operações que não podem

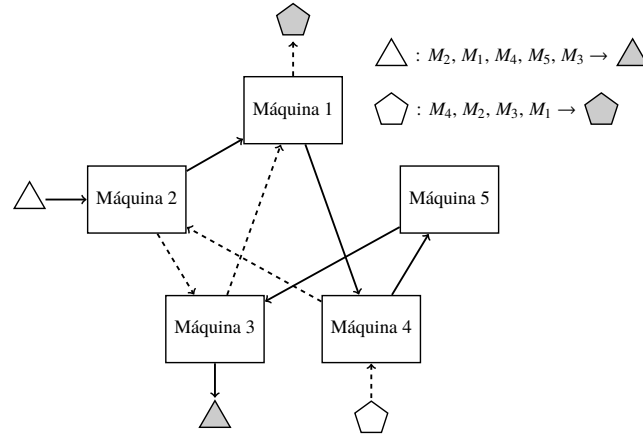


Figura 3.1: Diagrama de um problema de escalonamento *job shop*

ser processadas em determinada máquina (Behnke & Geiger, 2012).

Tabela 3.1: Atribuição dos tempos de processamento em um problema clássico de JSSP

$p_{i,j}$		M_1	M_k	\dots	M_m
J_1	$O_{1,1}$	∞	∞	\dots	$p_{1,1}$
	$O_{1,2}$	$p_{1,2}$	∞	\dots	∞
	\dots	\dots	\dots	\dots	\dots
	O_{1,h_1}	∞	p_{1,h_1}	\dots	∞
J_i	$O_{i,1}$	∞	$p_{i,1}$	\dots	∞
	$O_{i,2}$	$p_{i,2}$	∞	\dots	∞
	\dots	\dots	\dots	\dots	\dots
	O_{i,h_i}	∞	∞	\dots	p_{i,h_i}
\dots		\dots	\dots	\dots	\dots
J_n	$O_{n,1}$	$p_{n,1}$	∞	\dots	∞
	$O_{n,2}$	∞	∞	\dots	$p_{n,2}$
	\dots	\dots	\dots	\dots	\dots
	O_{n,h_n}	∞	p_{n,h_n}	\dots	∞

3.1.2 Flexible Job Shop Scheduling Problem - FJSSP

O problema de escalonamento *job shop* flexível é uma generalização que estende o problema clássico de JSSP. Nele considera-se, igualmente, n jobs a serem processados em m máquinas, e a principal diferença é que cada operação pode ser processada por um conjunto de máquinas (flexibilidade parcial) ou por todas as máquinas (flexibilidade total). Nesse caso, ainda que cada *job* possua m operações, uma máquina pode ser visitada mais de uma vez.

O diagrama apresentado na Figura 3.2 ilustra o que é problema de escalonamento *job shop* flexível (flexibilidade parcial) com as mesmas características do exemplo para JSSP. A principal diferença é que o *job* pentágono, após ser processado na máquina 2, pode ser admitido na máquina 3 ou na 5. Qualquer que seja a máquina escolhida, a operação seguinte tem que ser processada na máquina 1. Nesse caso, tem-se que $\triangle : M_2, M_1, M_4, M_5, M_3 \rightarrow \triangle$

e $\diamondsuit : M_4, M_2, M_3$ ou $M_5, M_1 \rightarrow \diamondsuit$

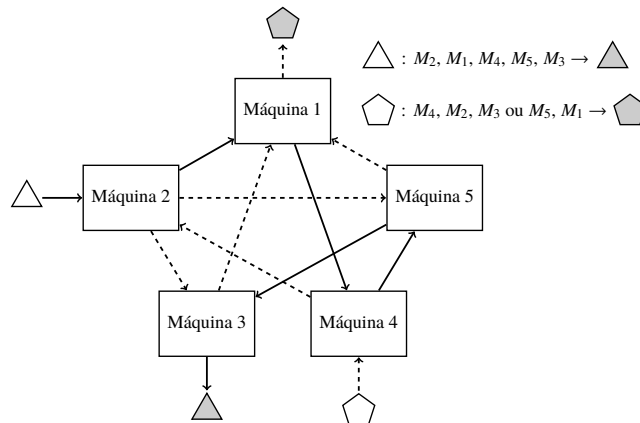


Figura 3.2: Diagrama de um problema de escalonamento *job shop* flexível

A Tabela 3.2 apresenta a atribuição dos tempos de processamento das operações em função das máquinas para um exemplo de flexibilidade total. Por isso, todos os campos da tabela são preenchidos com algum valor $p \in \mathbb{R}_+$, ou seja, com algum número real não-negativo (Behnke & Geiger, 2012).

Tabela 3.2: Atribuição dos tempos de processamento em um problema de FJSSP com flexibilidade total

$p_{i,j,k}$		M_1	M_k	\dots	M_m
J_1	$O_{1,1}$	$p_{1,1,1}$	$p_{1,1,k}$	\dots	$p_{1,1,m}$
	$O_{1,2}$	$p_{1,2,1}$	$p_{1,2,k}$	\dots	$p_{1,2,m}$
	\dots	\dots	\dots	\dots	\dots
	O_{1,h_1}	$p_{1,h_1,1}$	$p_{1,h_1,k}$	\dots	$p_{1,h_1,m}$
J_i	$O_{i,1}$	$p_{i,1,1}$	$p_{i,1,k}$	\dots	$p_{i,1,m}$
	$O_{i,2}$	$p_{i,2,1}$	$p_{i,2,k}$	\dots	$p_{i,2,m}$
	\dots	\dots	\dots	\dots	\dots
	O_{i,h_i}	$p_{i,h_i,1}$	$p_{i,h_i,k}$	\dots	$p_{i,h_i,m}$
\dots		\dots	\dots	\dots	\dots
J_n	$O_{n,1}$	$p_{n,1,1}$	$p_{n,1,k}$	\dots	$p_{n,1,m}$
	$O_{n,2}$	$p_{n,2,1}$	$p_{n,2,k}$	\dots	$p_{n,2,m}$
	\dots	\dots	\dots	\dots	\dots
	O_{n,h_n}	$p_{n,h_n,1}$	$p_{n,h_n,k}$	\dots	$p_{n,h_n,m}$

3.1.3 Job Shop Scheduling Problem com bloqueio

Os problemas de escalonamento *job shop* consideram a existência de *buffers* ilimitados entre as máquinas de tal forma que, imediatamente após uma máquina finalizar sua tarefa, ela fica apta a processar outra operação. Dentro do contexto de simulações, é possível considerar essa suposição, mas, em sistemas reais, não há como conceber um espaço para armazenamento de tamanho ilimitado.

Um cenário mais realista é o de *buffers* limitados ou inexistentes entre as máquinas. Nesse último caso, a própria máquina desempenha o papel do *buffer*, ou seja, ao finalizar uma operação, a máquina retém o *job* até que a próxima máquina esteja disponível para processar a operação seguinte, caracterizando o bloqueio.

O problema do bloqueio pode ter duas configurações distintas: o bloqueio com troca (*blocking with swap* - BWS) e o bloqueio sem troca (*blocking no-swap* - BNS). Em um cenário em que um ciclo de duas ou mais máquinas estejam bloqueadas aguardando o *job* da outra máquina desse ciclo, o problema de bloqueio com troca permite que cada operação seja admitida, simultaneamente, na máquina subsequente do ciclo. Já o problema de bloqueio sem troca não permite esse movimento, caracterizando um *deadlock*.

Alguns trabalhos como Pranzo & Pacciarelli (2016), AitZai et al. (2012) e Louaqad & Kamach (2015) abordam a questão da infactibilidade de soluções em problemas de *job shop* com bloqueio. Dependendo da forma como o algoritmo é implementado, pode acontecer de, durante a busca e construção da solução, serem encontradas soluções que não podem ser implementadas do ponto de vista lógico ou físico.

Quando isso acontece, uma alternativa é descartar a solução, o que significa desperdício de recurso computacional. Outra alternativa utilizada é voltar alguns passos e buscar um novo caminho para a solução. Entretanto não há a garantia de que o novo caminho escolhido resulta em uma solução factível. Esse processo pode acontecer inúmeras vezes até que o algoritmo encontre uma solução, acarretando um aumento do custo computacional.

Para contornar esse problema, outras técnicas podem ser utilizadas, como a escolhida neste trabalho: utilizar autômatos e a TCS para resolver problemas de escalonamento *job shop* com bloqueio.

3.2 Sistemas a Eventos Discretos

Sistemas a eventos discretos são sistemas dinâmicos regidos pela ocorrência de eventos, que são entidades externas e internas que interagem com o modelo, modificando seu estado de forma instantânea. Os eventos podem representar o acionamento de um botão ou de um sensor, a chegada de um produto, a finalização de uma tarefa, entre outros. Esse tipo de sistema se diferencia daqueles dirigidos pelo tempo, por exemplo, em que o estado varia continuamente e cujas soluções utilizam recursos da Teoria de Controle Clássica.

3.2.1 Linguagens

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 $|\sigma_1\sigma_2\dots\sigma_i| = i$, onde $i > 0$. O comprimento de uma 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 \bar{L} de uma linguagem $L \subseteq \Sigma^*$ é o conjunto de todos os prefixos de cadeias em L , expresso por $\bar{L} = \{s \in \Sigma^* | s \leq t \text{ para algum } t \in L\}$.

A projeção natural $P : \Sigma_1 \rightarrow \Sigma_2$ é realizada de um alfabeto Σ_1 para outro Σ_2 , sendo $\Sigma_2 \subseteq \Sigma_1$. Essa operação apaga os eventos de uma cadeia que estão em um alfabeto e não estão em outro e é definida como

$$\begin{aligned} P(\epsilon) &= \epsilon \\ P(\sigma) &= \begin{cases} \sigma & \text{se } \sigma \in \Sigma_2 \\ \epsilon & \text{se } \sigma \in \Sigma_1 \setminus \Sigma_2 \end{cases} \\ P(s\sigma) &= P(s)P(\sigma) \text{ para } s \in \Sigma_1^*, \sigma \in \Sigma_1. \end{aligned}$$

A projeção inversa P^{-1} é realizada sobre uma cadeia de um alfabeto menor Σ_2 para um maior Σ_1 e é definida como

$$P^{-1}(t) = \{s \in \Sigma_1 \mid P(s) = t\}.$$

As definições de projeção natural e projeção inversa podem ser estendidas para linguagens, conforme mostrado abaixo

$$\begin{aligned} P(L) &= \{t \in \Sigma_2^* \mid (\exists s \in L) [P(s) = t]\}, \text{ para } L \subseteq \Sigma_1^* \\ P^{-1}(L) &= \{s \in \Sigma_1^* \mid (\exists t \in L) [P(s) = t]\}, \text{ para } L \subseteq \Sigma_2^*. \end{aligned}$$

3.2.2 Autômatos

Uma forma gráfica de representar linguagens é por meio de autômatos, que 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, $\delta : Q \times \Sigma \rightarrow Q$, 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 $q_1 = q_2$.

Um exemplo de autômato determinístico de estados finitos é apresentado na Figura 3.3. Nesse autômato tem-se que: $Q = \{s_0, s_1\}$, $\Sigma = \{a, b\}$, $\delta(s_0, a) = s_1$, $\delta(s_1, b) = s_0$, $q_0 = s_0$ e $Q_m = \{s_0\}$.

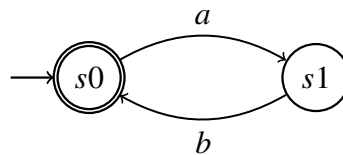


Figura 3.3: Exemplo de um autômato determinístico

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 $\Gamma : Q \rightarrow 2^\Sigma$ é dado 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 $\mathcal{L}(G) = \{s \in \Sigma^* \mid \delta(q_0, s)!\}$ e representa o conjunto de cadeias de eventos que, partindo do estado inicial, leva a algum estado do

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

Operações Sobre Autômatos

Dentre as diferentes operações sobre autômatos, a operação de parte acessível $Ac(G)$ remove todos aqueles estados do autômato original G que não podem ser acessados a partir do estado inicial. Sendo assim, a linguagem gerada e a linguagem marcada não se alteram. Um autômato é dito acessível se $G = Ac(G)$. A Figura 3.4 mostra um exemplo dessa operação, na qual o estado s_2 é removido.

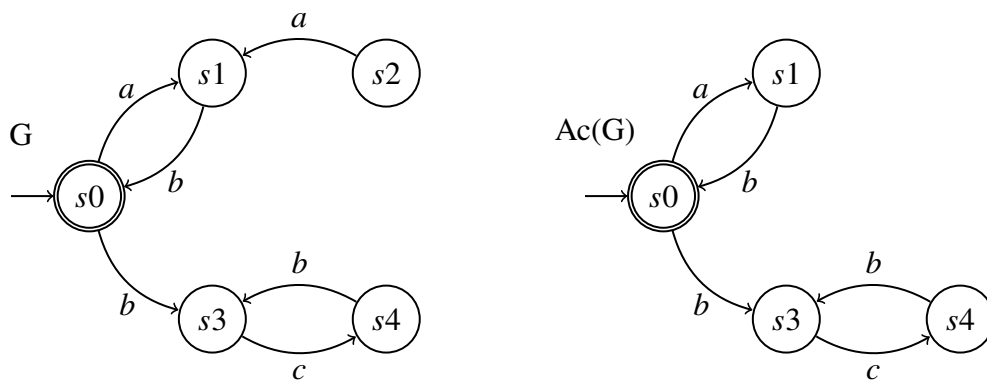


Figura 3.4: Operação parte acessível $Ac(G)$

A operação de parte coacessível $CoAc(G)$, por sua vez, remove do autômato G todos os estados a partir dos quais não se acessa um estado marcado. A linguagem gerada pode ser reduzida, mas a linguagem marcada permanece inalterada. Um autômato é dito coacessível se $G = CoAc(G)$ e, nesse caso $\mathcal{L}(G) = \overline{\mathcal{L}_m(G)}$. A Figura 3.5 mostra um exemplo dessa operação, na qual os estados s_3 e s_4 são removidos.

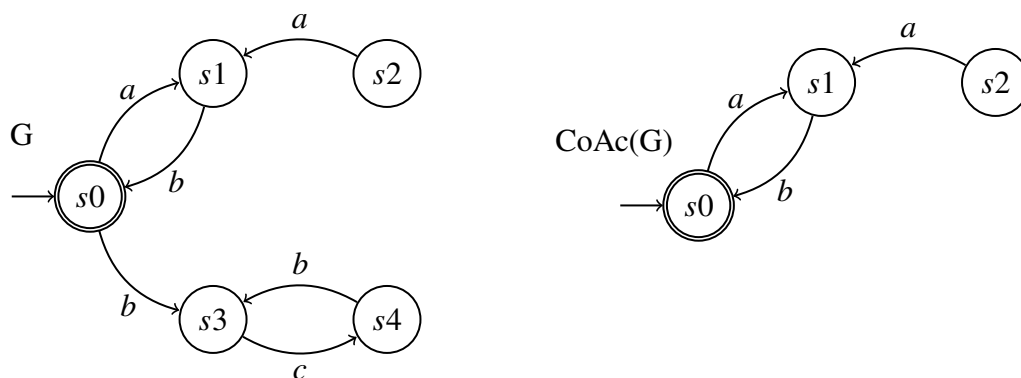


Figura 3.5: Operação parte coacessível $CoAc(G)$

Quando um autômato não atende ao critério da coacessibilidade, ele é chamado de bloqueante, já que existem estados em que, uma vez neles, não é mais possível alcançar um estado marcado. O bloqueio pode ocorrer em dois cenários distintos. Um deles, chamado de

deadlock, ocorre quando o sistema chega a um estado não marcado que não possui nenhum evento habilitado, ou seja para algum $q \in (Q \setminus Q_m), \delta(q, \sigma) = \emptyset, \forall \sigma \in \Sigma$. Nesse caso, o sistema permanece travado nesse estado. Outro cenário, chamado de *livelock*, ocorre quando um conjunto de estados não marcados formam uma componente fortemente conexa. Nesse caso, embora haja sempre transições habilitadas em todos esses estados, não há nenhuma transição que leve para um estado fora desse conjunto, ou seja, o sistema fica “vivo”, mas nunca alcança o estado marcado. Um *livelock* pode ser observado nos estados $s3$ e $s4$ do autômato G da Figura 3.5.

O *trim* é uma operação que realiza tanto a operação de parte acessível quanto a de parte coacessível, em qualquer ordem, resultando em um autômato cuja linguagem marcada é a mesma do autômato sobre o qual se aplicou a operação. A operação é definida como $trim(G) = CoAc[Ac(G)] = Ac[CoAc(G)]$. A Figura 3.6 mostra um exemplo dessa operação.

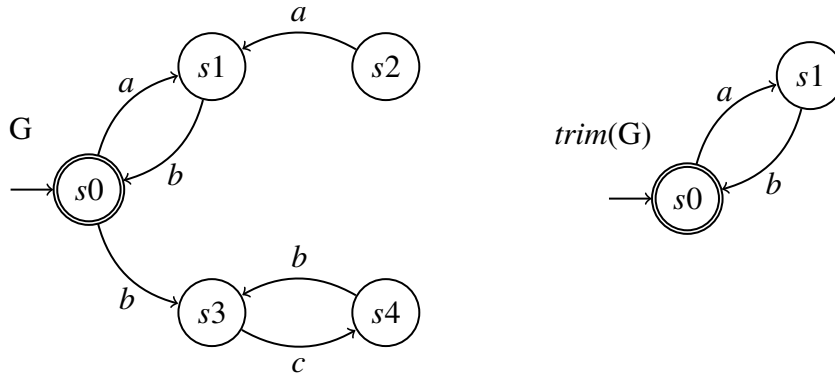


Figura 3.6: Operação $Trim(G)$

Uma das operações 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

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

e $\Gamma_{1 \parallel 2}(q_1, q_2) = [\Gamma_1(q_1) \cap \Gamma_2(q_2)] \cup [\Gamma_1(q_1) \setminus \Sigma_2] \cup [\Gamma_2(q_2) \setminus \Sigma_1]$, em que a notação $\Gamma(q) \setminus \Sigma$ indica a subtração dos conjuntos. A linguagem gerada pelo autômato resultante é dada por $\mathcal{L}(G_{1 \parallel 2}) = P_{\Sigma_2}^{-1}(\mathcal{L}(G_1)) \cap P_{\Sigma_1}^{-1}(\mathcal{L}(G_2))$, em que $P_{\Sigma_2}^{-1} : \Sigma_1 \cup \Sigma_2 \rightarrow \Sigma_2$ e $P_{\Sigma_1}^{-1} : \Sigma_1 \cup \Sigma_2 \rightarrow \Sigma_1$.

A operação de composição paralela pode ser aplicada também a linguagens. A linguagem gerada e a linguagem marcada são dadas por:

$$\begin{aligned} \mathcal{L}(G_1 \parallel G_2) &= P_{\Sigma_1}^{-1}[\mathcal{L}(G_1)] \cap P_{\Sigma_2}^{-1}[\mathcal{L}(G_2)] \\ \mathcal{L}_m(G_1 \parallel G_2) &= P_{\Sigma_1}^{-1}[\mathcal{L}_m(G_1)] \cap P_{\Sigma_2}^{-1}[\mathcal{L}_m(G_2)] \end{aligned}$$

sendo $P_{\Sigma_2}^{-1} : \Sigma_1 \cup \Sigma_2 \rightarrow \Sigma_2$ e $P_{\Sigma_1}^{-1} : \Sigma_1 \cup \Sigma_2 \rightarrow \Sigma_1$.

3.3 Teoria de Controle Supervisório

A Teoria de Controle Supervisório (TCS) (Ramadge & Wonham, 1989) permite encontrar um supervisor que seja não-bloqueante e minimamente restritivo. Isso significa que se chega a uma estrutura de controle que opera desabilitando os eventos não seguros em relação à vivacidade e à segurança do sistema. Essa proteção está relacionada tanto à parte lógica, como evitar algum tipo de bloqueio (*deadlock* ou *livelock*), quanto à parte física, como evitar que um produto seja manufaturado duas vezes. A Figura 3.7 ilustra a atuação do supervisor em relação a um sistema. O supervisor observa os eventos ocorridos na planta e atua enviando para ela os eventos a serem desabilitados.

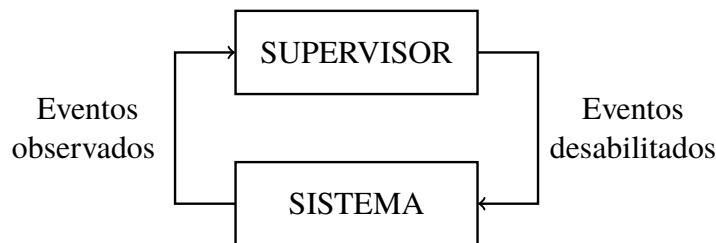


Figura 3.7: Funcionamento do supervisor

O sistema controlado é chamado de planta e seu comportamento em malha aberta pode ser representado pelo autômato G , que é obtido pela composição paralela dos modelos de cada uma das máquinas $G = \parallel G_i$, onde i é o índice de cada máquina modelada. O conjunto de restrições às quais o sistema está submetido, também chamadas de especificações, é obtido igualmente pela composição paralela de cada especificação modelada separadamente $E = \parallel E_i$.

Para essa modelagem, o alfabeto é particionado em dois subconjuntos disjuntos $\Sigma = \Sigma_c \cup \Sigma_{uc}$, sendo o primeiro, Σ_c , dos eventos controláveis, e o segundo, Σ_{uc} , dos eventos não controláveis. Este último, como o nome sugere, é composto pelos eventos que não podem ser impedidos de ocorrer.

Para se encontrar o autômato K , que modela o comportamento em malha fechada desse sistema, ou seja, o comportamento da planta sob as suas restrições, é feita a composição paralela entre os autômatos $K = G \parallel E$, de forma que $\mathcal{L}_m(K) = \mathcal{L}_m(G \parallel E) \subseteq \mathcal{L}_m(G)$. O autômato K atende ao critério da controlabilidade em relação a planta G se

$$\overline{K}\Sigma_{uc} \cap \mathcal{L}(G) \subseteq \overline{K}.$$

A propriedade da controlabilidade estabelece que nenhum evento não-controlável pode ser desabilitado, que equivale a dizer que para qualquer cadeia $s \in \overline{K}$ concatenada com um evento $\sigma \in \Sigma_{uc}$ que pertença a $\mathcal{L}(G)$, $s\sigma$ deve ser um prefixo de K , $s\sigma \in \overline{K}$.

No caso de K não atender a esse critério, existe uma máxima sublinguagem controlável, expressa por $SupC(K, G) \subseteq K$. Essa operação é feita de forma iterativa, eliminando aqueles

estados que desabilitam algum evento $\sigma \in \Sigma_{uc}$ e verificando novamente a controlabilidade. O processo continua até que a controlabilidade seja satisfeita. Nesse caso pode-se dizer que é encontrado um supervisor $S = SupC(K, G)$, minimamente restritivo e não-bloqueante, que modela todo o comportamento em malha fechada da planta controlada.

O supervisor obtido dessa forma contém todo o espaço de busca para o problema modelado. Assim, qualquer que seja a sequência de eventos escolhida que faça parte da linguagem gerada pelo supervisor, tem-se a garantia de que tal sequência é factível. Nesse novo cenário, não é necessário gastar recurso computacional tratando sequências que não podem ocorrer. Essa é, portanto, a principal vantagem do uso da TCS para tratar problemas como o de *job shop* com bloqueio.

3.4 Consideração Sobre o Termo “bloqueio”

Durante a realização deste trabalho observou-se que a palavra “bloqueio” tem significado diverso dependendo do contexto em que ela 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 um processo aguarda um recurso de outro processo que, por sua vez, aguarda um recurso do primeiro, por exemplo. Quando isso acontece, o sistema trava, pois nenhum dos processos envolvidos é capaz de completar sua tarefa. O *livelock* ocorre quando o sistema fica “preso” em um ciclo de ações que levam o sistema sempre ao mesmo ponto, sem gerar nenhum resultado em prol da conclusão da tarefa.

Quando no âmbito dos problemas de escalonamento de tarefas, “bloqueio” é uma das entradas da variável β , na notação $\alpha|\beta|\gamma$, referente às características do processo. Conforme já apresentado na Seção 3.1.3, o bloqueio existe quando os *buffers* são limitados ou inexistentes, de forma que uma máquina precisa reter o *job* até que a próxima máquina possa recebê-lo. Assim, o *job* que permanece na máquina a bloqueia de processar outro *job*. Embora um problema de escalonamento possibilite a ocorrência de *deadlocks*, fica clara a diferença do uso desse termo.

3.5 Ferramentas auxiliares

A fim de auxiliar a execução das etapas realizadas neste trabalho, são utilizadas algumas ferramentas auxiliares: um sistema conhecido pela comunidade de SED, os princípios da modelagem de SED, uma biblioteca para implementação de autômatos, uma heurística e o ambiente de desenvolvimento dos algoritmos. Essas ferramentas são apresentadas e explicadas nesta seção.

3.5.1 Sistema Flexível de Manufatura

Uma planta comumente utilizada em trabalhos envolvendo SED e a TCS é a do Sistema Flexível de Manufatura (SFM) (Queiroz et al., 2004). Esse sistema é capaz de produzir dois tipos diferentes de produtos: o Produto A, bloco com pino cônico sem pintura (Figura 3.8(a)), e o Produto B, bloco com pino cilíndrico pintado (Figura 3.8(b)).

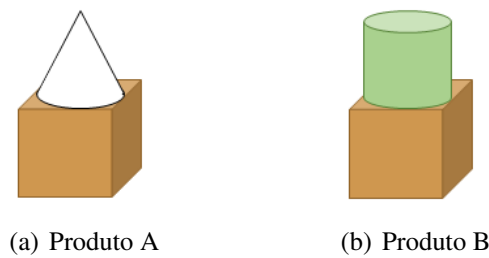


Figura 3.8: Produtos do Sistema Flexível de Manufatura

O sistema é composto por três esteiras C_1 , C_2 e C_3 , um robô, uma fresa, um torno, uma máquina de pintura, uma máquina de montagem e oito *buffers* unitários, apresentados na Figura 3.9.

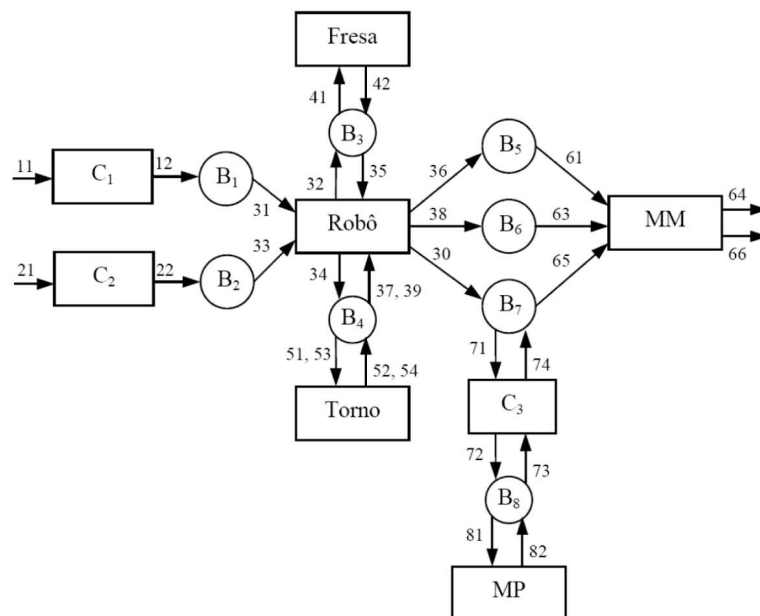


Figura 3.9: Diagrama do Sistema Flexível de Manufatura

A modelagem de cada planta está apresentada na Figura 3.10. Os eventos controláveis, pertencentes a Σ_c , são representados pelos números ímpares e estão relacionados à inicialização de uma tarefa, enquanto os eventos não controláveis, pertencentes a Σ_{uc} , são representados pelos números pares e estão relacionados à finalização de uma tarefa dos equipamentos da Figura 3.9.

A lógica de controle é adicionada por meio das especificações dos *buffers* para permitir o maior grau de liberdade para a produção das peças sem, contudo, permitir que ocorram *overflow* e *underflow*. A modelagem das especificações está apresentada na Figura 3.11.

Com as plantas e especificações, por meio da TCS, é possível encontrar o supervisor monolítico para esse sistema, que possui 45.504 estados e 200.124 transições. Para realizar as avaliações temporais das simulações, são utilizados os intervalos de tempo entre a ocorrência de um evento controlável e seu respectivo evento não controlável, conforme apresentados na Tabela 3.3, de Alves et al. (2016).

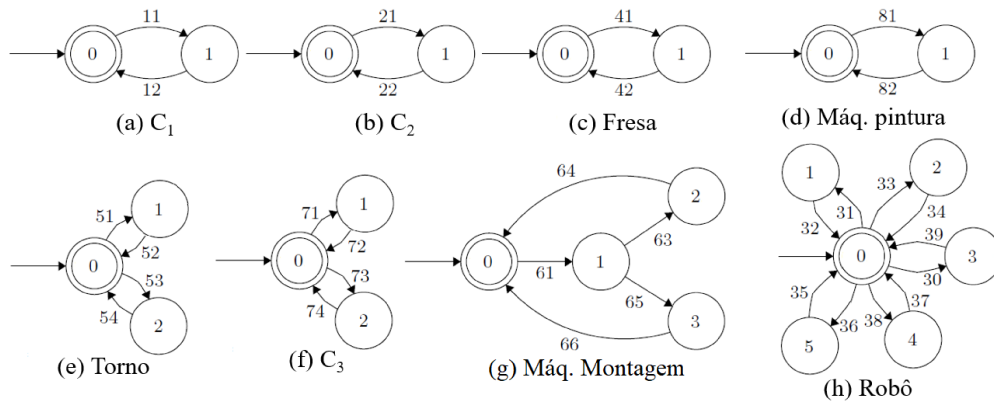


Figura 3.10: Modelagem das máquinas do Sistema Flexível de Manufatura.

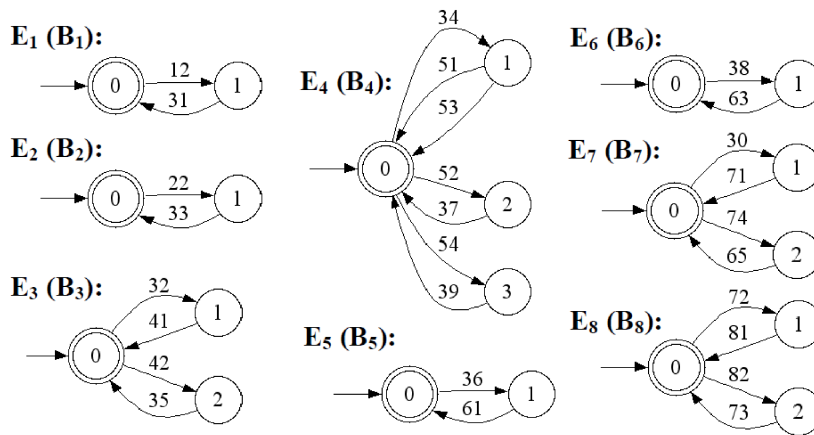


Figura 3.11: Especificações referentes ao *overflow* e *underflow* dos *buffers* do Sistema Flexível de Manufatura.

A produção é um conjunto de três subprodutos, cada um deles conforme a sequência definida abaixo. Os eventos podem ser intercalados para gerar uma sequência que tenha menor *makespan*, desde que obedeçam à ordem de cada sequência. O Produto A é a junção da *base* com o *pino_A* (cônico sem pintura) e o Produto B, a junção da *base* com o *pino_B* (cilíndrico pintado).

- *base* = (11, 12), (31, 32), (41, 42), (35, 36), (61)
- *pino_A* = (21, 22), (33, 34), (51, 52), (37, 38), (63, 64)
- *pino_B* = (21, 22), (33, 34), (53, 54), (39, 30), (71, 72), (81, 82), (73, 74), (65, 66)

3.5.2 Simulação de Sistemas a Eventos Discretos

Simulação pode ser definida como um procedimento em que o modelo de um sistema é avaliado numericamente, e os dados gerados são utilizados para estimar variáveis de interesse. A simulação tem a vantagem de evitar a necessidade da implantação do sistema para coletar as variáveis e, também, a possibilidade de testar diferentes cenários para análise, por meio de modificações em *software* e não em *hardware* (Cassandras & Lafortune, 2009).

Tabela 3.3: Intervalo de tempo entre os pares de eventos.

Eventos controláveis	Eventos não controláveis	Intervalo de tempo [s]
11	12	26
21	22	26
31	32	22
33	34	20
35	36	17
37	38	25
39	30	21
41	42	31
51	52	39
53	54	33
63	64	27
65	66	27
71	72	26
73	74	26
81	82	25

Ainda que autômatos não possuam nenhuma informação temporal em sua estrutura original, é possível incluir tais informações em estruturas auxiliares, tornando possível a avaliação numérica. A Figura 3.12 apresenta o diagrama de simulação clássica de SED, que tem os seguintes elementos:

- Estado: lista que armazena todas as variáveis de estado;
- Tempo: variável que armazena o tempo de simulação;
- Lista de Eventos Agendados: lista que armazena todos os eventos associados ao tempo em que cada um deve ocorrer;
- Rotina de Inicialização: rotina que inicializa as estruturas de dados no início da simulação;
- Rotina de Atualização de Tempo: rotina que identifica o próximo evento que deve ocorrer, com base na Lista de Eventos Agendados, e atualiza o tempo de simulação para o tempo de ocorrência desse evento;
- Rotina de Atualização de Estado: rotina que atualiza o estado com base na ocorrência do evento;
- Rotina de Remoção de Inactibilidades: rotina que remove da Lista de Eventos Agendados os eventos que não estão habilitados no estado atual;
- Rotina de Geração de Variações Aleatórias: rotina que gera os tempos de ocorrência dos eventos com base em uma dada distribuição de probabilidade;

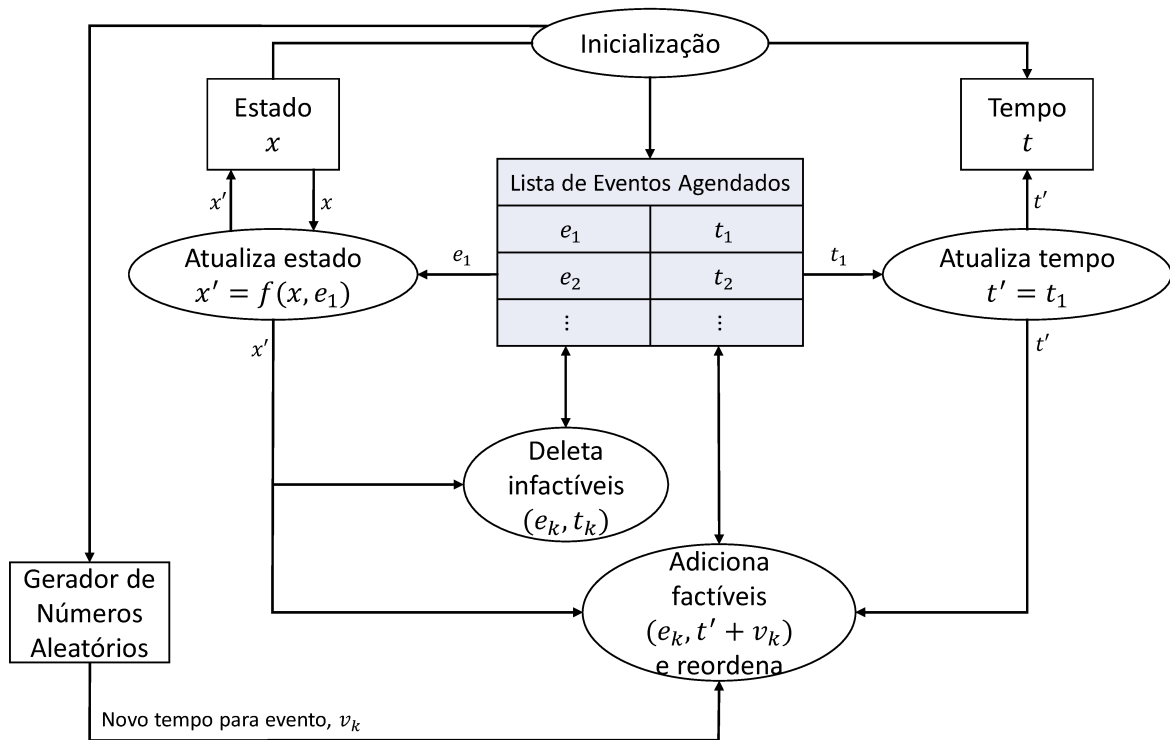


Figura 3.12: Diagrama de simulação de Sistemas a Eventos Discretos

- Programa Principal: responsável pela coordenação de todos os componentes. Primeiramente chama a Rotina de Inicialização e depois chama repetidamente as demais rotinas até o critério de parada definido;

Neste trabalho, um esquema de simulação é utilizado, adaptado de Cassandras & Lafor-tune (2009), como o objetivo de verificar a factibilidade temporal das sequências de eventos ao longo da otimização e de avaliar o *makespan* das sequências encontradas.

3.5.3 UltraDES

O UltraDES é uma biblioteca orientada a objetos implementada em C# e baseada na plataforma *.NET Framework*. A biblioteca contém estruturas de dados e algoritmos para a modelagem, análise e controle de sistemas a eventos discretos. Além disso, seu desenvolvimento foi focado em usabilidade e expansibilidade de forma que um novo usuário não teria dificuldades em utilizar a ferramenta, e, quando necessário, novos algoritmos poderiam ser facilmente implementados (Alves et al., 2017).

A ferramenta permite a representação de:

- estados: definidos por suas características básicas como *alias* e marcação (estado marcado ou não marcado) e atribuídos a uma variável no momento da sua criação. Um exemplo para a criação do estado marcado *s1* é:

```
var s1 = new State("s1", Marking.Marked);
```

- eventos: são definidos por características como *alias* e controlabilidade (evento controlável ou não-controlável). São igualmente atribuídos a uma variável na sua criação, conforme o exemplo para a criação do evento não-controlável *e1*:

```
var e1 = new Event("e1", Controllability.Uncontrollable);
```

- transições: são definidas por três elementos sendo: um estado de origem, um evento que dispara a transição e um estado de destino. O exemplo a seguir mostra a criação de uma transição *t* que vai do estado *s1* ao estado *s2* por meio do evento *e1*:

```
var t = new Transition(s1,e1,s2);
```

- autômatos finitos determinísticos: são definidos por uma lista de transições, o estado inicial e o *alias*. Um autômato de dois estados *s1* e *s2*, sai do estado inicial *s1* com o evento *e1* e retorna com o evento *e2* e é criado da seguinte forma:

```
var G = new DeterministicFiniteAutomaton(new[] {
    new Transition(s1, e1, s2),
    new Transition(s2, e2, s1)}, s1, "G");
```

A classe *DeterministicFiniteAutomaton* possui propriedades que retornam algumas informações sobre o autômato criado, como listas dos estados, dos estados marcados, das transições, o estado inicial, o nome, entre outros.

Estão definidas, também, algumas operações que podem ser feitas em um único autômato, como parte acessível, parte coacessível, *trim*; entre dois autômatos, como a composição paralela; e para obtenção de um supervisor, seja ele monolítico ou modular local.

3.5.4 Minimização de Tempo Heurístico

Há, na literatura uma heurística chamada de Minimização de Tempo Heurístico (MTH) proposta por Alves et al. (2019), que busca resolver o problema de minimização do *makespan* e utiliza como universo de busca, a estrutura de um supervisor.

Essa heurística tem como princípio priorizar os eventos controláveis em detrimento dos não-controláveis, de forma que o número de caminhos analisados seja significativamente reduzido. Outro procedimento realizado é que, sempre que um caminho chega a um estado já visitado, verifica-se aquele que tem o menor tempo, excluindo o de maior tempo.

Em outras palavras, para cada estado do supervisor, verificam-se as transições cujos eventos são controláveis e, para cada transição, vai para o estado de destino, armazenando as informações necessárias até aquele estado. Somente quando não houver, em um estado, nenhum evento controlável habilitado é que são executadas as transições cujos eventos são não-controláveis. Esse procedimento se repete até que se chegue ao estado marcado. A heurística é apresentada no Algoritmo 1.

A inicialização da heurística acontece com: $path[(q_0, a_0, 0)]$, recebendo o caminho até o estado inicial (ϵ), onde q_0 é o estado inicial e a_0 é o agendador de eventos inicial (adaptado de Cassandras & Lafortune (2009)) (linha 1); e $time[(q_0, a_0, 0)]$ recebendo o tempo gasto para chegar nesse mesmo estado (0) (linha 2). Uma estrutura de fila, *F*, recebe o estado inicial (linha 4) e, enquanto essa fila possui elementos, o seu primeiro estado é retirado da fila (linha 7). Verifica-se, então, se existem eventos controláveis factíveis e que não geram acréscimo de

Algoritmo 1: Minimização de Tempo Heurístico

```

1  $path[(q_0, a_0, 0)] \leftarrow \epsilon$ 
2  $time[(q_0, a_0, 0)] \leftarrow 0$ 
3
4  $F \leftarrow (q, a, 0)$ 
5
6 enquanto  $F \neq \emptyset$  faça
7    $(q, a, i) \leftarrow F$ 
8   se  $\Gamma(q) \cap \Sigma_c \neq \emptyset$  e  $\forall \sigma \in \Gamma(q) \cap \Sigma_c, a[\sigma] = 0$  então
9      $E \leftarrow \Gamma(q) \cap \Sigma_c$ 
10  senão
11     $t_{min} \leftarrow \min_{\sigma \in \Gamma(q) \cap \Sigma_u} a[\sigma]$ 
12     $E \leftarrow \Gamma(q) \cap \{\sigma_t : \sigma_t \in \Sigma \wedge a[\sigma_t] \leq t_{min}\}$ 
13  para cada evento  $\sigma$  em  $E$  faça
14     $v \leftarrow \delta(q, \sigma)$ 
15     $t \leftarrow t + a[\sigma]$ 
16     $a_n \leftarrow update(a, \sigma)$ 
17    se  $t = \infty$  então
18       $break$ 
19    se  $F \not\supseteq (v, a_n, i + 1)$  então
20       $F \leftarrow (v, a_n, i + 1)$ 
21     $t_{tot} \leftarrow time[(q, a, i)] + t$ 
22    se  $\nexists time[(v, a, i)]$  ou  $t_{tot} < time[(v, a_n, i + 1)]$  então
23       $path[(v, a_n, i + 1)] \leftarrow path[(q, a, i)]\sigma$ 
24       $time[(v, a_n, i + 1)] \leftarrow t_{tot}$ 

```

tempo (linha 8). Caso isso seja verdadeiro, esse conjunto de eventos é atribuído à estrutura E (linha 9). Caso contrário, determina-se o menor tempo para a ocorrência de um evento não controlável e atribui-se a E o conjunto de todos os eventos que gerem uma variação de tempo menor que esse valor (linhas 11-12).

Cada um dos eventos contidos em E é avaliado. Caso a transição seja temporalmente factível, são atualizados o estado atual, o tempo gasto até esse estado ($t + a[\sigma]$, onde $a[\sigma]$ é o tempo restante para a ocorrência do evento σ pelo agendador a) e o agendador de eventos pela função $update(a, \sigma)$ (linhas 14-16). Se a estrutura F não contiver o novo estado, ele é adicionado à fila (linha 20). Caso o novo estado não exista em $time$ ou o tempo acrescido seja menor que o tempo total até aquele estado, o novo evento é incluído em $path$, e o tempo é atualizado (linhas 23-24).

O Algoritmo 1 está implementado no UltraDES e é utilizado, neste trabalho, para encontrar sequências otimizadas para o problema de escalonamento *job shop* com bloqueio (Seção 4.5) e como parte da solução proposta para a otimização de sequências de produção com demandas estocásticas (Seção 5.2).

3.5.5 Descrição do Computador Utilizado

Para encontrar os resultados do Capítulo 4, um computador com sistema operacional Windows 10 de 64 bits, memória RAM de 64,00 GB e processador Intel Xeon E5-2650 / 2,00 GHz foi utilizado. Já para os resultados do Capítulo 5 foi utilizado um computador com sistema operacional Windows 10 de 64 bits, memória RAM de 8,00 GB e processador Intel Core i5-4210U / 1,70 GHz. Em todos os casos utilizou-se a IDE Microsoft Visual Studio Community 2019 e a biblioteca UltraDES (Alves et al., 2017), específica para a modelagem, análise e controle de SED. A linguagem de programação escolhida foi C#.

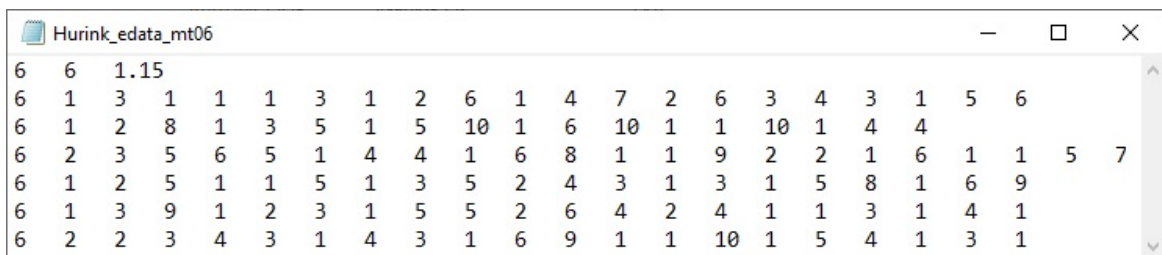
Capítulo 4

Tradução Automática do Problema

O presente capítulo trata da metodologia desenvolvida para a obtenção dos modelos dos autômatos que representam um problema de escalonamento *job shop* com bloqueio, apresentando as etapas para alcançar tal resultado.

4.1 Interpretação das Instâncias

A ordem das máquinas que processam um *job* e a atribuição dos tempos associados a elas muitas vezes é representada na forma da Tabela 3.1 (Seção 3.1.1) para a fácil visualização das informações. Para o uso desses dados por algoritmos, no entanto, o formato de uma tabela nem sempre é o mais eficiente, não só pelo seu tamanho, mas também pela quantidade de campos que não contêm valores relevantes para o cálculo da solução, como é o caso dos campos com valor ∞ . Por esse motivo, são encontradas nas bases de dados da literatura instâncias em formato de texto, como apresentado na Figura 4.1. Tais instâncias são encontradas em formatos de arquivos na extensão *.fjs*.



6	6	1.15																				
6	1	3	1	1	1	3	1	2	6	1	4	7	2	6	3	4	3	1	5	6		
6	1	2	8	1	3	5	1	5	10	1	6	10	1	1	10	1	4	4				
6	2	3	5	6	5	1	4	4	1	6	8	1	1	9	2	2	1	6	1	1	5	7
6	1	2	5	1	1	5	1	3	5	2	4	3	1	3	1	5	8	1	6	9		
6	1	3	9	1	2	3	1	5	5	2	6	4	2	4	1	1	3	1	4	1		
6	2	2	3	4	3	1	4	3	1	6	9	1	1	10	1	5	4	1	3	1		

Figura 4.1: Exemplo de instância

A primeira linha do arquivo indica as dimensões do problema, sendo que o primeiro campo representa o número de *jobs* n e o segundo, o número de máquinas m . O terceiro campo é opcional e indica a média do número de máquinas capazes de processar cada operação, como um parâmetro da flexibilidade do problema. A média um indica o caso menos flexível, em que cada operação pode ser processada por apenas uma máquina. Médias maiores do que um indicam problemas mais flexíveis. Se esse valor é igual a dois, por exemplo, trata-se um problema em que, na média, cada operação tem duas opções de máquinas para ser processada.

No exemplo de instância apresentado na Figura 4.1 os valores da primeira linha são:

6 6 1.15

Isso indica que o problema, nesse caso, tem 6 *jobs*, 6 máquinas e uma média de 1,15 máquinas por operação.

Em seguida, há n linhas que contêm, cada uma, os dados de um *job*. O primeiro valor da linha indica a quantidade h de operações daquele *job*. Há, então, h conjuntos de dados referente a cada operação. O primeiro campo de cada conjunto 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 relacionadas com o *job* J_1 , conforme apresentado a seguir com as suas respectivas indicações:

h	$O_{1,1}$	$O_{1,2}$	$O_{1,3}$	$O_{1,4}$	$O_{1,5}$	$O_{1,6}$
6	1 3 1	1 1 3	1 2 6	1 4 7	2 6 3 4 3	1 5 6
	r=1 dupla	r=1 dupla	r=1 dupla	r=1 dupla	r=2 dupla	dupla

Nesse *job*, a primeira operação ($O_{1,1}$) é executada na máquina três e tem o tempo de processamento de uma unidade. A segunda operação ($O_{1,2}$) é executada na máquina um e tem o tempo de processamento de três unidades. Somente a operação $O_{1,5}$ pode ser processada por duas máquinas, sendo elas as máquinas quatro e seis, ambas com tempo de processamento de três unidades de tempo. As demais operações apresentam apenas uma opção. A atribuição das máquinas e tempos para esse *job*, conforme o procedimento de interpretação apresentado, está na Tabela 4.1.

Tabela 4.1: Atribuição dos tempos de processamento de um *job* conforme exemplo

$p_{i,j}$		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	∞

As demais linhas do arquivo são avaliadas utilizando o mesmo procedimento.

4.2 FJSSP Modelado como Autômato

Este trabalho se propõe a desenvolver um método de tradução automática de um problema de *job shop* em autômatos finitos determinísticos. A tradução é feita a partir da leitura de um arquivo como o mostrado na Figura 4.1 e da construção de um autômato para cada *job* e para cada máquina.

A modelagem de um problema de FJSSP em autômatos se dá seguindo os passos e notações a seguir:

- cada operação é composta por um par de eventos de início e fim;
 - 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 *ijk*, onde:
 - * *i* se refere ao *job*;
 - * *j* se refere à operação;
 - * *k* se refere à máquina.
- a linguagem marcada desse autômato corresponde à sequência das operações do *job*.
- os estados são nomeados como *sd.x*, onde:
 - *d* é a profundidade do autômato;
 - *x* é um número natural para diferenciar estados de mesma profundidade.

A Figura 4.2 mostra o exemplo de um *job* modelado conforme descrito acima. No autômato, a execução de um par de eventos *a* e *b* com o mesmo índice representa a conclusão de uma operação do *job*. Assim, os estados de profundidade ímpar indicam que a operação está sendo processada, enquanto os de profundidade par indicam que a próxima operação aguarda para ser admitida em alguma máquina. A exceção existe somente para o estado de maior profundidade em que o *job* está totalmente processado sendo liberado do sistema e, por isso, esse estado é marcado.

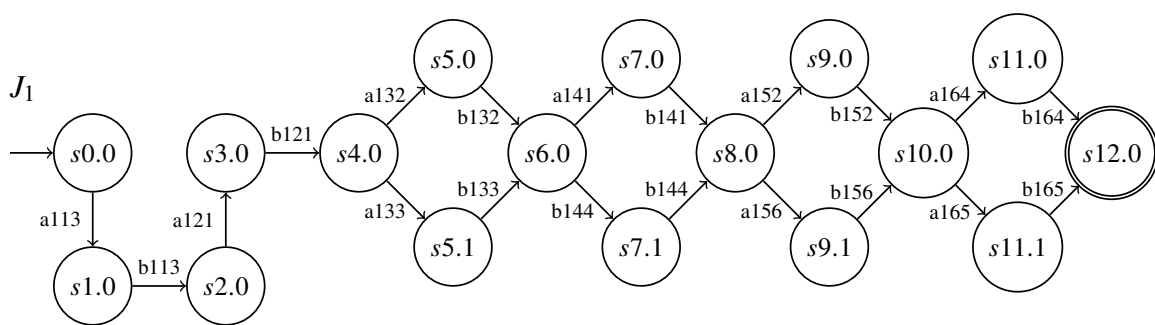


Figura 4.2: Exemplo da modelagem de um *job* em autômato

Das seis operações desse *job*, as duas primeiras podem ser processadas por apenas uma máquina, enquanto as outras quatro podem ser processadas por duas máquinas. No caso do estado *s4.0*, por exemplo, em que duas operações já foram processadas, a terceira pode ser admitida tanto na máquina dois (*a132*), quanto na máquina três (*a133*). No autômato, a execução desses eventos leva a caminhos distintos, mas de mesma profundidade. A

diferenciação entre os estados é feita por um índice que varia conforme a quantidade de máquinas capazes de processar determinada operação. No caso do exemplo apresentado, as transições do estado $s4.0$ podem levar para o estado $s5.0$ ou para o $s5.1$. Independentemente da quantidade de caminhos possíveis para a inicialização de uma operação, os eventos de término levam para um único estado, em que o *job* aguarda ser admitido na máquina subsequente.

A modelagem das máquinas visa a assegurar que uma máquina processe apenas uma operação de cada vez. Por isso, os autômatos referentes a elas têm apenas dois estados: um para a máquina ociosa ($s0$) e outro para a máquina trabalhando ($s1$). A máquina vai do estado $s0$ para o estado $s1$ com qualquer evento que a utilize, independentemente do *job* ou da operação, e retorna com o término do processamento.

A Figura 4.3 mostra um exemplo para a máquina 1, cujas transições estão terminadas em 1. A ocorrência do evento $a121$, por exemplo, que inicia a segunda operação do *job* 1 na máquina 1, leva o autômato do estado $s0$ para o estado $s1$. Nesse estado, a quinta operação do *job* 2, a terceira operação do *job* 3 e a primeira operação do *job* 4 estão impedidas de serem iniciadas. Ao término do processamento dessa operação, retorna-se para o estado $s0$, com o evento $b121$, habilitando novamente os eventos que utilizam aquela máquina. Os modelos dos *jobs* e das máquinas para uma instância de FJSSP são apresentados no Apêndice A.

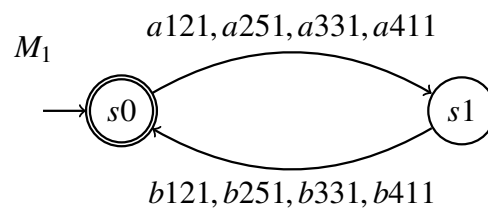


Figura 4.3: Exemplo da modelagem de uma especificação

Na modelagem de sistemas de manufatura, em geral, define-se que as máquinas são as plantas a serem controladas. No caso da modelagem proposta no presente trabalho, foi definido que os *jobs* são as plantas, pois modelam todas as combinações para a produção desejada, enquanto as máquinas são restrições para o problema, já que não podem processar mais de uma operação ao mesmo tempo. Com essas definições, gerou-se o supervisor para algumas instâncias dessa classe de problemas.

Observou-se que o comportamento em malha fechada, modelado pelo supervisor encontrado, é controlável. Ou seja, esse supervisor já é a máxima sublinguagem controlável ou, utilizando a notação apresentada, $K = SupC(K, G)$. Isso se deve ao fato de as restrições impostas aos problemas em questão serem relativamente simples do ponto de vista lógico, uma vez que cada máquina não depende de outras para finalizar uma tarefa. A dependência existe somente para iniciar uma tarefa, pois deve respeitar a ordem de precedência. A verificação de que $K = SupC(K, G)$ é feita comparando o número de estados e de transições dos dois autômatos, que são sempre iguais para esse caso.

4.3 FJSSP com Bloqueio Modelado como Autômato

A modelagem do problema de FJSSP foi um passo intermediário para a obtenção do modelo de FJSSP com bloqueio. Embora a abordagem da TCS resolva tanto o problema clássico quanto o problema com bloqueio, esse último é o problema de interesse deste trabalho, visto que mais restrições são impostas, aumentando a relevância dos resultados encontrados. Além disso, o problema com bloqueio é mais difícil de ser resolvido, uma vez que, por ter mais restrições, o comportamento em malha fechada do sistema resulta em um autômato maior que aquele obtido quando o problema não tem bloqueio. Foi considerada a configuração de bloqueio sem troca (BNS).

A principal diferença da modelagem para esse problema em relação ao anterior é observada no modelo das máquinas. Isso pelo fato de que nesse novo cenário, quando uma máquina finaliza sua tarefa, deve aguardar a admissão do *job* em outra máquina para, então, estar disponível novamente. Isso significa que é necessário acrescentar um estado ao no modelo, que será chamado de s_2 , e indicará que a máquina realizou sua tarefa e aguarda ser liberada.

A Figura 4.4 mostra o modelo proposto. Uma máquina no estado inicial s_0 está disponível. Ao iniciar o processamento de um *job* (evento a_1), vai para o estado s_1 e, ao finalizá-lo (evento b_1), vai para o estado s_2 . Nesse estado, dois caminhos são possíveis: um deles é a operação subsequente ser admitida pela mesma máquina (evento a_2), retornando ao estado s_1 ; o outro é a máquina ser liberada quando o *job* é admitido em outra máquina (evento a_3). Um caso particular deve ser considerado, referente ao cenário em que a máquina processa a última operação de um *job* (evento b_2). Nesse caso, a máquina é imediatamente liberada, pois o *job* é retirado do sistema uma vez que foi integralmente processado. Do ponto de vista do autômato, isso representa uma transição do estado s_1 (trabalhando) diretamente para o estado s_0 (liberada).

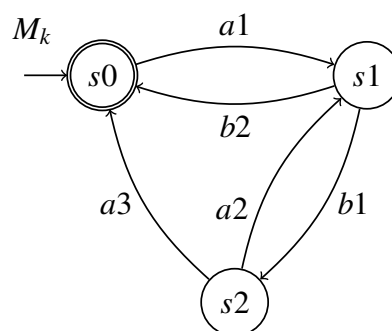


Figura 4.4: Exemplo da modelagem de uma máquina em autômato

Outra diferença diz respeito ao nome dos eventos. Observou-se que a indexação ijk , descrita anteriormente, não é suficiente para distinguir os caminhos nos autômatos de forma que as próprias especificações são bloqueantes. Para ilustrar esse problema, considera-se um exemplo em que o *job* 1 é definido por

h	$O_{1,1}$			$O_{1,2}$				$O_{1,3}$			
3	1	2	8	2	3	5	1	4	1	1	4

Esses dados indicam que o *job* tem três operações, a serem processadas nas máquinas 2, 1 ou 3, 1, nessa ordem. De acordo com o modelo proposto, o autômato resultante para as máquinas 1, 2 e 3 estão apresentadas na Figura 4.5, sendo $M_1 = (_, _, \delta_1, _, _)$, $M_2 = (_, _, \delta_2, _, _)$ e $M_3 = (_, _, \delta_3, _, _)$.

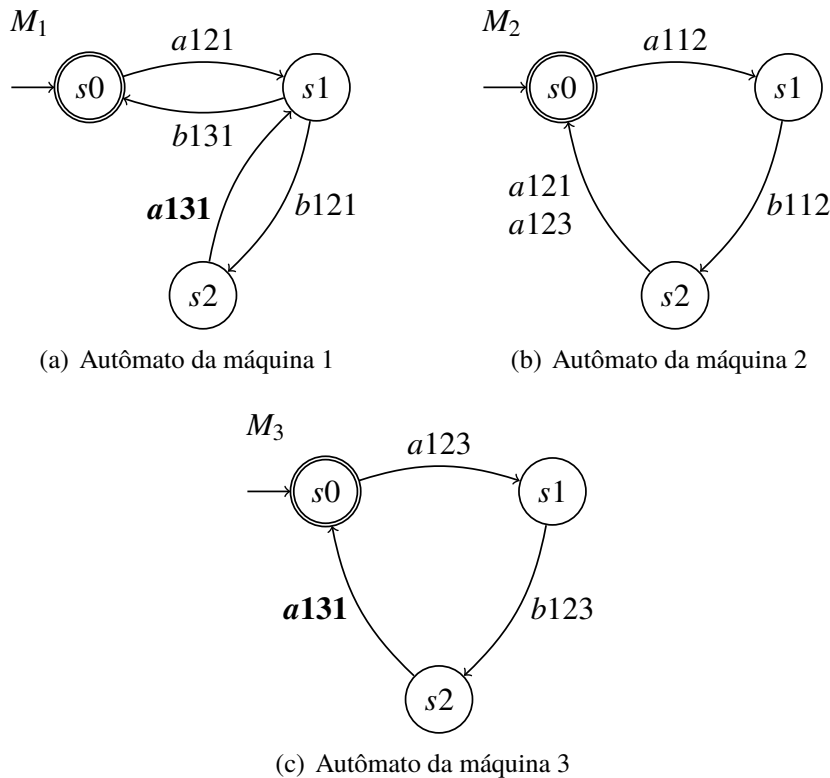


Figura 4.5: Autômatos das máquinas para o exemplo proposto

As duas sequências de eventos possíveis para esse *job* são:

- (a) $a_{112}, b_{112}, a_{121}, b_{121}, a_{131}, b_{131}$
- (b) $a_{112}, b_{112}, a_{123}, b_{123}, a_{131}, b_{131}$.

A evolução dos estados dos três autômatos em função da execução da sequência (a) está apresentada na Tabela 4.2. Após o quarto evento, os autômatos das máquinas 1, 2 e 3 estão nos estados s_2 , s_0 e s_0 , respectivamente. O próximo evento a ser executado é o a_{131} que não pode acontecer, visto que a transição $\delta_3(s_0, a_{131})$ não é factível.

O mesmo acontece executando-se os eventos da sequência (b), cuja progressão é apresentada na Tabela 4.3, em que a execução do quinto evento $\delta_1(s_0, a_{131})$ não é factível. Nesse cenário, ao tentar encontrar o comportamento em malha fechada do sistema com essas especificações, chega-se a um supervisor totalmente bloqueante, ou seja, que desabilita todos os eventos controláveis.

Tabela 4.2: Progressão dos estados em função dos eventos da sequência (a)

	q_0	a112	b112	a121	b121	a131	b131
M_1	s0	s0	s0	s1	s2	s1	s0
M_2	s0	s1	s2	s0	s0	s0	s0
M_3	s0	s0	s0	s0	s0	∅	

Tabela 4.3: Progressão dos estados em função dos eventos da sequência (b)

	q_0	a112	b112	a123	b123	a131	b131
M_1	s0	s0	s0	s0	s0	∅	
M_2	s0	s1	s2	s0	s0	s0	s0
M_3	s0	s0	s0	s1	s2	s1	s0

A alternativa encontrada para solucionar esse problema foi incluir um índice a mais no nome do evento, indicando a máquina imediatamente anterior pela qual o *job* passou. Para a primeira operação de cada *job*, esse índice tem valor 0. A Figura 4.6 mostra como o mesmo exemplo é modelado considerando-se a nova nomenclatura dos eventos. As duas sequências de eventos renomeados possíveis são:

(a) $a1120, b1120, a1212, b1212, a1311, b1311$

(b) $a1120, b1120, a1232, b1232, a1313, b1313$

Qualquer que seja a sequência escolhida não há, em qualquer autômato, nenhuma transição que não seja factível.

No que diz respeito à modelagem dos *jobs*, os procedimentos são similares aos apresentados na seção anterior, sendo a única mudança relacionada à nova nomenclatura dos eventos. Como um evento é nomeado de acordo com a máquina em que a última operação foi processada, a linguagem do autômato que modela o *job* deve conter os eventos dos diferentes caminhos. O autômato do exemplo dado é apresentado na Figura 4.7.

Conforme indicado nessa figura, a transição do estado $s4.0$ para o estado $s5.0$ pode ocorrer tanto com o evento $a1311$ quanto com $a1313$, ou seja, há uma duplicação do evento $a131$ com a inclusão ao final o índice referente à máquina imediatamente anterior à operação a ser processada.

Os modelos dos *jobs* e das máquinas para uma instância de FJSSP com bloqueio são apresentados no Apêndice B.

4.3.1 Modelagem do SFM

Com o intuito de facilitar o entendimento da relação entre autômatos e FJSSP, o SFM, apresentado na Seção 3.5.1, será utilizado. Embora esse sistema se caracterize como um problema de escalonamento *flow shop*, ele será analisado como um problema de *job shop*, já que o primeiro é um caso particular do segundo.

Para adequar a nomenclatura da máquinas, elas serão chamadas de M_1 (C_1), M_2 (C_2), M_3 (C_3), M_4 (robô), M_5 (fresa), M_6 (torno), M_7 (máquina de pintura), M_8 (máquina de

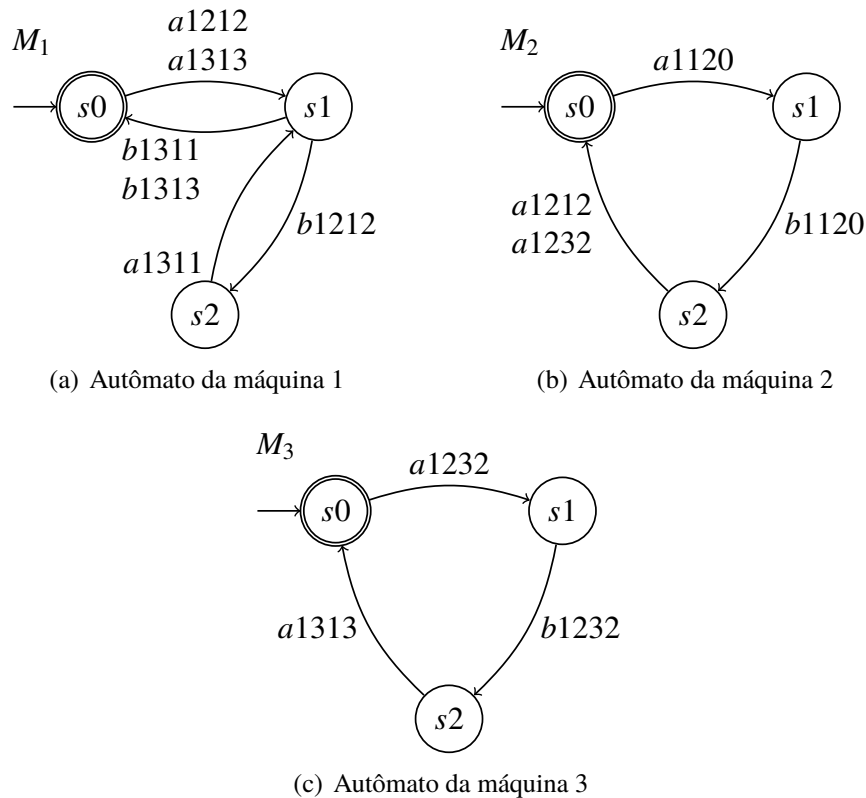


Figura 4.6: Autômatos das máquinas para o exemplo proposto com os eventos renomeados

montagem). Os *buffers* unitários serão suprimidos de forma a considerar o sistema com *buffers* inexistentes. Os produtos A e B serão chamados de job_1 e job_2 , respectivamente.

A representação desse problema no formato textual, conforme mostrado na Figura 4.1, é apresentado na Figura 4.8, utilizando os tempos de processamento da Tabela 3.3. O problema original permite a execução paralela da base e do pino de cada produto, mas como o problema de *job shop* determina a relação de precedência entre as operações, uma adaptação foi feita, considerando-se que a base é produzida antes do pino. Isso, no entanto, não afeta o objetivo de ilustrar a relação entre as duas representações.

Partindo dessa instância e executando os procedimentos anteriormente apresentados, é possível obter o modelo dos *jobs*. O modelo do job_1 é apresentado na Figura 4.9.

O modelo das máquinas, considerando o problema com bloqueio, também é obtido e o autômato da máquina M_1 está apresentado na Figura 4.10. Os modelos do outro *jobs* e das demais máquinas são apresentados no Apêndice C.

É possível observar, a partir do exemplo apresentado, que o procedimento de tradução desenvolvido também pode ser aplicado a problemas conhecidos da área de SED quando esses são apresentados no formato textual padrão dos problemas de FJSSP.

4.4. OBTENÇÃO DO SUPERVISOR PARA O PROBLEMA DE FJSSP COM BLOQUEIO35

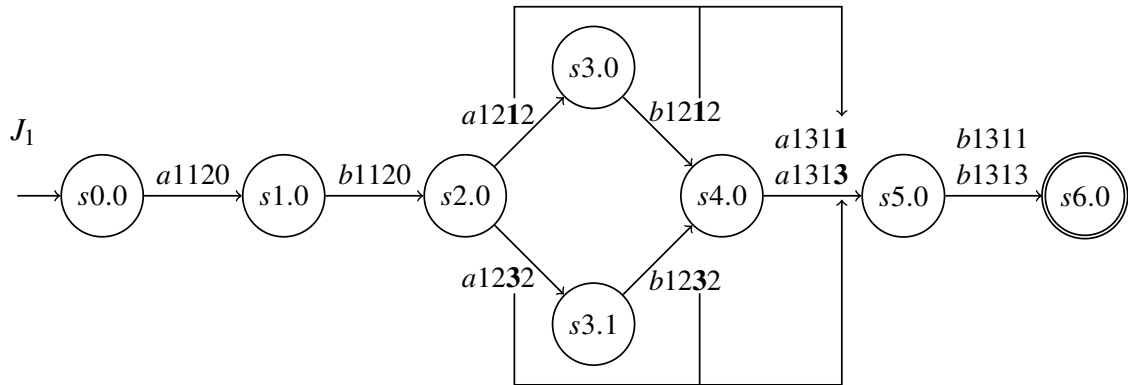


Figura 4.7: Exemplo da modelagem de um *job* em autômato com os eventos renomeados

fsm																																				
2	8																																			
9	1	1	26	1	4	22	1	5	31	1	4	17	1	2	26	1	4	20	1	6	39	1	4	25	1	8	27									
12	1	1	26	1	4	22	1	5	31	1	4	17	1	2	26	1	4	20	1	6	33	1	4	21	1	3	26	1	7	25	1	3	26	1	8	27

Figura 4.8: Instância para o problema do SFM

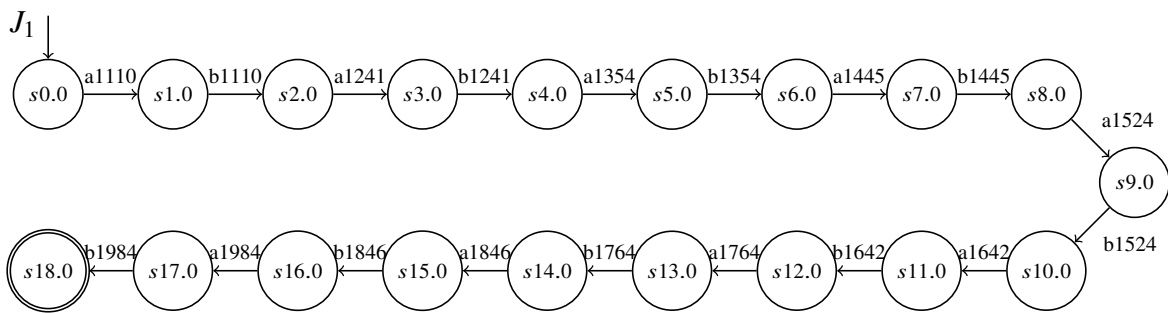


Figura 4.9: Exemplo da modelagem do *job*₁ do SFM

4.4 Obtenção do Supervisor para o Problema de FJSSP com Bloqueio

Os procedimentos para modelar os *jobs* e as máquinas foram implementados em um programa, conforme apresentado no Algoritmo 2, e os autômatos foram gerados utilizando-se a biblioteca UltraDES. Com as plantas (modelo dos *jobs*) e especificações (restrições das máquinas), o supervisor monolítico para cada instância foi gerado.

O número de estados e de transições de cada supervisor são apresentados na Tabela 4.4. A tabela mostra apenas as instâncias para as quais foi possível encontrar uma solução. O valor $|M_{j,k}|$ indica a média de máquinas por operação (flexibilidade).

Embora o tamanho dos supervisores encontrados seja relativamente grande, todo o universo de busca está contido nele. Isso significa que todo e qualquer caminho escolhido do estado inicial até o estado marcado do supervisor representa uma sequência de produção factível, livre de bloqueios e que atende às restrições do problema.

Outras instâncias foram submetidas ao mesmo procedimento, mas não foi possível com-

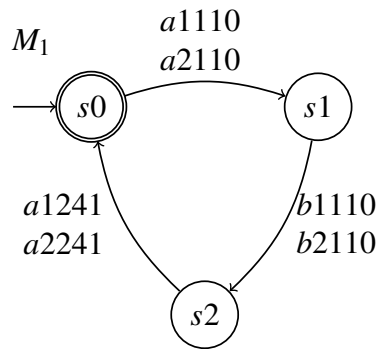


Figura 4.10: Exemplo da modelagem de M_1 do SFM

putar o seu 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 em dois exemplos: 1) comparando as instâncias “Hurink_sdata_la01.fjs” e “Hurink_sdata_car1.fjs” em que o aumento de 10% no número de *jobs* provoca um aumento no número de estados de $\approx 1.150\%$, e no número de transições, de $\approx 1.230\%$; 2) comparando as instâncias “Hurink_sdata_mt06.fjs” e “Hurink_sdata_car5.fjs” em que o aumento de $\approx 66,6\%$ no número de *jobs* provoca um aumento no número de estados de $\approx 71.990\%$, e no número de transições, de $\approx 84.180\%$.

É possível observar, também, que o tamanho do supervisor aumenta não somente em função do aumento da dimensão do problema, mas também da sua flexibilidade. As quatro primeiras instâncias da tabela, por exemplo, são o mesmo problema (mt06) variando-se apenas a sua flexibilidade, conforme explicado na Seção 2.4. O número de estados e de transições, no entanto, cresce de forma não linear.

O supervisor para o problema apresentado na Seção 3.5.1 utilizando os autômatos dos *jobs* e das especificações modelados na Seção 4.3.1 também foi gerado. O tamanho da estrutura encontrada foi de 346 estados e 591 transições, significativamente menor que o tamanho do supervisor desenrolado¹ para a produção de um Produto A e um Produto B, que é de 3.550 estados e 10.440 transições. Essa diferença se deve ao fato de que uma modelagem define a sequência de produção, enquanto a outra contempla todas as combinações de eventos que levam à produção desejada.

4.5 Solução do Problema de FJSSP com uma Heurística

Como foi possível encontrar o espaço de estados para as instâncias buscou-se encontrar uma solução para elas. Visto que o tamanho dos supervisores é relativamente grande, um algoritmo de busca exata levaria um tempo impraticável para encontrar uma solução e, por isso, optou-se por utilizar uma heurística da literatura.

A heurística escolhida foi a MTH, já apresentada na Seção 3.5.4. O principal motivo

¹Os supervisores, em geral, apresentam ciclos e têm a linguagem marcada infinita. Por isso, uma das formas de se obterem todas as cadeias de eventos que levam à produção de uma determinada quantidade de produtos é fazer a composição paralela entre o supervisor e a especificação de produção. Essa especificação é um autômato linear cujas transições são os eventos que representam a finalização de um produto e cuja profundidade equivale à quantidade da produção.

Algoritmo 2: Geração do supervisor a partir de uma instância de FJSSP

Entrada: instância_FJSSP
Saída: supervisor

- 1 $n \leftarrow$ número de jobs
- 2 $m \leftarrow$ número de máquinas
- 3
- 4 **para cada** job_n **faça**
- 5 $estado \leftarrow 0$
- 6 **para cada** $operação_h$ **faça**
- 7 $x \leftarrow 0$
- 8 **para cada** $máquina_m$ **faça**
- 9 adiciona [$estado.0 \rightarrow a_{ijkl} \rightarrow (estado + 1).x$] ao job_n
- 10 adiciona [$(estado + 1).x \rightarrow b_{ijkl} \rightarrow (estado + 2).0$] ao job_n
- 11 $x \leftarrow x + +$
- 12 adiciona [$0 \rightarrow a_{ijkl} \rightarrow 1$] à maq_m
- 13 adiciona [$1 \rightarrow b_{ijkl} \rightarrow 0$] à maq_m
- 14 $estado \leftarrow estado + 2$
- 15 $G_set \leftarrow$ gera autômato job_n
- 16 **para cada** maq_m **faça**
- 17 $E_set \leftarrow$ gera autômato maq_m
- 18 supervisor \leftarrow gera_supervisor_monolitico (G_set, E_set)

Tabela 4.4: Tamanho dos supervisores obtidos

Instância	Dimensão	$ M_{j,k} $	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

dessa escolha foi o que a heurística utiliza como espaço de busca e a qualidade dos resultados. Algumas modificações foram feitas para adequar às características do problema tratado.

4.5.1 Resultados

A heurística foi aplicada para todas as instâncias em que foi possível encontrar o supervisor. Os resultados encontrados estão apresentados na Tabela 4.5. A primeira e a segunda colunas identificam a instância. A terceira e a quarta colunas indicam o tamanho do supervisor encontrado. As duas colunas seguintes apresentam o tempo de processamento e o valor de *makespan* encontrado ao se aplicar a heurística sobre o supervisor. As duas últimas colunas apresentam valores de *makespan* encontrados na literatura. É possível observar a presença de algumas células vazias, indicando a inexistência de resultados conhecidos para as respectivas instâncias.

Tabela 4.5: 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					

Em alguns casos só foi possível encontrar uma solução em um computador com sistema operacional Linux Debian com 256 GB de memória RAM. Essas instâncias estão marcadas com um asterisco (*). Para os demais problemas, não foi possível encontrar solução por falta de memória, dada a quantidade de possibilidades de sequências que o algoritmo tem que armazenar durante seu processamento ou a quantidade de estados e transições que precisam ser enumeradas.

Analisando-se os dados tabelados, observa-se que, para a maioria das instâncias em que foi encontrado o supervisor, foi também possível computar uma solução utilizando a heurística. Foram retirados de dois trabalhos da literatura alguns valores de *makespan* para comparação com os resultados encontrados. Nesse aspecto, o MTH teve melhor desempenho que os resultados de Mascis & Pacciarelli (2002) e pior desempenho que os de Pranzo & Pacciarelli (2016). Esses e outros trabalhos apresentam resultados para instâncias maiores, como (10 × 10), que a metodologia proposta não foi suficiente para computar.

Pranzo & Pacciarelli (2016) propõem um algoritmo *Iterated Greedy* (IG) para resolver o problema de escalonamento *job shop* com bloqueio. Essa metaheurística se baseia em fases de destruição e fases de construção, em que a primeira etapa remove parte de uma solução e a segunda constrói uma nova solução, por meio de um algoritmo guloso, a partir da

solução parcial da etapa anterior. Nesse trabalho, foram executadas 10 rodadas do algoritmo proposto, com duração de 60 segundos cada uma. O resultado considerado é a melhor solução encontrada.

Mascis & Pacciarelli (2002) apresentam quatro algoritmos gulosos que se diferenciam pelas regras de despacho às quais estão associados. A heurística consiste em, dado um grafo alternativo (extensão do grafo disjuntivo), gerar uma solução iterativamente de acordo com a regra de despacho definida. Os critérios de parada são encontrar uma solução factível ou uma infactibilidade durante a busca. Os valores presentes na Tabela 4.5 são os melhores resultados dentre os quatro algoritmos propostos.

Embora os resultados encontrados pela aplicação do MTH sobre o supervisor não sejam melhores que os apresentados por Pranzo & Pacciarelli (2016), isso não significa que esse último será sempre preferível para a otimização. Isso porque as técnicas de pesquisa operacional para resolver problemas de escalonamento, muitas vezes, apresentam as soluções em função das chamadas *release dates*, ou seja, o instante de tempo em que cada operação deve ser iniciada. Em situações em que possam ocorrer variações nos tempos previstos para o processo, por exemplo, é possível que as soluções obtidas por meio das técnicas de pesquisa operacional percam sua validade. Nesse mesmo cenário, o controle supervisorio, por sua vez, é capaz de continuar funcionando corretamente.

Observa-se que, para as instâncias cujo supervisor encontrado é da ordem de centenas de milhões de estados e transições, o tempo de processamento da solução é de algumas horas, o que se justifica pela grande quantidade de avaliações que o algoritmo tem que realizar, dado o número de caminhos existentes no supervisor. Já nos problemas menores (até 13 milhões de estados e 40 milhões de transições), ainda que o tamanho dos supervisores seja considerado grande, o tempo de processamento é de alguns segundos apenas.

A maior dificuldade para computar o *makespan* está relacionado com a estrutura interna do MTH que, ao inicializar, carrega todas as transições do supervisor para, então, iniciar o procedimento de escolha do caminho. Como o número de transições é grande, o tempo gasto nessa etapa é, também, grande. Diferentes formas de implementação foram testadas, mas não houve melhora de desempenho nesse aspecto.

O diagrama de Gantt da solução encontrada pela heurística para o problema Hurink_edata_mt06 é apresentado na Figura 4.11. O eixo horizontal marca os instantes de tempo que as operações iniciam e terminam, enquanto o eixo vertical separa as operações pelas máquinas nas quais são processadas. Cada *job* está representado com uma cor diferente. Além do processamento das operações, foi incluído o período em que a máquina permanece com a peça retida, aguardando ser admitida na máquina subsequente, estando, assim, bloqueada (*bloq.*)

O *job* representado pela cor azul, por exemplo, tem a sua primeira operação $O_{1,1}$ processada na máquina M_3 entre os tempos 0 e 1. Após finalizada, é imediatamente admitida na máquina M_1 , sendo o tempo de processamento nela de 3 unidades. Embora a operação tenha sido finalizada, a peça permanece na máquina até que esse *job* possa ser admitido em M_2 , o que somente acontece após a conclusão de $O_{2,1}$ e $O_{6,1}$. Assim, M_1 permanece bloqueada entre os instantes de tempo 4 e 11.

O diagrama de Gantt para a solução da mesma instância, considerando o problema sem bloqueio, também foi gerado para efeito de comparação e está apresentado na Figura 4.12. Embora haja momentos em que as máquinas permanecem ociosas, o valor de *makespan* é

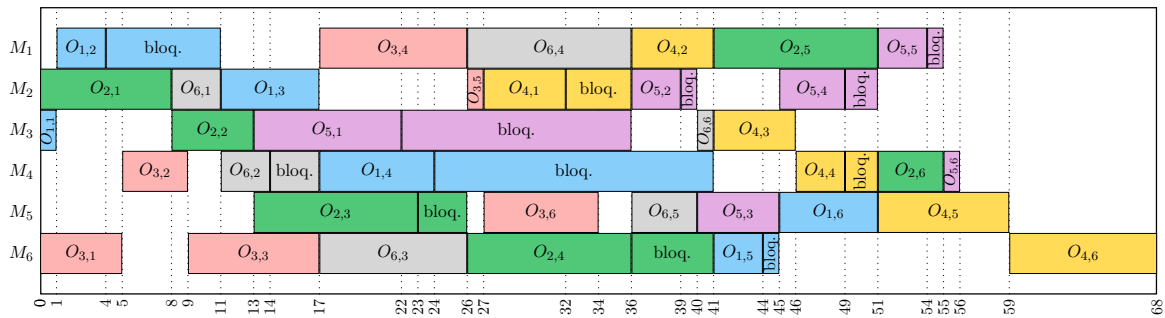


Figura 4.11: Diagrama de Gantt da solução do problema Hurink_edata_mt06 com bloqueio

menor que no caso com bloqueio. Isso era esperado, uma vez que, na ausência do bloqueio, as máquinas estão liberadas imediatamente após finalizarem uma tarefa e, portanto, novos caminhos mais eficientes se tornam factíveis.

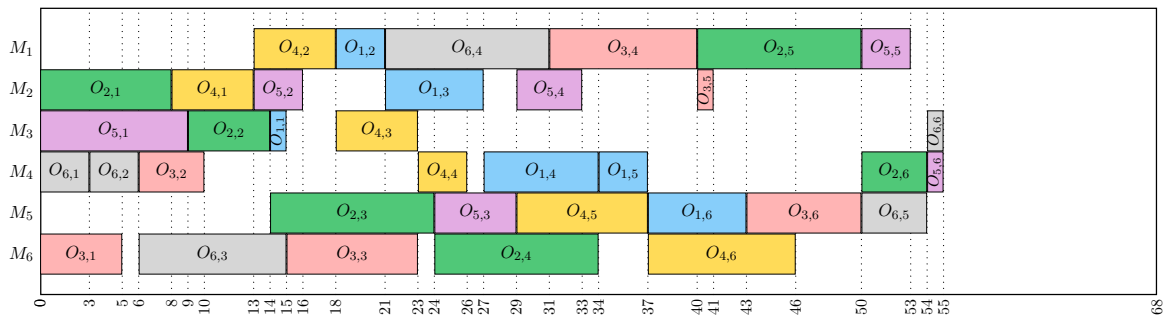


Figura 4.12: Diagrama de Gantt da solução do problema Hurink_edata_mt06 sem bloqueio

Aplicando o MHT sobre o supervisor obtido para o SFM modelado como FJJSP com bloqueio, foi possível encontrar o valor de *makespan* igual a 337. Esse valor é maior que o *makespan* ótimo para a produção da mesma quantidade de produtos, que é igual a 238 (Malik & Pena, 2018). Essa diferença era esperada, uma vez que no primeiro caso, executa-se uma sequência pré-definida, com a base sendo produzida antes do pino, fazendo com que o algoritmo tenha poucas opções de caminhos. Já no segundo caso, há mais liberdade na escolha dos caminhos, de forma que sequências melhores podem ser encontradas.

Capítulo 5

Escalonamento de Tarefas com Demandas Estocásticas

Dentro do contexto da otimização do escalonamento de tarefas, diversos problemas práticos ainda necessitam de atenção. Neste capítulo é apresentado um desses problemas bem como a sua solução proposta.

5.1 Contextualização

Um cenário em particular é considerado: o de um sistema, que compartilha recursos, que tem um fluxo contínuo de produção de um determinado tipo de produto e, em um instante qualquer, recebe uma nova demanda por um outro tipo de produto. Deseja-se encontrar a cadeia de eventos que minimize os tempos de produção total e do novo lote recebido.

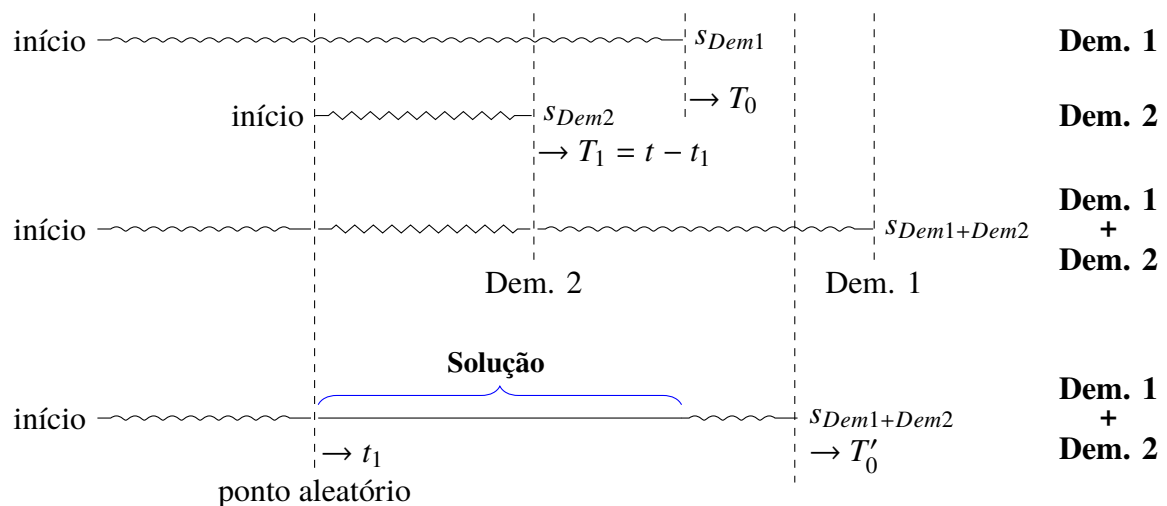


Figura 5.1: Diagrama ilustrativo do cenário abordado

A Figura 5.1 ilustra esse tipo de problema. O desenho da primeira linha, ondulada, representa a cadeia de eventos otimizada s_{Dem1} que minimiza o *makespan* da demanda 1,

referente à produção contínua de um tipo de produto. A segunda linha, em *zigzag*, representa a cadeia de eventos otimizada s_{Dem2} para a produção da demanda 2, que chega em um instante de tempo aleatório, referente à produção de outro tipo de produto.

A terceira linha representa uma das possíveis cadeias de eventos $s_{Dem1+Dem2}$ para a produção dos dois lotes em questão. Para obtê-la, basta simular o cenário em que a produção da primeira demanda é suspensa até que a segunda seja finalizada e, então, retoma-se a produção do ponto em que parou anteriormente. Embora essa seja a forma mais simples para encontrar a nova cadeia, não se aproveitam as possibilidades de executar algumas tarefas em paralelo, o que reduziria o *makespan* dessa cadeia.

Levanta-se o questionamento de qual seria, então, a forma mais rápida de produzir a nova demanda causando o menor atraso para a produção inicial. A quarta linha da Figura 5.1 representa esta situação em que é necessário combinar a produção das duas demandas, considerando que parte da demanda original já foi produzida. Um algoritmo é proposto para realizar o cálculo da nova sequência.

5.1.1 Definição do Problema

Seja um sistema de manufatura capaz de produzir dois tipos de produtos, P1 e P2, com recursos compartilhados. Sejam os tempos de operação das máquinas conhecidos e fixos. Seja s_{Dem1}^* a cadeia otimizada (mínimo *makespan*) de eventos para a produção da demanda 1 (um lote de uma quantidade arbitrária de produtos), definida como $Dem1$, cujo *makespan* (C_{max}) é dado por $C_{max}(s_{Dem1}^*) = T_0$. Dada a chegada da demanda de produção de um novo lote $Dem2$ no instante de tempo t_1 , sendo $0 < t_1 < T_0$, deseja-se encontrar uma nova cadeia de eventos $s_{Dem1+Dem2}$ de forma que o tempo $T_1 = t - t_1$ de produção do novo lote e o tempo T'_0 de produção total (lote original + novo lote) sejam os menores possíveis.

5.2 Solução Proposta

Nesse contexto, buscou-se uma solução baseada em duas ideias centrais. A primeira delas é que produzir dois tipos de produtos ao mesmo tempo é mais rápido que produzir um tipo para, depois, produzir outro tipo. A segunda ideia está relacionada ao aproveitamento das peças que já se encontram no sistema no momento da chegada de uma nova demanda. Ou seja, os produtos inacabados no instante da chegada da nova demanda já fazem parte desse novo lote. Para solucionar o problema apresentado, foi proposto o Algoritmo 3.

O algoritmo inicia executando um algoritmo de otimização a fim de obter uma cadeia s de produção para o lote inicial (linha 1). Nesse caso, o algoritmo de simulação escolhido foi o MTH (Alves et al., 2019), pelos mesmos motivos já apresentados na Seção 4.5.

São passados para a heurística os seguintes parâmetros:

- q_0 : o estado no qual é iniciada a busca, neste caso, o estado inicial do supervisor;
- t_0 : o tempo total, neste caso, zero unidades de tempo;
- a_0 : o agendador de eventos inicial;

Algoritmo 3: Minimização de *makespan* da produção total e da nova demanda

```

1  $s \leftarrow MTH(q_0, t_0, a_0, r_0, (P1_0, P2_0))$ 
2
3 enquanto  $\neg$  nova demanda faça
4    $v \leftarrow \sigma \in s$ 
5    $q \leftarrow \delta(q, \sigma)$ 
6    $t \leftarrow a[\sigma]$ 
7    $a \leftarrow update(a, \sigma)$ 
8   se  $\sigma \in \Sigma_c$  então
9      $r[\sigma] = r[\sigma] - 1$ 
10
11  $r_1 \leftarrow nova\_restrição(P1_1, P2_1)$ 
12  $(x, q, t, a, r) \leftarrow MTH(q, t, a, r_1, (P1_1, P2_1))$ 
13
14  $r_r \leftarrow nova\_restrição(P1_r, P2_r)$ 
15  $(y, t) \leftarrow MTH(q, t, a, r_r, (P1_r, P2_r))$ 
16
17  $u \leftarrow vxy$ 

```

- r_0 : a estrutura de restrição inicial, que contém a quantidade de vezes que cada evento pode ser executado (implicitamente relacionada à profundidade da busca a ser feita);
- $(P1_0, P2_0)$: a demanda inicial, indicando a quantidade de Produtos A ($P1_0$) e de Produtos B ($P2_0$) a serem produzidos.

O MTH devolve a sequência otimizada na variável s . O programa segue executando essa cadeia de eventos e os armazena em v (linha 4) enquanto uma nova demanda não é recebida. A cada evento, as estruturas de estado atual (q), tempo total (t), agendador de eventos (a) e de restrição (r) são atualizadas (linhas 5-9).

Ao se observar a chegada de uma nova demanda, a sequência é interrompida e os valores das demais variáveis são armazenados. A estrutura de restrição r_1 recebe um novo valor em função do tamanho do novo lote ($P1_1, P2_1$), indicando a quantidade de Produtos A e Produtos B (linha 11), considerando as peças ainda presentes no sistema. O valor de $P1_1$ e de $P2_1$ são os valores da nova demanda somados com uma parte da demanda original que ainda não foi produzida, resultando em uma mescla. O valores de q , t e a da etapa anterior são passados, juntamente com a nova restrição r_1 , para o algoritmo MTH que é, então, novamente executado. Esse procedimento representa o cenário em que o sistema pára quando recebe uma nova demanda e retoma com uma nova configuração.

Com a execução do algoritmo de otimização, a cadeia de eventos gerada é armazenada na variável x . Além disso, o novo estado atual do sistema, o tempo decorrido até então e as estruturas do agendador e das restrições (linha 12) são novamente armazenados. Mais uma vez, um novo valor de restrição é atribuído a r_r , considerando a quantidade de eventos necessários para a produção do restante dos produtos do lote original que ainda não foram produzidos até então (linha 14). O Algoritmo MTH é executado pela terceira vez do ponto onde havia parado, porém, considerando a restrição r_r e um lote do tamanho dos produtos

que ainda não haviam sido produzidos. O algoritmo retorna uma cadeia de eventos y e o tempo total de produção (linha 15).

O algoritmo finaliza concatenando as cadeias obtidas anteriormente, resultando na cadeia $u = vxy$ (linha 17), que é a cadeia de eventos que minimiza o tempo de produção total e o tempo de produção do novo lote. Essa cadeia é gerada para fins de análise e comparação apenas, uma vez que, em um sistema real, a cadeia de interesse é somente xy , pois, quando o sistema identifica a chegada de uma nova demanda, os eventos da cadeia v já teriam sido executados. O sistema necessita receber apenas a nova sequência para a produção dos itens restantes.

Em sua versão original, o algoritmo Minimização de Tempo Heurístico é capaz de encontrar soluções otimizadas apenas para um lote inicial de produtos. O Algoritmo 3 estende esse resultado, permitindo que a otimização seja feita com a inclusão de um lote intermediário. Para isso, foram feitas algumas adaptações em relação à estrutura de restrição, que contém a informação do número de vezes que cada evento deve ocorrer para produzir um lote. Foi necessário criar funções para realizar operações de adição e de subtração entre as estruturas a fim de chegar à quantidade de eventos a serem passados para a heurística em cada etapa. Isso se deve, principalmente, ao fato de serem levadas em consideração as peças presentes no sistema no momento da chegada da nova demanda. Assim, o cálculo da nova estrutura de restrição r_1 é feito da seguinte forma:

$$\beta - \gamma = \alpha$$

$$\alpha - \lambda = \theta$$

$$\phi - \theta = r_1$$

onde:

α = quantidade de cada evento feito até o momento

β = quantidade de cada evento da demanda inicial

γ = quantidade de cada evento por fazer no momento

θ = quantidade de cada evento “no sistema” no momento

λ = quantidade de cada evento dos produtos concluídos até o momento

ϕ = quantidade de cada evento da nova demanda

r_1 = quantidade de cada evento considerado para a heurística

Um exemplo do passo a passo para calcular a estrutura de restrição r_1 é apresentado na Tabela 5.2. Considerando-se a demanda inicial em que sejam necessários, para sua produção, uma quantidade de 10 eventos a_1 , 10 eventos a_2 e 5 eventos a_3 (β), representado pelo vetor (10,10,5) e que em um instante de tempo aleatório a quantidade de eventos por fazer é $\gamma = (4,5,3)$, tem-se que $\beta - \gamma = \alpha = (6,5,2)$, que equivale à quantidade de eventos feitos até o momento. Subtraindo de α a quantidade de eventos dos produtos já concluídos $\lambda = (4,4,2)$, obtém-se o número de eventos que iniciaram a produção de produtos que ainda não foram finalizados $\theta = (2,1,0)$. Estes eventos que ainda estão “no sistema” são aproveitados para a nova demanda e, portanto, são subtraídos do número de eventos necessários para esse novo lote $\phi = (5,5,10)$, resultando em $r_1 = (3,4,10)$. Isso significa que, dos 5 eventos a_1 , 5 eventos a_2 e 10 eventos a_3 da nova demanda, é necessário executar apenas 3 eventos a_1 , 4 eventos a_2

e 10 eventos a_3 para concluir essa etapa.

Tabela 5.1: Cálculo de r_1

$\beta - \gamma = \alpha$				$\alpha - \lambda = \theta$				$\phi - \theta = r_1$			
	β	γ	α		α	λ	θ		ϕ	θ	r_1
a_1	10	4	6	a_1	6	4	2	a_1	5	2	3
a_2	10	5	5	a_2	5	4	1	a_2	5	1	4
a_3	5	3	2	a_3	2	2	0	a_3	10	0	10

O cálculo de r_r , por sua vez, é feito da seguinte forma:

$$\gamma + \phi - r_1 = r_r$$

Utilizando-se os mesmos valores do exemplo dado, os eventos que devem ser executados na última rodada da heurística são obtidos somando-se a quantidade de eventos por fazer no instante em que chega a nova demanda $\gamma = (4,5,3)$ com o número de eventos para produzir a nova demanda $\phi = (5,5,10)$. Desse valor, subtrai-se a quantidade de eventos executados na etapa anterior $r_1 = (3,4,10)$ e obtém-se a quantidade de eventos que ainda precisam ser executados $r_r = (6,6,3)$.

Tabela 5.2: Cálculo de r_r

$\gamma + \phi - r_1 = r_r$				
	γ	ϕ	r_1	r_r
a_1	4	5	3	6
a_2	5	5	4	6
a_3	3	10	10	3

5.3 Estudo de Caso

Para testar o algoritmo proposto, foi escolhido o Sistema Flexível de Manufatura, apresentado na Seção 3.5.1. Nesse caso, a modelagem da planta e das especificações foi feita utilizando a TCS na sua forma tradicional.

5.3.1 Simulação

Para a simulação do Algoritmo 3, foi definida uma demanda inicial de produção de 100 Produtos A e 0 Produtos B, para representar uma produção contínua de um só produto. No instante de tempo 3.000 segundos, é acrescentada uma nova demanda de 0 Produtos A e 40 Produtos B. Uma vez que a restrição r_1 é definida em função do tamanho do lote da nova demanda, podendo ser mesclada com os produtos ainda não finalizados da demanda inicial, há mais de uma solução possível.

No caso simulado, verificou-se que, após 3.000 segundos, havia 36 Produtos A prontos e, naturalmente, 64 ainda por ser feitos. Dessa forma, é possível simular o algoritmo

com diversas mesclagens que podem ser a combinação de qualquer quantidade entre 0 e 64 Produtos A com os 40 Produtos B. A combinação (0,40) indica que não há mescla e representa o caso da linha 3 da Figura 5.1 em que a produção contínua é interrompida para a produção da nova demanda. O outro extremo é a mescla (64,40), ou seja, mesclar todos os produtos restantes. Quaisquer outras combinações representam um cenário entre os dois exemplos dados.

A simulação de todas as combinações pôde ser realizada considerando o instante 3.000 segundos e o instante em que o último produto da nova demanda é finalizado (*makespan* da nova demanda) e o tempo entre o início e o fim da produção da demanda inicial (*makespan* total). Os resultados encontrados estão apresentados na Figura 5.2. Nela estão indicadas as quantidades mescladas para alguns dos resultados encontrados.

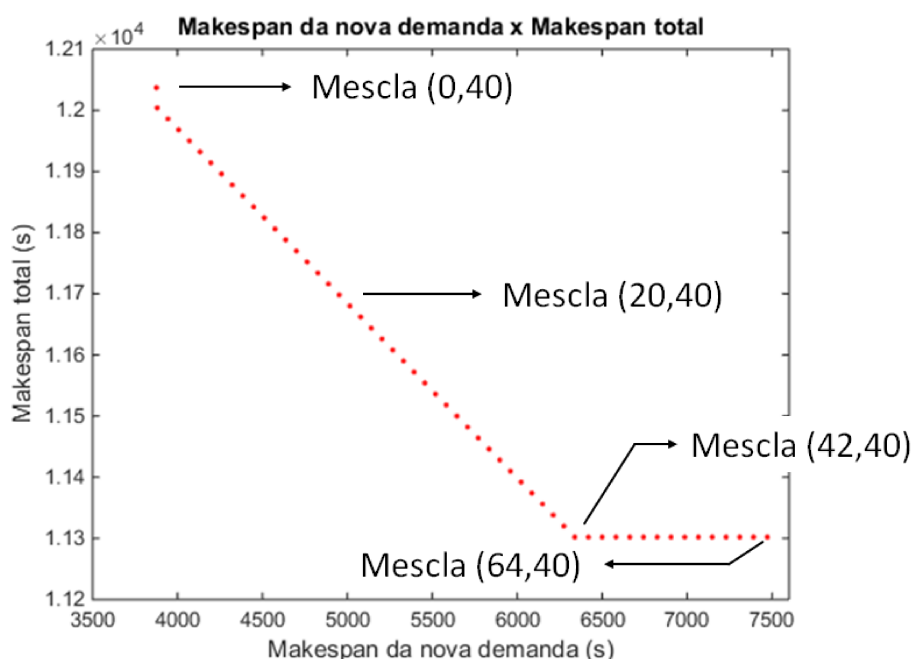


Figura 5.2: Valores de *makespan* das soluções encontradas

É possível observar que a mescla de uma quantidade pequena de Produtos A com a nova demanda de Produtos B permite a finalização mais rápida do novo lote, mas provoca o aumento do *makespan* total, uma vez que nesse cenário há a execução de poucas tarefas em paralelo. À medida em que a mesclagem dos produtos aumenta, o tempo de produção da nova demanda também aumenta, pois a planta compartilha recursos, mas o *makespan* total é reduzido, pois a produção ocorre simultaneamente.

A partir do momento em que a quantidade de produtos mesclados é aproximadamente igual, como no ponto (42,40), observa-se que não há variação no *makespan* total, embora o *makespan* da nova demanda cresça. Isso acontece pelo fato de que, a partir do ponto mencionado, aumentar a mesclagem possibilita a produção de mais Produtos A entre os Produtos B, atrasando o término da produção da nova demanda. A grande vantagem desse cenário está relacionada às diferentes possibilidades de soluções encontradas.

Visto que a priorização da nova demanda implica no aumento do tempo de produção total, fica clara, então, a necessidade de se fazer um *trade off* entre as duas medidas, com base nos

critérios e restrições que cada problema pode apresentar, como *deadlines*, penalidades por atraso e outros. Além disso, esse cenário possibilita que seja adaptado ao problema proposto o ferramental utilizado na solução de problemas multiobjetivos, como os métodos de auxílio à tomada de decisão.

Também foi registrado o tempo de execução do algoritmo para cada uma das soluções, do instante em que chega a nova demanda até o fim da execução do algoritmo. Estes valores são sempre menores que 8,5 segundos, conforme a Figura 5.3. Novamente, os valores de mescla de algumas soluções estão indicadas na figura.

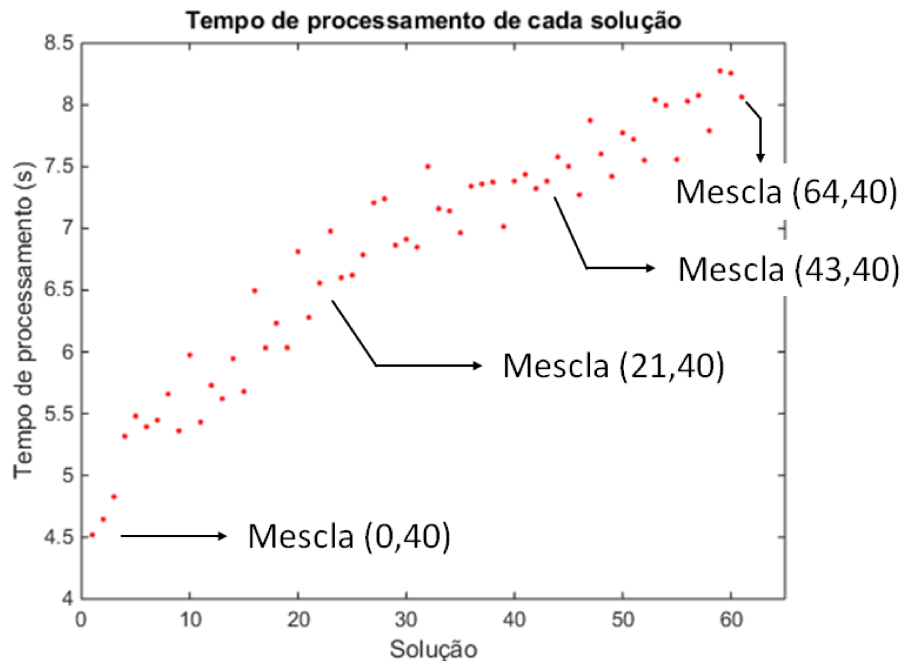


Figura 5.3: Tempo de execução do algoritmo para cada solução encontrada.

A partir do gráfico é possível observar que o tempo de execução do algoritmo cresce, aproximadamente, de forma linear em função da quantidade de produtos mesclados. A explicação para isso é o fato de o *branch factor* crescer à medida que a quantidade de produtos também cresce. Quando a quantidade de Produtos A mesclados é pequena, o algoritmo tem que sequenciar, praticamente, apenas Produtos B em uma etapa e apenas Produtos A em outra etapa. Quando a mescla é grande, diversas são as possibilidades para a produção dos dois produtos, aumentando o custo computacional.

O tempo total para encontrar todas as soluções foi de seis minutos e 55 segundos. Comparando-se o maior tempo para encontrar uma solução, 8 segundos, com o tempo de produção da nova demanda obtém-se uma relação de $\approx 0,13\%$ para o caso simulado. Pode-se, então, considerar o tempo de execução baixo, o que torna essa abordagem viável e indicada para problemas dessa natureza.

Para trabalhos futuros, outros contextos com mais condições podem ser simulados, como a existência de um *deadline* para o término da produção da nova demanda ou a chegada de mais de uma demanda ao longo da produção contínua. Uma melhoria interessante é encontrar formas eficazes de definir o valor das mesclas a serem simuladas, de forma a não ser necessário computar todas as soluções possíveis para tomar uma decisão.

Capítulo 6

Conclusão

Neste trabalho foi desenvolvido um algoritmo para converter automaticamente um problema de escalonamento *job shop* flexível com bloqueio em autômatos usando a Teoria de Controle Supervisório. Foram apresentados o mecanismo para a interpretação dos dados de uma instância em seu formato textual e a metodologia para a geração dos autômatos que representam suas plantas e suas especificações, tanto com bloqueio quanto sem.

Como um trabalho exploratório, esta pesquisa mostrou que é possível usar linguagens e autômatos e a Teoria do Controle Supervisório para representar problemas bastante discutidos em outras áreas do conhecimento, como os problemas de escalonamento *job shop*. Com isso, além de expandir os campos de atuação da TCS, dentro da comunidade de SED, abrem-se novas possibilidades de desenvolvimentos de estudos. Uma delas é a busca pelo aprimoramento dos resultados obtidos, de forma a aproximá-los daqueles encontrados no estado da arte dos problemas de escalonamento *job shop*. Outra é o desenvolvimento ainda maior da teoria de Sistemas a Eventos Discretos, que é, muitas vezes, impulsionado por novos problemas, tal como o que foi objeto deste trabalho.

Os supervisores, de algumas instâncias, que modelam todo o comportamento em malha fechada do sistema (que contém todo o universo de busca) foram encontrados. Isso faz com que eles possam ser utilizados por qualquer heurística, sem que seja necessário realizar nenhuma reavaliação em relação à factibilidade lógica da solução. Uma heurística foi aplicada sobre o supervisor a fim de exemplificar o uso do universo de busca. Foi possível encontrar sequências otimizadas para alguns problemas e os valores de *makespan* encontrados se mostraram próximos dos resultados da literatura.

Outro resultado deste trabalho foi desenvolvimento de um algoritmo para a obtenção de cadeias de eventos que minimizam os tempos de produção em um sistema flexível de manufatura. Este trabalho estendeu resultados anteriores em que se encontram soluções para um único lote inicial, aumentando sua aplicabilidade. A simulação de um cenário ilustrativo desse problema mostrou que a produção mais rápida de uma nova demanda compromete o tempo para a produção da demanda total.

A principal limitação encontrada neste trabalho diz respeito ao uso de memória para sintetizar o supervisor. Tal limitação está intimamente ligada ao problema da explosão do espaço de estados, que impossibilitou a computação de soluções para problemas de dimensões maiores, como (10×10) ou (15×5) . Dessa forma, faz-se necessário, em trabalhos futuros, encontrar 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.

O novo campo aberto por este trabalho possibilita que pesquisas futuras sejam realizadas a partir do estudo de outras heurísticas que possam fornecer melhores resultados de *makespan*, considerando o universo de busca apresentado. Nesse sentido, seria possível combinar os benefícios da TCS com os algoritmos típicos para minimização de *makespan* de FJSSP, consolidados após décadas de estudos e experimentos.

Referências Bibliográficas

- Abdeddaim, Y. & Maler, O. (2001). Job-shop scheduling using timed automata? In *International Conference on Computer Aided Verification* (pp. 478–492).: Springer.
- Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3), 391–401.
- AitZai, A., Benmedjdoub, B., & Boudhar, M. (2012). A branch and bound and parallel genetic algorithm for the job shop scheduling problem with blocking. *International Journal of Operational Research*, 14(3), 343–365.
- Alves, L., Pena, P., & Takahashi, R. (2019). Planejamento da produção em sistemas a eventos discretos usando heurística. *XII Simpósio Brasileiro de Automação Inteligente (SBAI)*.
- Alves, L. V., Martins, L. R., & Pena, P. N. (2017). Ultradesc-a library for modeling, analysis and control of discrete event systems. *IFAC-PapersOnLine*, 50(1), 5831–5836.
- Alves, L. V., Pena, P. N., & Takahashi, R. H. (2016). Planejamento da produção baseado no critério do máximo paralelismo com restrições temporais. In *Anais do Congresso Brasileiro de Automática*.
- Applegate, D. & Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, 3(2), 149–156.
- Baker, K. R. (1974). *Introduction to sequencing and scheduling*. John Wiley & Sons.
- Baker, K. R. & Trietsch, D. (2013). *Principles of sequencing and scheduling*. John Wiley & Sons.
- Behnke, D. & Geiger, M. J. (2012). Test instances for the flexible job shop scheduling problem with work centers. *Logistics Management Department, Hamburg, Germany*.
- Błażewicz, J., Pesch, E., & Sterna, M. (2000). The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research*, 127(2), 317–331.
- 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., & 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.
- Cassandras, C. G. & Lafortune, S. (2009). *Introduction to discrete event systems*. Editora Springer, 2^a edition.
- Chambers, J. B. & 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, ORP96-10*.
- Coffman, E. G. & Bruno, J. L. (1976). *Computer and job-shop scheduling theory*. John Wiley & Sons.
- Costa, T. A., Pena, P. N., & Takahashi, R. H. (2018). Sco-concat: a solution to a planning problem in flexible manufacturing systems using supervisory control theory and optimization techniques. *Journal of Control, Automation and Electrical Systems*, (pp. 1–12).
- Dauzère-Pérès, S. & 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.
- Dell’Amico, M. & Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations research*, 41(3), 231–252.
- Eccles, H. E. (1954). Logistics-what is it? *Naval Research Logistics Quarterly*, 1(1), 5–15.
- French, S. (1982). Sequencing and scheduling. *An Introduction to the Mathematics of the Job-shop*.
- Garey, M. R. & Johnson, D. S. (1979). *Computers and intractability: A Guide to the Theory of NP-Completeness*, volume 29. W.H. Freeman New York.
- Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5 (pp. 287–326). Elsevier.
- Hill, R. C. & Lafortune, S. (2016). Planning under abstraction within a supervisory control context. In *2016 IEEE 55th Conference on Decision and Control (CDC)* (pp. 4770–4777).: IEEE.
- Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2001). Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1), 60–65.
- Hu, S. J. (2013). Evolving paradigms of manufacturing: from mass production to mass customization and personalization. *Procedia Cirp*, 7, 3–8.
- Hurink, J., Jurisch, B., & Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum*, 15(4), 205–215.

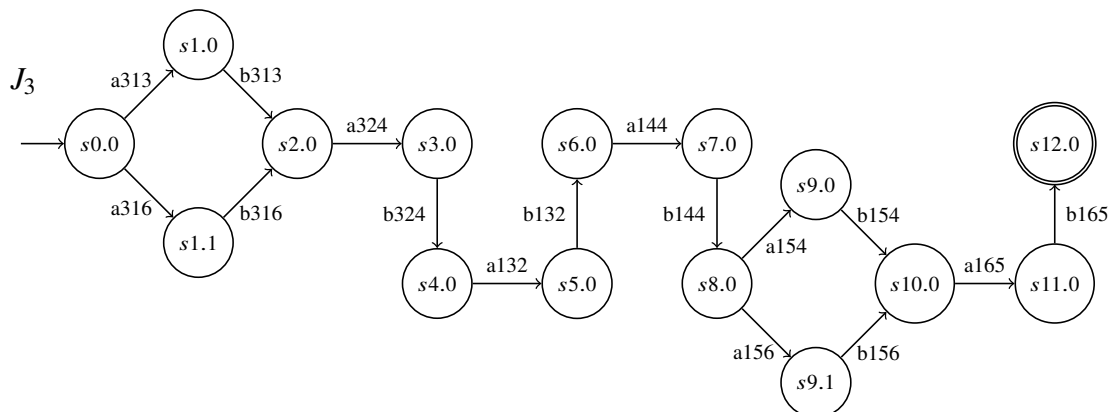
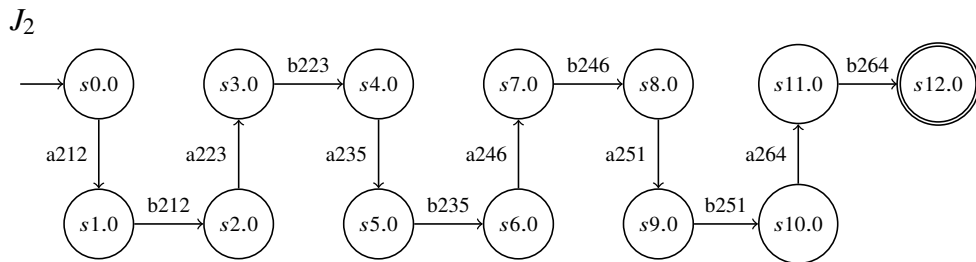
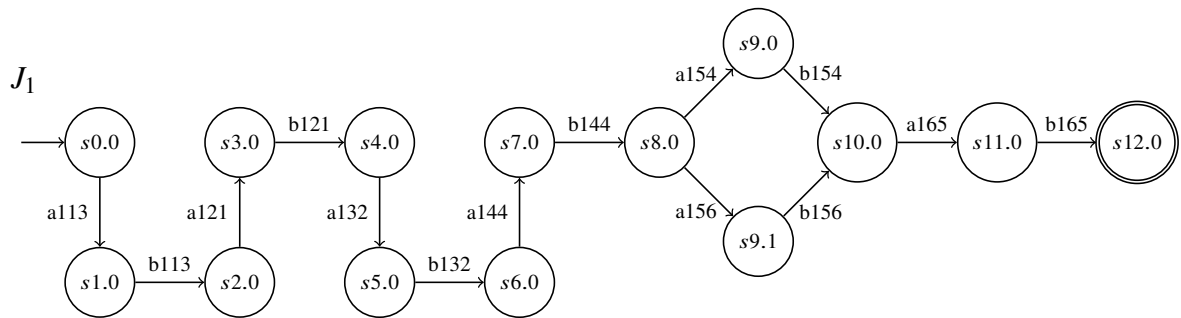
- Jezequel, L. & Fabre, E. (2012). Turbo planning. *IFAC Proceedings Volumes*, 45(29), 301–306.
- Jo, H.-W., Ahn, J.-H., Park, J.-S., Oh, J.-H., & Lim, J.-T. (2010). Task planning for service robots with optimal supervisory control. In *2010 IEEE Conference on Robotics, Automation and Mechatronics* (pp. 303–308): IEEE.
- Johnson, S. M. (1954). Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1), 61–68.
- Lawrence, S. (1984). An experimental investigation of heuristic scheduling techniques. *Supplement to resource constrained project scheduling*.
- Levy, F. K., Thompson, G. L., & Wiest, J. D. (1962). Multiship, multishop, workload-smoothing progfum. *Naval Research Logistics Quarterly*, 9(1), 37–44.
- Louaquad, S. & Kamach, O. (2015). Scheduling of blocking and no wait job shop problems in robotic cells.
- Lustosa, Junqueira, L., de Mesquita, M. A., & Oliveira, R. J. (2008). *Planejamento e controle da produção*. Coleção Campus-ABEPRO. Editora Elsevier Brasil, 4ª edition.
- Malik, R. & Pena, P. N. (2018). Optimal task scheduling in a flexible manufacturing system using model checking. *IFAC-PapersOnLine*, 51(7), 230–235.
- Mascis, A. & 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., & Winters, P. R. (1963). *Industrial scheduling*. Prentice-Hall.
- Petrônio, G. M. & Laugeni, P. (2005). *Administração da produção*. Editora Saraiva, 2ª edition.
- Pezzella, F., Morganti, G., & Ciaschetti, G. (2008). A genetic algorithm for the flexible jobshop scheduling problem. *Computers and Operations Research*, 35(10), 3202–3212.
- Pinedo, M. L. (2016). *Scheduling: theory, algorithms, and systems*. New York, New York: Springer, 5ª edition.
- Pinha, D. C., de Queiroz, M. H., & Cury, J. E. (2011). Optimal scheduling of a repair shipyard based on supervisory control theory. In *2011 IEEE International Conference on Automation Science and Engineering* (pp. 39–44): IEEE.
- Pranzo, M. & Pacciarelli, D. (2016). An iterated greedy metaheuristic for the blocking job shop scheduling problem. *Journal of Heuristics*, 22(4), 587–611.
- Queiroz, M. H. d. et al. (2004). Controle supervisório modular e multitarefa de sistemas compostos.
- Ramadge, P. J. & Wonham, W. M. (1989). The control of discrete event systems. *Proceedings of IEEE*, 77, 81–98.

- Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2), 59–66.
- Su, R., Van Schuppen, J. H., & Rooda, J. E. (2011). The synthesis of time optimal supervisors by using heaps-of-pieces. *IEEE Transactions on Automatic Control*, 57(1), 105–118.
- Van Laarhoven, P. J., Aarts, E. H., & Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations research*, 40(1), 113–125.
- Weaver, P. (2006). A brief history of scheduling: Back to the future. In *Proceedings of the 2006 myPrimavera06 Conference* (pp. 4–6).
- Yamada, T. & Nakano, R. (1997). Job shop scheduling. *IEE control Engineering series*, (pp. 134–134).

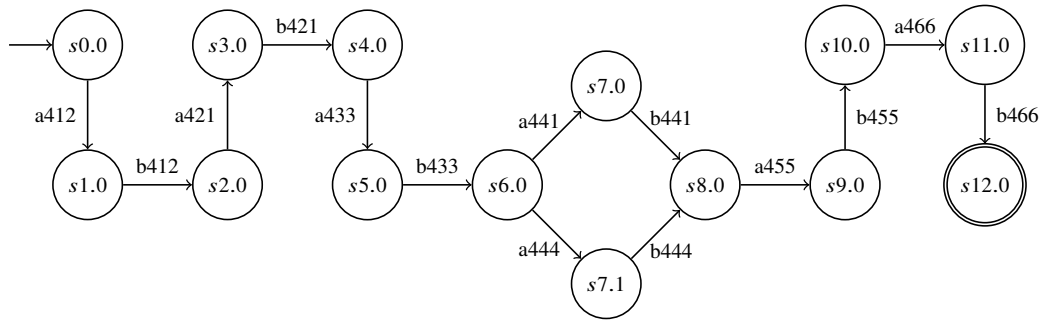
Apêndices

Apêndice A

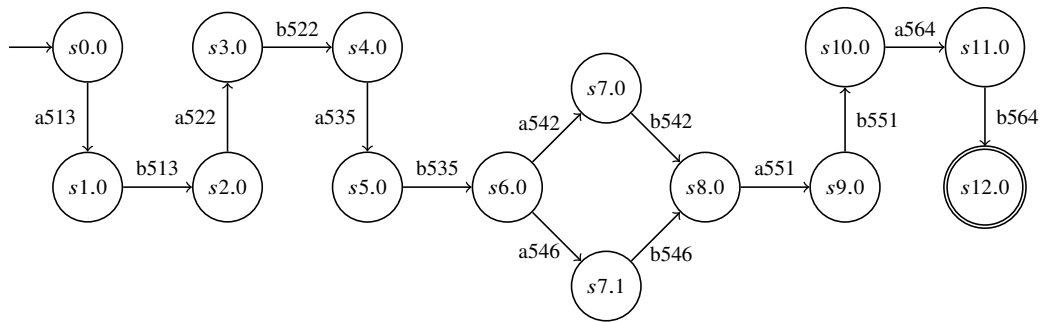
Modelagem do Problema Hurink_edata_mt06 em Autômatos



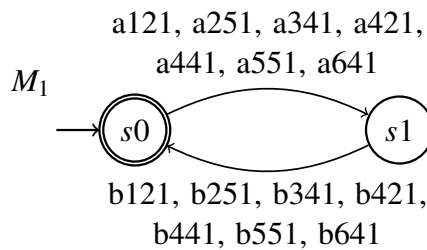
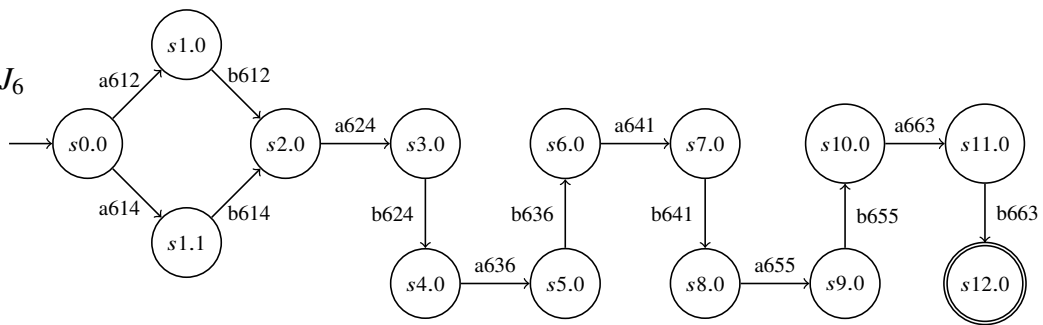
J_4

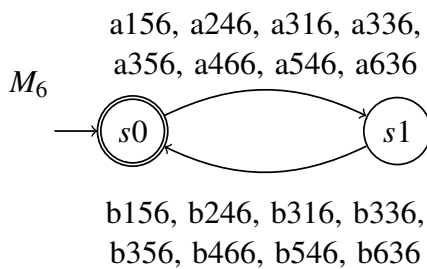
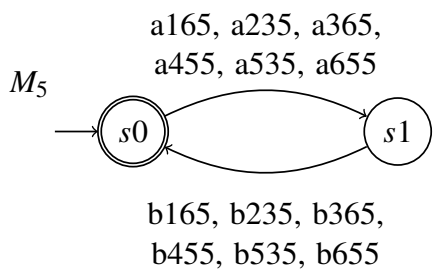
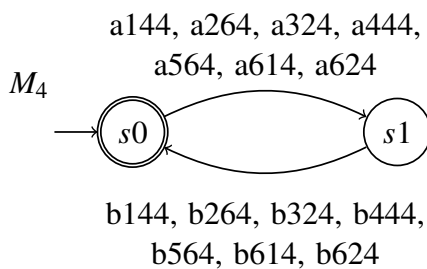
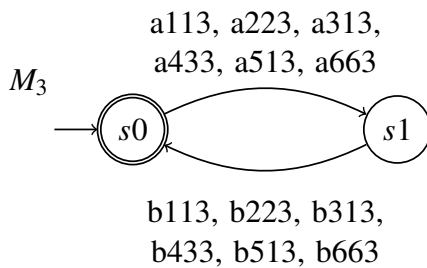
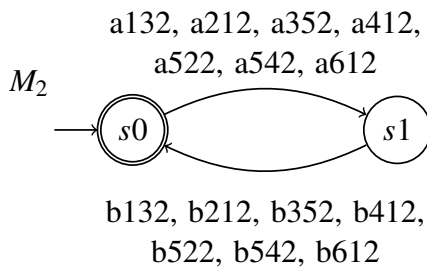


J_5



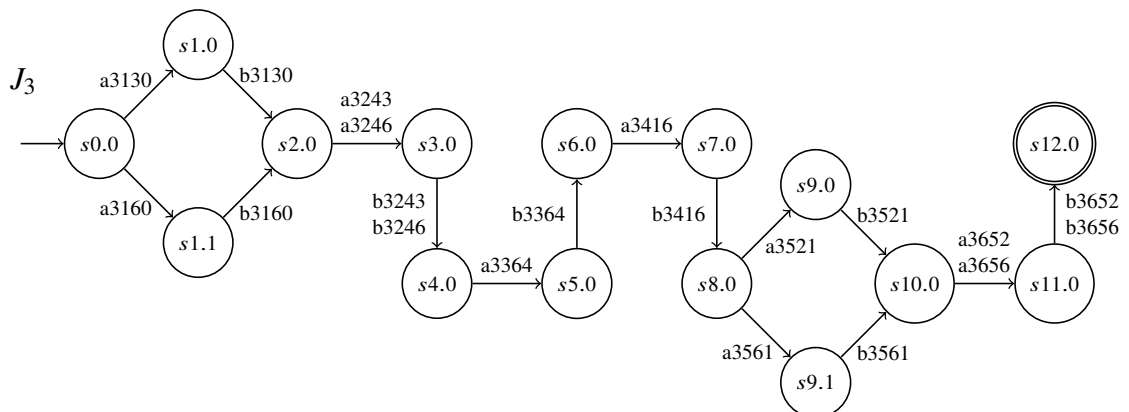
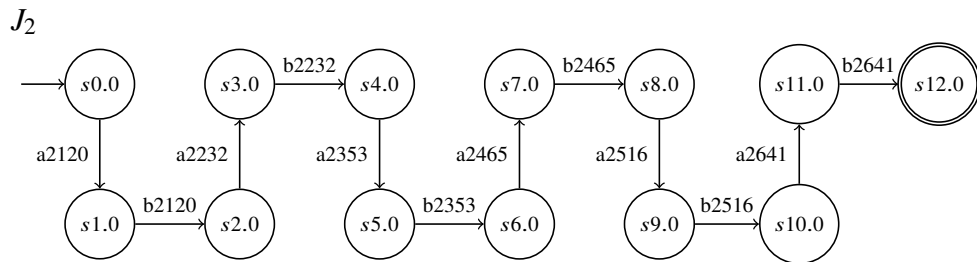
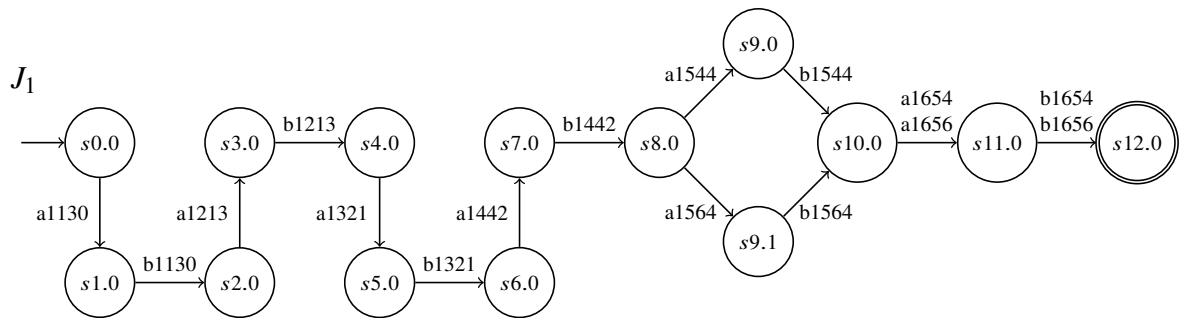
J_6



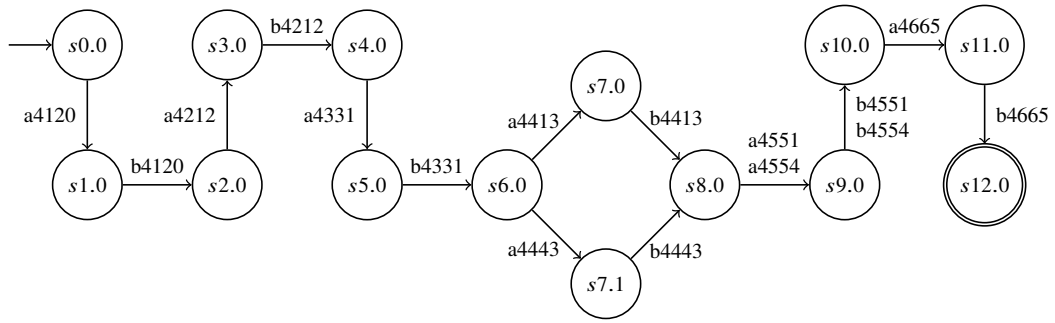


Apêndice B

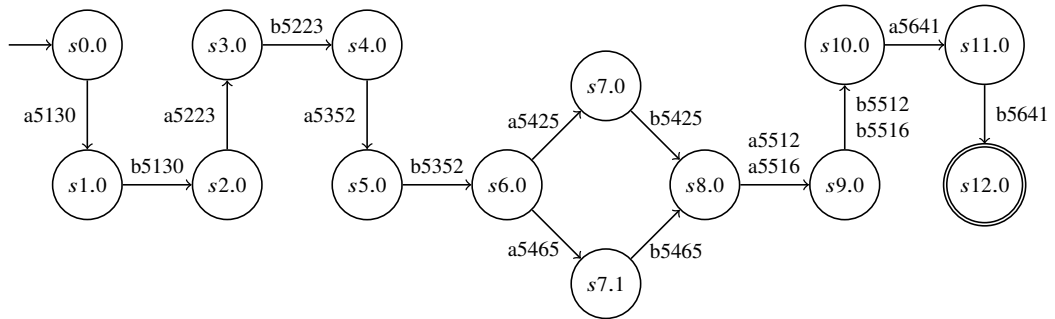
Modelagem do Problema Hurink_edata_mt06 com Bloqueio em Autômatos



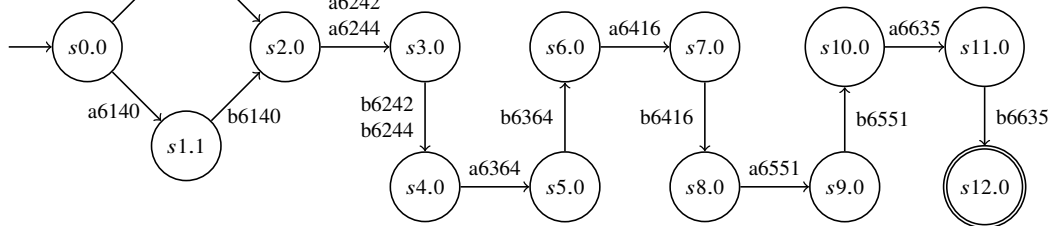
J_4



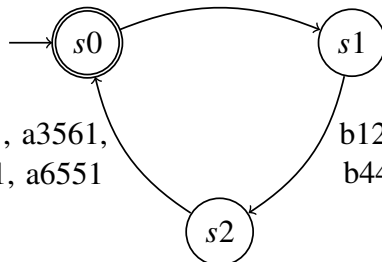
J_5



J_6

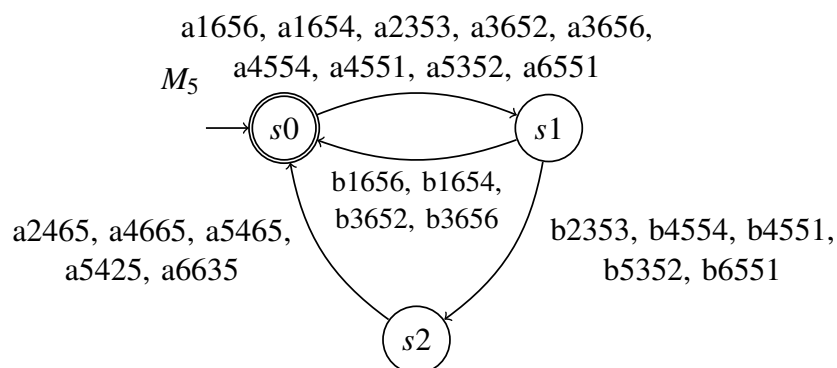
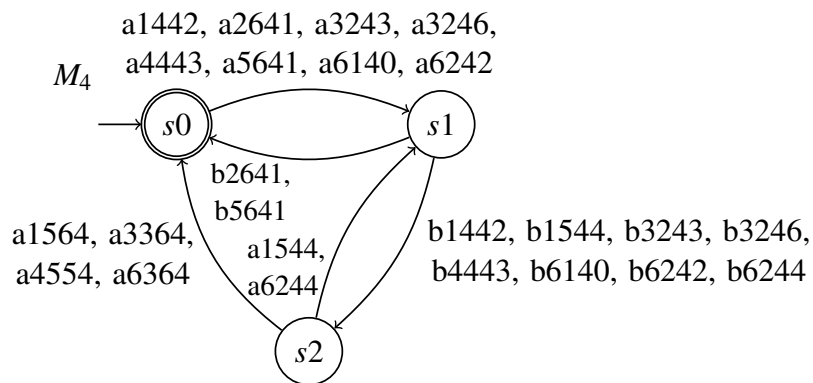
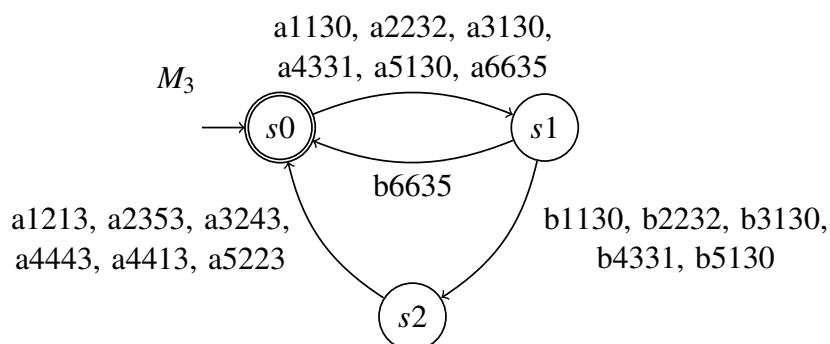
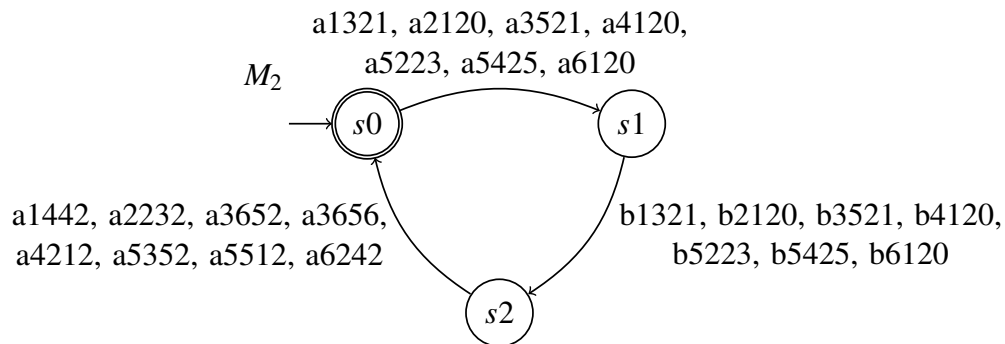


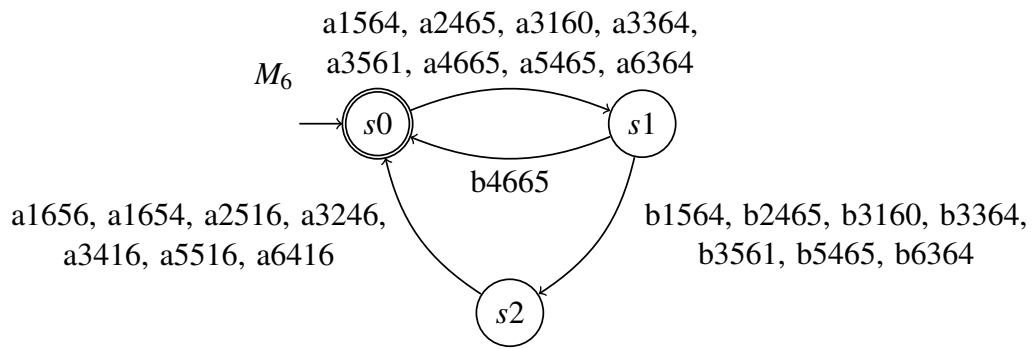
M_1 a1213, a2516, a3416, a4212,
a4413, a5516, a5512, a6416



a1321, a2641, a3521, a3561,
a4331, a4551, a5641, a6551

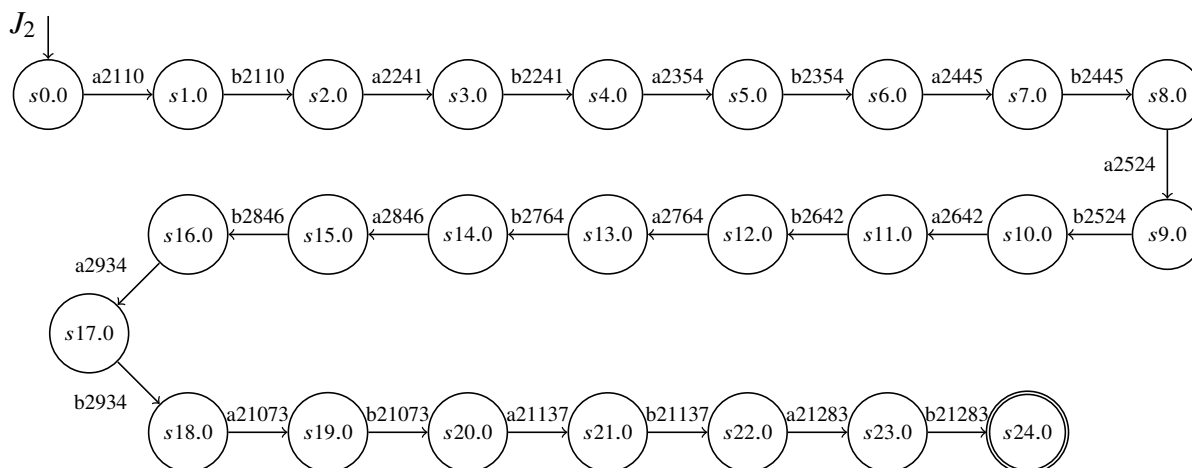
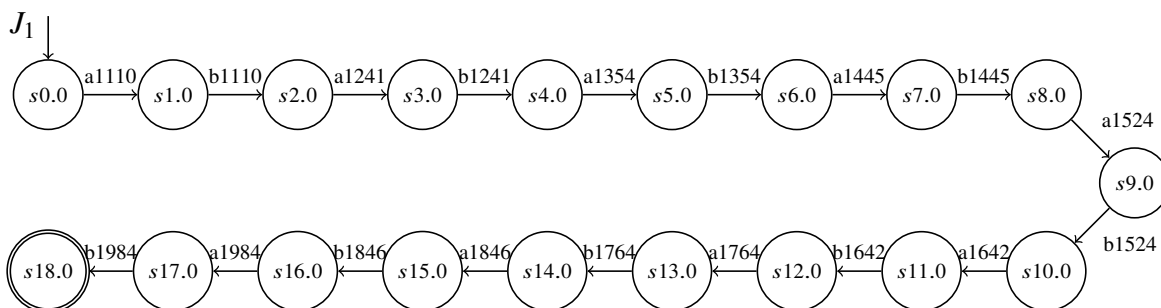
b1213, b2516, b3416, b4212,
b4413, b5516, b5512, b6416

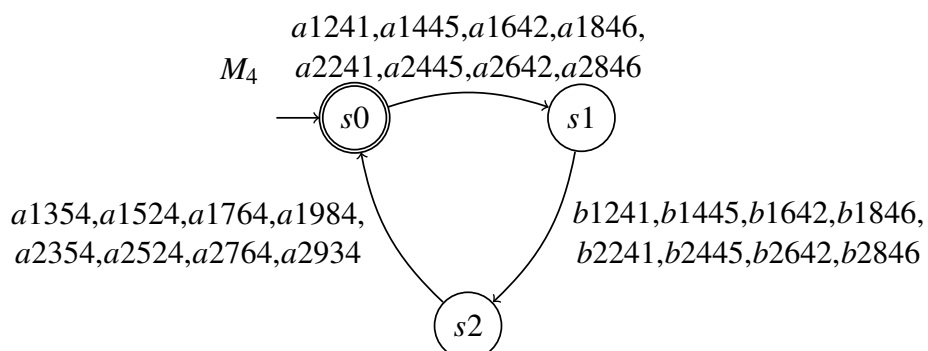
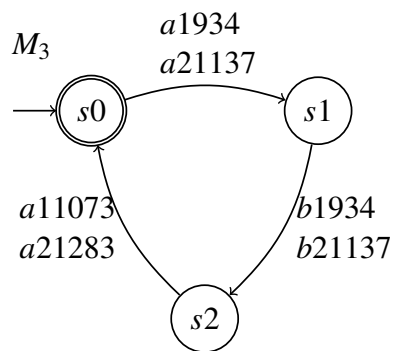
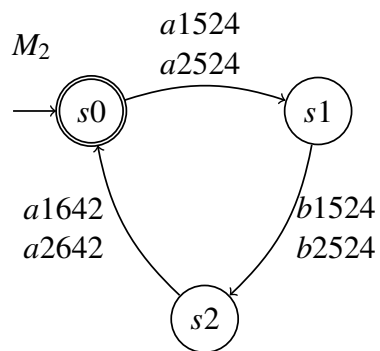
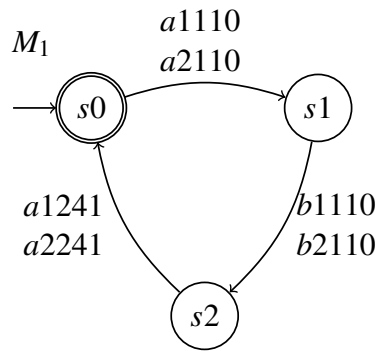


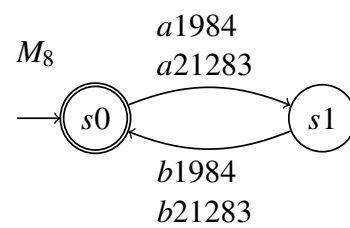
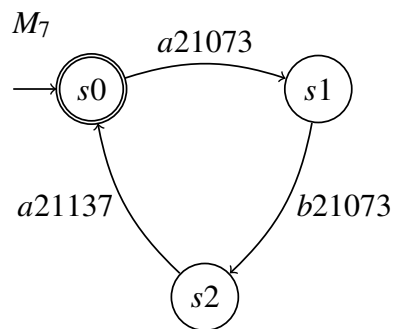
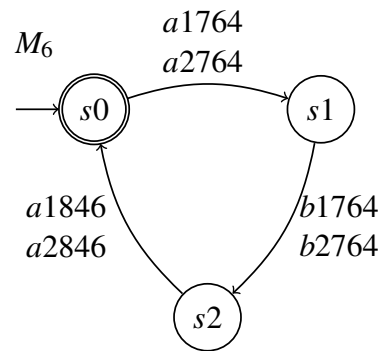
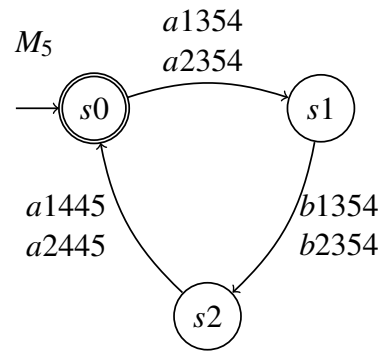


Apêndice C

Modelagem do Sistema Flexível de Manufatura







Anexos

Anexo A

Limites Superior e Inferior para Instâncias de FJSSP

Hurink-sdata							
Instance	n x m	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date
sdata-abz5	10 x 10	1234	[CPO]	nov 2013	1234	[Q]	mar 2013
sdata-abz6	10 x 10	943	[CPO]	nov 2013	943	[Q]	mar 2013
sdata-abz7	20 x 15	656	[CPO]	nov 2013	656	[CPO]	nov 2013
sdata-abz8	20 x 15	653	[CPO]	Dec 2014	667	[CPO]	nov 2013
sdata-abz9	20 x 15	678	[CPO]	nov 2013	678	[CPO]	nov 2013
sdata-car1	11 x 5	7038	[Q]	mar 2013	7038	[Q]	mar 2013
sdata-car2	13 x 4	7166	[Q]	mar 2013	7166	[Q]	mar 2013
sdata-car3	12 x 5	7312	[Q]	mar 2013	7312	[Q]	mar 2013
sdata-car4	14 x 4	8003	[Q]	mar 2013	8003	[Q]	mar 2013
sdata-car5	10 x 6	7702	[CPO]	nov 2013	7702	[Q]	mar 2013
sdata-car6	8 x 9	8313	[CPO]	nov 2013	8313	[Q]	mar 2013
sdata-car7	7 x 7	6558	[CPO]	nov 2013	6558	[Q]	mar 2013
sdata-car8	8 x 8	8264	[CPO]	nov 2013	8264	[Q]	mar 2013
sdata-la01	10 x 5	666	[2]	Feb 2000	666	[2]	Feb 2000
sdata-la02	10 x 5	655	[2]	Feb 2000	655	[2]	Feb 2000
sdata-la03	10 x 5	597	[2]	Feb 2000	597	[2]	Feb 2000
sdata-la04	10 x 5	590	[2]	Feb 2000	590	[2]	Feb 2000
sdata-la05	10 x 5	593	[2]	Feb 2000	593	[2]	Feb 2000
sdata-la06	15 x 5	926	[2]	Feb 2000	926	[2]	Feb 2000
sdata-la07	15 x 5	890	[2]	Feb 2000	890	[2]	Feb 2000
sdata-la08	15 x 5	863	[2]	Feb 2000	863	[2]	Feb 2000
sdata-la09	15 x 5	951	[2]	Feb 2000	951	[2]	Feb 2000
sdata-la10	15 x 5	958	[2]	Feb 2000	958	[2]	Feb 2000
sdata-la11	20 x 5	1222	[2]	Feb 2000	1222	[2]	Feb 2000
sdata-la12	20 x 5	1039	[2]	Feb 2000	1039	[2]	Feb 2000
sdata-la13	20 x 5	1150	[2]	Feb 2000	1150	[2]	Feb 2000
sdata-la14	20 x 5	1292	[2]	Feb 2000	1292	[2]	Feb 2000

Table continued from previous page

Instance	n x m	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date
sdata-la15	20 x 5	1207	[2]	Feb 2000	1207	[2]	Feb 2000
sdata-la16	10 x 10	945	[2]	Feb 2000	945	[2]	Feb 2000
sdata-la17	10 x 10	784	[2]	Feb 2000	784	[2]	Feb 2000
sdata-la18	10 x 10	848	[2]	Feb 2000	848	[2]	Feb 2000
sdata-la19	10 x 10	842	[2]	Feb 2000	842	[2]	Feb 2000
sdata-la20	10 x 10	902	[2]	Feb 2000	902	[2]	Feb 2000
sdata-la21	15 x 10	1046	[2]	Feb 2000	1046	[2]	Feb 2000
sdata-la22	15 x 10	927	[2]	Feb 2000	927	[2]	Feb 2000
sdata-la23	15 x 10	1032	[2]	Feb 2000	1032	[2]	Feb 2000
sdata-la24	15 x 10	935	[2]	Feb 2000	935	[2]	Feb 2000
sdata-la25	15 x 10	977	[2]	Feb 2000	977	[2]	Feb 2000
sdata-la26	20 x 10	1218	[2]	Feb 2000	1218	[2]	Feb 2000
sdata-la27	20 x 10	1235	[2]	Feb 2000	1235	[2]	Feb 2000
sdata-la28	20 x 10	1216	[2]	Feb 2000	1216	[2]	Feb 2000
sdata-la29	20 x 10	1152	[CPO]	nov 2013	1152	[CPO]	nov 2013
sdata-la30	20 x 10	1355	[2]	Feb 2000	1355	[2]	Feb 2000
sdata-la31	30 x 10	1784	[2]	Feb 2000	1784	[2]	Feb 2000
sdata-la32	30 x 10	1850	[2]	Feb 2000	1850	[2]	Feb 2000
sdata-la33	30 x 10	1719	[2]	Feb 2000	1719	[2]	Feb 2000
sdata-la34	30 x 10	1721	[2]	Feb 2000	1721	[2]	Feb 2000
sdata-la35	30 x 10	1888	[2]	Feb 2000	1888	[2]	Feb 2000
sdata-la36	15 x 15	1268	[2]	Feb 2000	1268	[CPO]	nov 2013
sdata-la37	15 x 15	1397	[2]	Feb 2000	1397	[2]	Feb 2000
sdata-la38	15 x 15	1196	[CPO]	nov 2013	1196	[2]	Feb 2000
sdata-la39	15 x 15	1233	[2]	Feb 2000	1233	[2]	Feb 2000
sdata-la40	15 x 15	1222	[2]	Feb 2000	1222	[CPO]	nov 2013
sdata-mt06	6 x 6	55	[2]	Feb 2000	55	[2]	Feb 2000
sdata-mt10	10 x 10	930	[2]	Feb 2000	930	[2]	Feb 2000
sdata-mt20	20 x 5	1165	[2]	Feb 2000	1165	[2]	Feb 2000
sdata-orb1	10 x 10	1059	[CPO]	nov 2013	1059	[CPO]	nov 2013
sdata-orb10	10 x 10	944	[CPO]	nov 2013	944	[Q]	mar 2013
sdata-orb2	10 x 10	888	[CPO]	nov 2013	888	[Q]	mar 2013
sdata-orb3	10 x 10	1005	[CPO]	nov 2013	1005	[Q]	mar 2013
sdata-orb4	10 x 10	1005	[CPO]	nov 2013	1005	[Q]	mar 2013
sdata-orb5	10 x 10	887	[CPO]	nov 2013	887	[Q]	mar 2013
sdata-orb6	10 x 10	1010	[CPO]	nov 2013	1010	[Q]	mar 2013
sdata-orb7	10 x 10	397	[CPO]	nov 2013	397	[CPO]	nov 2013
sdata-orb8	10 x 10	899	[CPO]	nov 2013	899	[Q]	mar 2013
sdata-orb9	10 x 10	934	[CPO]	nov 2013	934	[Q]	mar 2013

Barnes							
Instance	n x m	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date
mt10c1	10 x 11	927	[13]	Sep 2013	927	[3]	Oct 2008
mt10cc	10 x 12	908	[13]	Sep 2013	908	[5]	jan 2010
mt10x	10 x 11	918	[13]	Sep 2013	918	[2]	Feb 2000
mt10xx	10 x 12	918	[13]	Sep 2013	918	[2]	Feb 2000
mt10xxx	10 x 13	918	[13]	Sep 2013	918	[2]	Feb 2000
mt10xy	10 x 12	905	[13]	Sep 2013	905	[3]	Oct 2008
mt10xyz	10 x 13	847	[13]	Sep 2013	847	[2]	Feb 2000
setb4c9	15 x 11	914	[13]	Sep 2013	914	[3]	Oct 2008
setb4cc	15 x 12	907	[13]	Sep 2013	907	[2]	Feb 2000
setb4x	15 x 11	925	[13]	Sep 2013	925	[2]	Feb 2000
setb4xx	15 x 12	925	[13]	Sep 2013	925	[2]	Feb 2000
setb4xxx	15 x 13	925	[13]	Sep 2013	925	[2]	Feb 2000
setb4xy	15 x 12	910	[13]	Sep 2013	910	[4]	jan 2011
setb4xyz	15 x 13	902	[13]	Sep 2013	902	[13]	Sep 2013
seti5c12	15 x 16	1169	[13]	Sep 2013	1169	[13]	Sep 2013
seti5cc	15 x 17	1135	[13]	Sep 2013	1135	[13]	Sep 2013
seti5x	15 x 16	1198	[13]	Sep 2013	1198	[14]	mar 2013
seti5xx	15 x 17	1194	[13]	Sep 2013	1194	[13]	Sep 2013
seti5xxx	15 x 18	1194	[13]	Sep 2013	1194	[13]	Sep 2013
seti5xy	15 x 17	1135	[13]	Sep 2013	1135	[13]	Sep 2013
seti5xyz	15 x 18	1125	[13]	Sep 2013	1125	[2]	Feb 2000

Brandimarte							
Instance	n x m	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date
Mk01	10 x 6	40	[13]	Sep 2013	40	[2]	Feb 2000
Mk02	10 x 6	26	[CPO]	nov 2013	26	[2]	Feb 2000
Mk03	15 x 8	204	[1]	Dec 2010	204	[2]	Feb 2000
Mk04	15 x 8	60	[13]	Sep 2013	60	[2]	Feb 2000
Mk05	15 x 4	172	[Q]	jan 2014	172	[3]	Oct 2008
Mk06	10 x 15	57	[CPO]	nov 2013	57	[Q]	Sep 2013
Mk07	20 x 5	139	[Q]	jan 2014	139	[3]	Oct 2008
Mk08	20 x 10	523	[1]	Dec 2010	523	[2]	Feb 2000
Mk09	20 x 10	307	[13]	Sep 2013	307	[2]	Feb 2000
Mk10	20 x 15	189	[Q]	jan 2014	193	[Q]	May 2014

Dauzere							
Instance	n x m	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date
01a	10 x 5	2505	[1]	Dec 2010	2505	[CPO]	nov 2013
02a	10 x 5	2228	[1]	Dec 2010	2228	[Q]	nov 2015
03a	10 x 5	2228	[1]	Dec 2010	2228	[Q]	mar 2013
04a	10 x 5	2503	[1]	Dec 2010	2503	[2]	Feb 2000
05a	10 x 5	2192	[Q]	jan 2014	2204	[Q]	nov 2015
06a	10 x 5	2163	[Q]	jan 2014	2171	[Q]	jan 2016
07a	15 x 8	2216	[CPO]	Dec 2014	2264	[Q]	nov 2015
08a	15 x 8	2061	[1]	Dec 2010	2061	[Q]	nov 2015
09a	15 x 8	2061	[1]	Dec 2010	2061	[CPO]	nov 2013
10a	15 x 8	2212	[CPO]	Dec 2014	2241	[Q]	nov 2015
11a	15 x 8	2018	[Q]	jan 2014	2037	[Q]	jan 2016
12a	15 x 8	1969	[1]	Dec 2010	1984	[Q]	jan 2016
13a	20 x 10	2197	[CPO]	Dec 2014	2239	[Q]	jan 2016
14a	20 x 10	2161	[1]	Dec 2010	2161	[Q]	May 2014
15a	20 x 10	2161	[1]	Dec 2010	2161	[Q]	jan 2014
16a	20 x 10	2193	[CPO]	Dec 2014	2231	[Q]	jan 2016
17a	20 x 10	2088	[1]	Dec 2010	2105	[Q]	jan 2016
18a	20 x 10	2057	[1]	Dec 2010	2070	[Q]	jan 2016

Hurink-edata							
Instance	n x m	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date
edata-abz5	10 x 10	1167	[CPO]	nov 2013	1167	[Q]	mar 2013
edata-abz6	10 x 10	925	[CPO]	nov 2013	925	[Q]	mar 2013
edata-abz7	20 x 15	604	[CPO]	Dec 2014	610	[Q]	nov 2015
edata-abz8	20 x 15	625	[CPO]	Dec 2014	636	[CPO]	nov 2013
edata-abz9	20 x 15	644	[CPO]	nov 2013	644	[CPO]	nov 2013
edata-car1	11 x 5	6176	[CPO]	nov 2013	6176	[Q]	mar 2013
edata-car2	13 x 4	6327	[CPO]	nov 2013	6327	[Q]	mar 2013
edata-car3	12 x 5	6856	[CPO]	nov 2013	6856	[Q]	mar 2013
edata-car4	14 x 4	7789	[Q]	mar 2013	7789	[Q]	mar 2013
edata-car5	10 x 6	7229	[Q]	mar 2013	7229	[Q]	mar 2013
edata-car6	8 x 9	7990	[CPO]	nov 2013	7990	[Q]	mar 2013
edata-car7	7 x 7	6123	[CPO]	nov 2013	6123	[Q]	mar 2013
edata-car8	8 x 8	7689	[CPO]	nov 2013	7689	[Q]	mar 2013
edata-la01	10 x 5	609	[2]	Feb 2000	609	[2]	Feb 2000
edata-la02	10 x 5	655	[2]	Feb 2000	655	[2]	Feb 2000
edata-la03	10 x 5	550	[2]	Feb 2000	550	[2]	Feb 2000
edata-la04	10 x 5	568	[2]	Feb 2000	568	[2]	Feb 2000
edata-la05	10 x 5	503	[2]	Feb 2000	503	[2]	Feb 2000
edata-la06	15 x 5	833	[2]	Feb 2000	833	[2]	Feb 2000
edata-la07	15 x 5	762	[2]	Feb 2000	762	[2]	Feb 2000
edata-la08	15 x 5	845	[2]	Feb 2000	845	[2]	Feb 2000

Table continued from previous page

Instance	n x m	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date
edata-la09	15 x 5	878	[2]	Feb 2000	878	[2]	Feb 2000
edata-la10	15 x 5	866	[2]	Feb 2000	866	[2]	Feb 2000
edata-la11	20 x 5	1103	[13]	Sep 2013	1103	[2]	Feb 2000
edata-la12	20 x 5	960	[2]	Feb 2000	960	[2]	Feb 2000
edata-la13	20 x 5	1053	[2]	Feb 2000	1053	[2]	Feb 2000
edata-la14	20 x 5	1123	[2]	Feb 2000	1123	[2]	Feb 2000
edata-la15	20 x 5	1111	[2]	Feb 2000	1111	[2]	Feb 2000
edata-la16	10 x 10	892	[2]	Feb 2000	892	[2]	Feb 2000
edata-la17	10 x 10	707	[2]	Feb 2000	707	[2]	Feb 2000
edata-la18	10 x 10	842	[2]	Feb 2000	842	[2]	Feb 2000
edata-la19	10 x 10	796	[2]	Feb 2000	796	[2]	Feb 2000
edata-la20	10 x 10	857	[2]	Feb 2000	857	[2]	Feb 2000
edata-la21	15 x 10	1009	[CPO]	nov 2013	1009	[15]	jul 2011
edata-la22	15 x 10	880	[13]	Sep 2013	880	[15]	jul 2011
edata-la23	15 x 10	950	[2]	Feb 2000	950	[2]	Feb 2000
edata-la24	15 x 10	908	[13]	Sep 2013	908	[15]	jul 2011
edata-la25	15 x 10	936	[13]	Sep 2013	936	[13]	Sep 2013
edata-la26	20 x 10	1106	[Q]	mar 2013	1106	[CPO]	nov 2013
edata-la27	20 x 10	1181	[2]	Feb 2000	1181	[15]	jul 2011
edata-la28	20 x 10	1142	[CPO]	nov 2013	1142	[15]	jul 2011
edata-la29	20 x 10	1107	[CPO]	nov 2013	1107	[CPO]	nov 2013
edata-la30	20 x 10	1188	[CPO]	nov 2013	1188	[CPO]	nov 2013
edata-la31	30 x 10	1532	[CPO]	nov 2013	1532	[CPO]	nov 2013
edata-la32	30 x 10	1698	[2]	Feb 2000	1698	[2]	Feb 2000
edata-la33	30 x 10	1547	[2]	Feb 2000	1547	[2]	Feb 2000
edata-la34	30 x 10	1599	[13]	Sep 2013	1599	[2]	Feb 2000
edata-la35	30 x 10	1736	[2]	Feb 2000	1736	[2]	Feb 2000
edata-la36	15 x 15	1160	[13]	Sep 2013	1160	[15]	jul 2011
edata-la37	15 x 15	1397	[13]	Sep 2013	1397	[2]	Feb 2000
edata-la38	15 x 15	1141	[13]	Sep 2013	1141	[13]	Sep 2013
edata-la39	15 x 15	1184	[13]	Sep 2013	1184	[2]	Feb 2000
edata-la40	15 x 15	1144	[13]	Sep 2013	1144	[13]	Sep 2013
edata-mt06	6 x 6	55	[2]	Feb 2000	55	[2]	Feb 2000
edata-mt10	10 x 10	871	[2]	Feb 2000	871	[2]	Feb 2000
edata-mt20	20 x 5	1088	[2]	Feb 2000	1088	[2]	Feb 2000
edata-orb1	10 x 10	977	[CPO]	nov 2013	977	[CPO]	nov 2013
edata-orb10	10 x 10	933	[CPO]	nov 2013	933	[Q]	mar 2013
edata-orb2	10 x 10	865	[CPO]	nov 2013	865	[Q]	mar 2013
edata-orb3	10 x 10	951	[CPO]	nov 2013	951	[Q]	mar 2013
edata-orb4	10 x 10	984	[CPO]	nov 2013	984	[Q]	mar 2013
edata-orb5	10 x 10	842	[CPO]	nov 2013	842	[Q]	mar 2013

Table continued from previous page

Instance	n x m	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date
edata-orb6	10 x 10	958	[CPO]	nov 2013	958	[Q]	mar 2013
edata-orb7	10 x 10	389	[CPO]	nov 2013	389	[Q]	mar 2013
edata-orb8	10 x 10	894	[Q]	mar 2013	894	[Q]	mar 2013
edata-orb9	10 x 10	933	[CPO]	nov 2013	933	[Q]	mar 2013

Hurink-rdata							
Instance	n x m	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date
rdata-abz5	10 x 10	954	[CPO]	nov 2013	954	[Q]	mar 2013
rdata-abz6	10 x 10	807	[Q]	mar 2013	807	[Q]	mar 2013
rdata-abz7	20 x 15	493	[Q]	mar 2013	524	[Q]	jan 2016
rdata-abz8	20 x 15	507	[Q]	mar 2013	536	[Q]	jan 2016
rdata-abz9	20 x 15	517	[CPO]	Dec 2014	536	[Q]	jan 2016
rdata-car1	11 x 5	5034	[CPO]	nov 2013	5034	[CPO]	nov 2013
rdata-car2	13 x 4	5985	[CPO]	nov 2013	5985	[CPO]	nov 2013
rdata-car3	12 x 5	5622	[Q]	mar 2013	5622	[Q]	nov 2015
rdata-car4	14 x 4	6514	[Q]	mar 2013	6514	[Q]	Sep 2013
rdata-car5	10 x 6	5615	[CPO]	nov 2013	5615	[CPO]	nov 2013
rdata-car6	8 x 9	6147	[CPO]	nov 2013	6147	[Q]	mar 2013
rdata-car7	7 x 7	4425	[CPO]	nov 2013	4425	[Q]	mar 2013
rdata-car8	8 x 8	5692	[CPO]	nov 2013	5692	[Q]	mar 2013
rdata-la01	10 x 5	570	[2]	Feb 2000	570	[CPO]	nov 2013
rdata-la02	10 x 5	529	[2]	Feb 2000	529	[13]	Sep 2013
rdata-la03	10 x 5	477	[2]	Feb 2000	477	[Q]	Aug 2013
rdata-la04	10 x 5	502	[2]	Feb 2000	502	[2]	Feb 2000
rdata-la05	10 x 5	457	[2]	Feb 2000	457	[2]	Feb 2000
rdata-la06	15 x 5	799	[2]	Feb 2000	799	[2]	Feb 2000
rdata-la07	15 x 5	749	[2]	Feb 2000	749	[Q]	Aug 2013
rdata-la08	15 x 5	765	[2]	Feb 2000	765	[2]	Feb 2000
rdata-la09	15 x 5	853	[2]	Feb 2000	853	[2]	Feb 2000
rdata-la10	15 x 5	804	[2]	Feb 2000	804	[2]	Feb 2000
rdata-la11	20 x 5	1071	[2]	Feb 2000	1071	[2]	Feb 2000
rdata-la12	20 x 5	936	[2]	Feb 2000	936	[2]	Feb 2000
rdata-la13	20 x 5	1038	[2]	Feb 2000	1038	[2]	Feb 2000
rdata-la14	20 x 5	1070	[2]	Feb 2000	1070	[2]	Feb 2000
rdata-la15	20 x 5	1089	[2]	Feb 2000	1089	[Q]	Aug 2013
rdata-la16	10 x 10	717	[2]	Feb 2000	717	[2]	Feb 2000
rdata-la17	10 x 10	646	[2]	Feb 2000	646	[2]	Feb 2000
rdata-la18	10 x 10	666	[2]	Feb 2000	666	[2]	Feb 2000
rdata-la19	10 x 10	700	[13]	Sep 2013	700	[2]	Feb 2000
rdata-la20	10 x 10	756	[2]	Feb 2000	756	[2]	Feb 2000
rdata-la21	15 x 10	808	[2]	Feb 2000	825	[Q]	jan 2014

Table continued from previous page

Instance	n x m	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date
rdata-la22	15 x 10	741	[CPO]	Dec 2014	755	[CPO]	nov 2013
rdata-la23	15 x 10	816	[2]	Feb 2000	832	[14]	mar 2013
rdata-la24	15 x 10	775	[2]	Feb 2000	796	[Q]	nov 2015
rdata-la25	15 x 10	768	[CPO]	Dec 2014	783	[Q]	jan 2016
rdata-la26	20 x 10	1056	[2]	Feb 2000	1057	[Q]	jan 2016
rdata-la27	20 x 10	1085	[2]	Feb 2000	1085	[Q]	nov 2015
rdata-la28	20 x 10	1075	[2]	Feb 2000	1076	[Q]	Feb 2014
rdata-la29	20 x 10	993	[2]	Feb 2000	994	[Q]	jan 2014
rdata-la30	20 x 10	1068	[2]	Feb 2000	1071	[Q]	jan 2016
rdata-la31	30 x 10	1520	[2]	Feb 2000	1520	[Q]	Sep 2013
rdata-la32	30 x 10	1657	[2]	Feb 2000	1657	[Q]	Feb 2014
rdata-la33	30 x 10	1497	[2]	Feb 2000	1497	[Q]	May 2014
rdata-la34	30 x 10	1535	[2]	Feb 2000	1535	[Q]	Aug 2013
rdata-la35	30 x 10	1549	[2]	Feb 2000	1549	[CPO]	nov 2013
rdata-la36	15 x 15	1023	[13]	Sep 2013	1023	[13]	Sep 2013
rdata-la37	15 x 15	1062	[CPO]	nov 2013	1062	[CPO]	nov 2013
rdata-la38	15 x 15	954	[13]	Sep 2013	954	[13]	Sep 2013
rdata-la39	15 x 15	1011	[13]	Sep 2013	1011	[13]	Sep 2013
rdata-la40	15 x 15	955	[2]	Feb 2000	955	[CPO]	nov 2013
rdata-mt06	6 x 6	47	[2]	Feb 2000	47	[2]	Feb 2000
rdata-mt10	10 x 10	686	[Q]	mar 2013	686	[2]	Feb 2000
rdata-mt20	20 x 5	1022	[2]	Feb 2000	1022	[2]	Feb 2000
rdata-orb1	10 x 10	746	[CPO]	nov 2013	746	[Q]	mar 2013
rdata-orb10	10 x 10	742	[CPO]	nov 2013	742	[Q]	mar 2013
rdata-orb2	10 x 10	696	[CPO]	nov 2013	696	[Q]	mar 2013
rdata-orb3	10 x 10	712	[CPO]	nov 2013	712	[Q]	mar 2013
rdata-orb4	10 x 10	753	[Q]	mar 2013	753	[Q]	mar 2013
rdata-orb5	10 x 10	639	[CPO]	nov 2013	639	[Q]	mar 2013
rdata-orb6	10 x 10	754	[CPO]	nov 2013	754	[Q]	mar 2013
rdata-orb7	10 x 10	302	[CPO]	nov 2013	302	[Q]	mar 2013
rdata-orb8	10 x 10	639	[CPO]	nov 2013	639	[Q]	mar 2013
rdata-orb9	10 x 10	694	[Q]	mar 2013	694	[Q]	mar 2013

Hurink-vdata

Instance	n x m	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date
vdata-abz5	10 x 10	859	[Q]	mar 2013	859	[Q]	mar 2013
vdata-abz6	10 x 10	742	[Q]	mar 2013	742	[Q]	mar 2013
vdata-abz7	20 x 15	492	[Q]	mar 2013	492	[Q]	jan 2014
vdata-abz8	20 x 15	506	[Q]	mar 2013	507	[Q]	Feb 2014
vdata-abz9	20 x 15	497	[Q]	mar 2013	497	[Q]	jan 2016
vdata-car1	11 x 5	5005	[Q]	mar 2013	5005	[Q]	jan 2014

Table continued from previous page

Instance	n x m	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date
vdata-car2	13 x 4	5929	[Q]	mar 2013	5929	[Q]	Sep 2013
vdata-car3	12 x 5	5597	[Q]	mar 2013	5597	[CPO]	jan 2014
vdata-car4	14 x 4	6514	[Q]	mar 2013	6514	[CPO]	nov 2013
vdata-car5	10 x 6	4909	[Q]	mar 2013	4911	[Q]	nov 2015
vdata-car6	8 x 9	5486	[Q]	mar 2013	5486	[Q]	mar 2013
vdata-car7	7 x 7	4281	[Q]	mar 2013	4281	[Q]	mar 2013
vdata-car8	8 x 8	4613	[Q]	mar 2013	4613	[Q]	mar 2013
vdata-la01	10 x 5	570	[2]	Feb 2000	570	[2]	Feb 2000
vdata-la02	10 x 5	529	[2]	Feb 2000	529	[2]	Feb 2000
vdata-la03	10 x 5	477	[2]	Feb 2000	477	[2]	Feb 2000
vdata-la04	10 x 5	502	[2]	Feb 2000	502	[2]	Feb 2000
vdata-la05	10 x 5	457	[2]	Feb 2000	457	[2]	Feb 2000
vdata-la06	15 x 5	799	[2]	Feb 2000	799	[2]	Feb 2000
vdata-la07	15 x 5	749	[2]	Feb 2000	749	[2]	Feb 2000
vdata-la08	15 x 5	765	[2]	Feb 2000	765	[2]	Feb 2000
vdata-la09	15 x 5	853	[2]	Feb 2000	853	[2]	Feb 2000
vdata-la10	15 x 5	804	[2]	Feb 2000	804	[2]	Feb 2000
vdata-la11	20 x 5	1071	[2]	Feb 2000	1071	[2]	Feb 2000
vdata-la12	20 x 5	936	[2]	Feb 2000	936	[2]	Feb 2000
vdata-la13	20 x 5	1038	[2]	Feb 2000	1038	[2]	Feb 2000
vdata-la14	20 x 5	1070	[2]	Feb 2000	1070	[2]	Feb 2000
vdata-la15	20 x 5	1089	[2]	Feb 2000	1089	[2]	Feb 2000
vdata-la16	10 x 10	717	[2]	Feb 2000	717	[2]	Feb 2000
vdata-la17	10 x 10	646	[2]	Feb 2000	646	[2]	Feb 2000
vdata-la18	10 x 10	663	[2]	Feb 2000	663	[2]	Feb 2000
vdata-la19	10 x 10	617	[2]	Feb 2000	617	[2]	Feb 2000
vdata-la20	10 x 10	756	[2]	Feb 2000	756	[2]	Feb 2000
vdata-la21	15 x 10	800	[2]	Feb 2000	800	[CPO]	jan 2014
vdata-la22	15 x 10	733	[2]	Feb 2000	733	[CPO]	nov 2013
vdata-la23	15 x 10	809	[2]	Feb 2000	809	[Q]	jan 2014
vdata-la24	15 x 10	773	[2]	Feb 2000	773	[Q]	Feb 2014
vdata-la25	15 x 10	751	[2]	Feb 2000	751	[Q]	May 2014
vdata-la26	20 x 10	1052	[2]	Feb 2000	1052	[Q]	Sep 2013
vdata-la27	20 x 10	1084	[2]	Feb 2000	1084	[CPO]	nov 2013
vdata-la28	20 x 10	1069	[2]	Feb 2000	1069	[Q]	Sep 2013
vdata-la29	20 x 10	993	[2]	Feb 2000	993	[Q]	Sep 2013
vdata-la30	20 x 10	1068	[2]	Feb 2000	1068	[Q]	jan 2014
vdata-la31	30 x 10	1520	[2]	Feb 2000	1520	[2]	Feb 2000
vdata-la32	30 x 10	1657	[2]	Feb 2000	1657	[Q]	mar 2013
vdata-la33	30 x 10	1497	[2]	Feb 2000	1497	[2]	Feb 2000
vdata-la34	30 x 10	1535	[2]	Feb 2000	1535	[2]	Feb 2000

Table continued from previous page

Instance	n x m	LB	LB Ref.	LB Date	UB	UB Ref.	UB Date
vdata-la35	30 x 10	1549	[2]	Feb 2000	1549	[2]	Feb 2000
vdata-la36	15 x 15	948	[2]	Feb 2000	948	[2]	Feb 2000
vdata-la37	15 x 15	986	[2]	Feb 2000	986	[2]	Feb 2000
vdata-la38	15 x 15	943	[2]	Feb 2000	943	[2]	Feb 2000
vdata-la39	15 x 15	922	[2]	Feb 2000	922	[2]	Feb 2000
vdata-la40	15 x 15	955	[2]	Feb 2000	955	[2]	Feb 2000
vdata-mt06	6 x 6	47	[2]	Feb 2000	47	[2]	Feb 2000
vdata-mt10	10 x 10	655	[2]	Feb 2000	655	[2]	Feb 2000
vdata-mt20	20 x 5	1022	[2]	Feb 2000	1022	[2]	Feb 2000
vdata-orb1	10 x 10	695	[Q]	mar 2013	695	[Q]	mar 2013
vdata-orb10	10 x 10	681	[Q]	mar 2013	681	[Q]	mar 2013
vdata-orb2	10 x 10	620	[Q]	mar 2013	620	[Q]	mar 2013
vdata-orb3	10 x 10	648	[Q]	mar 2013	648	[Q]	mar 2013
vdata-orb4	10 x 10	753	[Q]	mar 2013	753	[Q]	mar 2013
vdata-orb5	10 x 10	584	[Q]	mar 2013	584	[Q]	mar 2013
vdata-orb6	10 x 10	715	[Q]	mar 2013	715	[Q]	mar 2013
vdata-orb7	10 x 10	275	[Q]	mar 2013	275	[Q]	mar 2013
vdata-orb8	10 x 10	573	[Q]	mar 2013	573	[Q]	mar 2013
vdata-orb9	10 x 10	659	[Q]	mar 2013	659	[Q]	mar 2013

Referências dos Dados das Tabelas

- [1] Abir Ben Hmida et al. "Discrepancy search for the exible job shop scheduling problem". In: *Computers & OR* 37.12 (2010), pp. 2192-2201.
- [2] Monaldo Mastrolilli and Luca Maria Gambardella. "Effective Neighborhood Functions for the Flexible Job Shop Problem: Appendix". 2000. url: <http://www.idsia.ch/monaldo/fjspresults/fjspresults.pdf>.
- [3] Guohui Zhang et al. "Variable Neighborhood Genetic Algorithm for the Flexible Job Shop Scheduling Problems". In: *Intelligent Robotics and Applications*. Ed. by Caihua Xiong et al. Vol. 5315. *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, pp. 503-512. isbn: 978-3-540-88516-0. DOI: 10.1007/978-3-540-88518-4_54. url: https://dx.doi.org/10.1007/978-3-540-88518-4_54.
- [4] Angelo Oddi et al. "Iterative Flattening Search for the Flexible Job Shop Scheduling Problem". In: *IJCAI*. 2011, pp. 1991-1996. url: <https://ijcai.org/papers11/Papers/IJCAI11-332.pdf>.
- [5] Wojciech Bożejko, Mariusz Uchroński, and Mieczyslaw Wodecki. "Parallel metaheuristics for the exible job shop problem". In: *Proceedings of the 10th international conference on Artificial intelligence and soft computing: Part II. ICAISC'10*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 395-402. isbn: 3-642-13231-6, 978-3-642-13231-5. url: <https://dl.acm.org/citation.cfm?id=1876119.1876171>.
- [6] G. I. Zobolas, Christos D. Tarantilis, and George Ioannou. "Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm". In: *Computers & OR* 36.4 (2009), pp. 1249-1267.
- [7] Martín Gómez Ravetti et al. "Parallel hybrid heuristics for the permutation flow shop problem". In: *Annals OR* 199.1 (2012), pp. 269-284.
- [8] Anurag Agarwal, Selcuk Colak, and Enes Eryarsoy. "Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach". In: *European Journal of Operational Research* 169.3 (2006), pp. 801-815.
- [9] Francis Sourd and Wim Nuijten. "Multiple-Machine Lower Bounds for Shop-Scheduling Problems". In: *INFORMS J. on Computing* 12.4 (Oct.2000), pp. 341-352. ISSN: 1526-5528. DOI: 10.1287/ijoc.12.4.341.11881. url: <https://dx.doi.org/10.1287/ijoc.12.4.341.11881>.
- [10] Pei-Chann Chang et al. "Artificial chromosomes embedded in genetic algorithm for a chip resistor scheduling problem in minimizing the makespan". In: *Expert Syst. Appl.* 36.3 (2009), pp. 7135-7141.
- [11] Robin Kumar Samuel and P. Venkumar. "Some Novel Methods for Flow Shop Scheduling". In: *International Journal of Engineering Science and Technology (IJEST)* 3.12 (2011), pp. 8395-8403. url: <https://www.ijest.info/docs/IJEST11-03-12-198.pdf>.
- [12] Mircea Ancu. "On solving Flowshop Scheduling Problems". In: *Proceedings of the Romanian Academy*. Vol. 13(1). 2012, pp. 71-79. url: <http://www.acad.ro/sectii2002/proceedings/doc2012-1/10-Ancau.pdf>.
- [13] Andreas Schutt, Thibaut Feydy, and Peter J Stuckey. "Scheduling Optional Tasks with Explanation". In: *Principles and Practice of Constraint Programming*. Springer. 2013, pp. 628-644. url: <https://www2.cs.mu.oz.au/pjs/papers/cp2013c.pdf>.
- [Q] Quintiq. "Flexible Job Shop Scheduling Problem". url: <https://www.quintiq.com/optimization/fjssp-world-records.html>.
- [CPO] IBM. Solving Flexible Job Shop Scheduling Problems (CP Optimizer 12.6). url: https://www.ibm.com/developerworks/community/blogs/jfp/entry/solving_flexible_job_shop_scheduling_problems?lang=en.