Mailla Cristine Spricigo

Aplicação de Algoritmos Genéticos para o Problema de Escalonamento de Tarefas em Sistemas de Manufatura com Controle Supervisório e Autômatos com Parâmetros

Dissertação de Metrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica e Computação. Área de concentração: Sistemas Dinâmicos e Energéticos.

Orientador: Prof. Dr. Romeu Reginato

Co-orientador: Prof. Dr. Sandro Battistella

Foz do Iguaçu

2018

Ficha de identificação da obra elaborada através do Formulário de Geração Automática do Sistema de Bibliotecas da Unioeste.

Spricigo, Mailla Cristine
Aplicação de Algoritmos Genéticos para o Problema de Escalonamento de Tarefas em Sistemas de Manufatura com Controle Supervisório e Autômatos com Parâmetros / Mailla Cristine Spricigo; orientador(a), Romeu Reginato; coorientador(a), Sandro Battistella, 2018.
126 f.

Dissertação (mestrado), Universidade Estadual do Oeste do Paraná, Centro de Engenharias e Ciências Exatas, Programa de Pós-Graduação em Engenharia Elétrica e Computação, 2018.

1. Sistemas a Eventos Discretos. 2. Escalonamento de Tarefas em Sistemas de Manufatura. 3. Algoritmo Genético. 4. Controle Supervisório. I. Reginato, Romeu . II. Battistella, Sandro. III. Título.

Aplicação de Algoritmos Genéticos para o Problema de Escalonamento de Tarefas em Sistemas de Manufatura com Controle Supervisório e Autômatos com Parâmetros

Mailla Cristine Spricigo

Esta Dissertação de Mestrado foi apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Computação e aprovada pela Banca Examinadora:
Data da defesa pública: 31/08/2018.

Prof. Dr. Romeu Reginatto (Orientador)
Universidade Estadual do Oeste do Paraná (Unioeste)

Prof. Dr. Sandro Battistella (Co-orientador)
Universidade Estadual do Oeste do Paraná (Unioeste)

Prof. Dr. Guilherme Kunz
Universidade Estadual do Oeste do Paraná (Unioeste)

Prof. Dr. César Rafael Claure Torrico
Universidade Tecnológica Federal do Paraná (UTFPR)

Resumo

O escalonamento de tarefas, também conhecido como *job-shop scheduling*, visa encontrar sequências ótimas de eventos que permitam aumentar os índices de produtividade em um sistema de produção. Este trabalho emprega algoritmos genéticos como metaheurística para obtenção de soluções ótimas de escalonamento, combinado com a teoria do controle supervisório (TCS) na geração automática de sequências de operações na produção de peças em um sistema didático de manufatura. A abordagem da TCS permite derivar estruturas de controle (supervisores) que formalmente garantem o funcionamento correto e seguro da planta. Por sua vez, o algoritmo genético busca pela melhor sequência possível de eventos, entre todas as sequências habilitadas pelos supervisores, explorando o paralelismo da planta e, assim, permitindo cumprir com objetivos de produção ao mesmo tempo que atende especificações de segurança e funcionamento. Um exemplo baseado em uma célula didática de manufatura ilustra a abordagem empregada, onde é possível observar a melhora do escalonamento de tarefas em comparação ao escalonamento sequencial. Três diferentes representações para o cromossomo foram utilizadas, sendo elas, a representação baseada em operações, a representação em chaves aleatórias e a representação baseada em regras de prioridade. Os resultados obtidos nas simulações do estudo de caso demonstram a eficácia na utilização do escalonamento de produção baseada em algoritmos genéticos em conjunto com a utilização da TCS para a descrição das restrições do sistema.

Palavras-chaves: Sistemas a eventos discretos, sistemas de manufatura, controle supervisório, algoritmo genético; *job-shop scheduling*.

Abstract

The job-shop scheduling aims to find optimal sequences of events that increases productivity indexes in a manufacturing systems. This work employs such metaheuristics to obtain optimal scheduling solutions, combined with the supervisory control theory (SCT) in the automatically generation of sequences of operations involved in the production of parts in a didactic manufacturing system. The SCT allows deriving control structures (supervisors) that formally guarantee the plant operation correctness and safeness. In turn, the genetic algorithm searches for the best possible sequence of events, among all enabled by the supervisors, exploring the plant parallelism, allowing to meet production goals while complying with safety and operational specifications. An example based on a didactic manufacturing cell illustrates the used approach where it is possible to observe the improvement of the task scheduling compared to sequential scheduling. Three different representations for the chromosome were used, which are the operation-based representation, random keys representation and priority rule-based representation. The obtained results in the case study simulation demonstrate the efficiency in the use of the job-shop-scheduling based in genetic algorithms with the use the SCT for description of the system restrictions.

Keywords: Discrete Event Systems, manufacturing systems, supervisory control, genetic algorithm, job-shop scheduling.

Dedico aos meus pais e ao Henrique que sem vocês, este trabalho não seria possível.

Agradecimentos

Primeiramente, a Deus por me conceder a minha vida e a oportunidade de poder estudar e realizar este trabalho.

Aos meus pais, Gislaine e Ercio, pelo amor, paciência, pelas cobranças e por, principalmente, sempre buscarem o melhor para mim e para meu irmão Lucas e incentivarem tanto a vida acadêmica. Agradeço também ao meu irmão, pelo companheirismo e bom humor nos momentos de estresse.

Ao meu noivo, Henrique, pelo companheirismo, apoio e amor incondicional. Por embarcar comigo nessa e em todas futuras etapas de nossas vidas e estar sempre ao meu lado.

Aos meus avós que sempre se fizeram presentes em toda minha vida e me incentivaram neste caminho, principalmente pelas orações e torcida.

Ao Professor Sandro, por todo apoio, orientações e contribuições que ajudar a idealizar e realizar este trabalho. Ao Professor Romeu, pela orientação e ajuda que forneceu sempre que necessário.

Aos meus amigos que sempre apoiaram, dividiram experiências e trouxeram bons momentos em meio a algumas dificuldades.

Ao meu colega de mestrado e amigo Cristiano, por se fazer sempre presente nesses dois anos, pela amizade e ajuda em todos os trabalhos.

Ao Programa de Pós-graduação em Engenharia Elétrica e Computação, pela oportunidade de estudo e desenvolvimento desse trabalho.

A Fundação Araucária pela concessão da bolsa que foi um auxilio durante toda a realização do trabalho.

A todos, por fim, que de uma maneira ou outra incentivaram e apoiaram durante os dois anos do mestrado.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Símbolos

α – Evento

β – Evento

Γ – Estrutura de controle

γ – Entrada de controle

δ – Função de Transição Parcial

ε – Cadeia Vazia

λ – Evento

Σ – Alfabeto

$\mathit{\Sigma}$ – Conjunto Finito de Eventos (autômatos)

σ – Símbolo

Σ_c – Conjunto de eventos controláveis

Σ_u – Conjunto de eventos não-controláveis

A – Conjunto das Atividades

AG – Algoritmo Genético

a_0 – Atividade inicial

A_m – Conjunto de atividades marcadas

CLP – Controlador Lógico Programável

CSM – Controle Supervisório Modular

CSML – Controle Supervisório Modular Local

D – Dupla

E – Composição Assíncrona das Especificações

G – Autômato/Planta

g – Guarda

G – Conjunto de guardas

GTA – Gráficos de Transição de Atividade

GTT – Gráficos de Transição Temporizada

IA – Inteligência Artificial

IHM – Interface Homem-Máquina

JSP – *Job-shop scheduling* (escalonamento de tarefas)

L – Linguagem Gerada

$\bar{L}$ – Linguagem Prefixo-fechada

L_a – Linguagem com máximo comportamento admissível

L_m – Linguagem Marcada

L_r – Linguagem com mínimo comportamento requerido

l_a – Tempo de limite inferior

m – Número de máquinas

MEF – Máquina de Estados Finitos

MEFcP – Máquina de Estados Finitos com Parâmetros

n – Número de tarefas

o_{nm} – Operação na máquina m da tarefa n

p – Parâmetro

P – Espaço Vetorial

R – Autômato resultante do processo de composição

S – Supervisor

SCADA – Sistema de Supervisão e Aquisição de Dados

SED – Sistema a Eventos Discretos

SEDT – Sistemas a Eventos Discretos Temporizados

SFM – Sistema Flexível de Manufatura

S/G – Sistema a ser controlado

TCS – Teoria de Controle Supervisório

TCS-RW – Teoria de Controle Supervisório de Ramadge e Wonham

t – Símbolo

t – Tempo

u – Símbolo

u_a – Tempo de limite superior

v – Símbolo

x – Estado

x – Duração

x_0 – Estado inicial

X – Conjunto finito de estados

X_m – Conjunto de estados marcados

Capítulo 1

Introdução

1.1 Motivação e Justificativa

Um Sistema Flexível de Manufatura (SFM) é um sistema integrado e complexo, onde um computador controla dispositivos de manuseamento de materiais automatizados e ferramentas de máquinas numericamente controladas e que podem processar simultaneamente volumes médios de diferentes tipos de peças (Browne, 1984). Devido ao crescimento rápido de tecnologias em computadores, SFMs têm recebido mais atenção nas últimas décadas emergindo assim como uma das mais importantes chaves para que uma indústria encontre o sucesso organizacional (Chan & Chan, 2002).

SFMs se enquadram em uma classe mais ampla de sistemas dinâmicos, denominada Sistemas a Eventos Discretos (SED). SEDs são sistemas dinâmicos que evoluem com a ocorrência abrupta de ocorrências físicas em seu ambiente (denominadas de eventos), em intervalos de tempo, em geral, irregulares e desconhecidos (Cassandras & Lafortune, 2008). Em virtude da tendência do crescimento do setor de serviços nos cenários nacional e mundial, aliada à complexidade geralmente atribuída a esses sistemas, tem-se estimulado a realização de pesquisas nessa área. Foram criadas diversas abordagens para a modelagem desses tipos de sistemas, sendo que essas abordagens refletem diferentes tipos de SEDs e possuem diferentes objetivos de análise. Esses modelos matemáticos devem descrever adequadamente o comportamento do sistema e prover um quadro geral para as técnicas analíticas para alcançar os objetivos de *design*, controle e aumento de performance (Cassandras & Lafortune, 2008). Uma dessas abordagens é a Teoria de Controle Supervisório (TCS).

Em particular, Ramadge e Wonham (1989) desenvolveram uma abordagem de controle supervisório, conhecida como TCS-RW, representada nesse trabalho como TCS. Nessa abordagem, um sistema a ser controlado, chamado de planta, corresponde, em geral, a um conjunto de subsistemas, geralmente equipamentos, que estão arranjados segundo uma distribuição espacial dada e a composição dos comportamentos de cada um dos subsistemas isolados (Cury, 2001). Para fazer com que esses subsistemas trabalhem de maneira coordenada de acordo com requisitos de operação, pode ser necessário alterar o comportamento da planta (malha-aberta), introduzindo, assim, um agende externo, chamado de supervisor. A TCS emprega linguagens e autômatos como formalismo para representação dos modelos matemáticos de sistemas a eventos discretos.

A TCS estabelece que todos os eventos são gerados de maneira espontânea pelo sistema em malha-aberta (planta), sendo que a ocorrência de alguns eventos pode ser evitada (Ramadge & Wonham, 1989). O supervisor, ou controlador, atua na planta em uma estrutura em malha-fechada, observando a evolução dinâmica da mesma, e exercendo o controle da planta através da proibição de ocorrência de alguns eventos. Os supervisores são obtidos através do processo de síntese da Teoria de Controle Supervisório, criado por Ramadge e Wonham (1989), e que fornece estruturas em malha fechada que limitam o comportamento dinâmico da planta (malha-aberta) ao comportamento considerado seguro e desejado (malha-fechada).

Entretanto, o processo de obtenção de supervisores pode enfrentar um alto custo computacional, para sistemas relativamente grandes, uma vez que o número de estados dos autômatos cresce exponencialmente com a composição de subsistemas e especificações (Queiroz & Cury, 2002). Para contornar tal problema, neste trabalho, se propõe a utilização da abordagem de controle modular local proposta por Queiroz & Cury (2002), que reduz o problema de explosão de estados.

Um SFM pode produzir múltiplos tipos de produtos utilizando diversos recursos, como robôs, máquinas com diversas funções, entre outros (Lee & DiCesare, 1994). O SFM apresenta flexibilidade pois pode fornecer um grande número de escolhas de recursos e de distribuição de ordem nas tarefas, impondo um problema conhecido como escalonamento de tarefas dentro da área de gestão de produção.

O problema de escalonamento de tarefas, conhecido também como *job-shop scheduling problem* (JSP) tem provido uma área de pesquisa muito ativa tendo como foco encontrar maneiras de distribuir e atribuir tarefas para as máquinas de tal forma que certos critérios sejam a-

tendidos e determinada função objetivo seja otimizada (Gao *et al*.,2014). Uma função objetivo representa um custo ou benefício a ser otimizado, e será utilizada no JSP utilizado neste trabalho para minimizar o tempo de produção do sistema.

A importância do escalonamento de tarefas tem crescido nos últimos anos devido ao crescimento da demanda por variedade, ciclo de vida de produtos reduzidos, mudanças no mercado e rápido desenvolvimento de novos processos e tecnologias (Ho, Tay & Lai, 2006). Por isso, o escalonamento de tarefas tem sido estudado extensivamente durante as últimas décadas e o objetivo é encontrar uma ordem de processamento ou regra de escalonamento em cada máquina buscando que a medida de desempenho seja ótima (Ramasesh, 1989).

Existem muitos motivos pelo qual um SFM deve desenvolver um escalonamento na sua produção, pois se a mesma for organizada, executada de maneira adequada e se houver escalonamento das atividades envolvidas, consegue-se obter produções na quantidade desejada, na qualidade exigida, no tempo necessário e com mínimo custo (Becerra & Coello, 2005).

Entretanto, o problema do escalonamento de tarefas, apesar de ser um problema comumente existente na indústria, é de resolução complexa por ser influenciado pelo número de tarefas, diferenças de critérios e função objetivo (Gao *et al*.,2014). Como o escalonamento de tarefas envolve centenas de máquinas e milhares de tarefas para serem atribuídas e distribuídas, o problema do escalonamento se torna exponencial e dificultando a aplicação de metodologias, é considerado um problema NP-difícil (Hoitomt, Uh & Pattipati, 1993), isto é, um problema em que é improvável a existência de algum algoritmo exato com convergência em tempo polinomial[1]. Por isso, metodologias de escalonamento requerem custo e tempo de análise e execução.

Ao comparar a otimização tradicional e métodos iterativos, as metaheurísticas usualmente podem encontrar boas soluções com menor esforço computacional e conseguem amenizar a dificuldade que alguns métodos heurísticos têm de escapar de ótimos locais. Heurísticas procuram alcançar resultados para os problemas rapidamente, mas não garantem que a solução obtida seja a solução ótima, mas apresentam resultados satisfatórios em um tempo computacional aceitável. As metaheurísticas também são métodos de solução que coordenam procedimentos de busca locais com estratégias de mais alto nível, de modo a criar um processo capaz de escapar

[1] O algoritmo é de tempo polinomial se o tempo de execução dele é limitado superiormente por uma expressão polinomial do tamanho da entrada (n) do algoritmo, onde o tempo T(n)=O(nk) depende da complexidade do cálculo do tempo (O) e de algumas constantes k (Rodrigues, 2000).

de mínimos/máximos locais e realizar uma busca robusta no espaço de solução de um problema. O sucesso de metaheurísticas tem como base a capacidade de proporcionar um equilíbrio entre a diversificação e intensificação, ou seja, de possuir habilidades de computação rápida e encontrar boas soluções com menor esforço computacional. Por esse motivo, metaheurísticas têm sido amplamente aplicadas aos problemas de escalonamento de tarefas (Gao *et al*,2014). Uma outra dificuldade da abordagem do escalonamento de tarefas está relacionada precisamente com a descrição analítica de restrições do problema. Restrições essas que são necessárias para garantir que as sequências de produção sejam consideradas "legais" (possíveis de serem realizadas na planta) ou factíveis. Tem-se, então, o controle supervisório que objetiva a construção de controladores que garantem que somente as sequências "legais" ocorram na planta. A partir da TCS, é possível obter uma descrição formal do sistema e que, sob a ação dos supervisores, gera uma sequência de eventos que atende às especificações de funcionamento e segurança. Essa sequência de eventos é, pelo processo de síntese da TCS, ótima no sentido da ação dos supervisores ser minimamente restritiva, o que significa que somente aquelas sequências de eventos consideradas indesejadas são impedidas de ocorrer (Ramadge & Wonham, 1989). Portanto, os supervisores são responsáveis em manter o funcionamento da planta de acordo com quesitos de operação e segurança.

Durante o processo de escalonamento de tarefas, é possível empregar o conjunto de sequências possíveis descritas pelo formalismo da TCS como base para a solução do JSP, através da otimização utilizando metaheurísticas. Desse modo, garante-se que a sequência encontrada no processo de escalonamento de tarefas seja uma boa solução, senão a ótima, além de factível, "legal" ou permitida pelos supervisores.

Sistemas Flexíveis de Manufatura (SFMs) são, em sua grande maioria, altamente automatizados e buscam integrar seus diversos níveis gerenciais e operacionais. Neste contexto, aparecem os sistemas de supervisão e aquisição de dados (SCADA), que monitoram e supervisionam o funcionamento correto dos equipamentos e também possibilitam a integração entre sistemas de gerenciamento de produção e o controle do chão de fábrica (Groover, 2011).

Diante disso, é possível observar a grande relevância em um SFM do escalonamento da produção, que busca organizar e otimizar a produção agregando benefícios como uma maior produtividade com menor custo, e ao mesmo tempo, a grande importância dos sistemas de con-

trole do chão de fábrica. A existência de tecnologia de automação e comunicação nestes dois níveis permite a integração do escalonamento de tarefas e a Teoria do Controle Supervisório por meio de sistemas SCADA.

Assim, um estudo da utilização de sistemas Scada para a integração de duas grandes áreas altamente complexas dentro de um SFM, uma gerencial e outra operacional, além de possuir grande valor acadêmico, é muito útil para aplicação em industrias, possibilitando que estas consigam ter uma produção bem planejada e livre de falhas.

1.2 Objetivos

Neste contexto, a proposta do presente trabalho consiste na elaboração de uma metodologia para a resolução de problemas de escalonamento de tarefas em SFMs cujo comportamento dinâmico dirigido a eventos de uma planta é modelado de acordo com a TCS, em particular a do controle supervisório modular local (CSML).

Para cumprir o objetivo geral do trabalho, foram definidos os seguintes objetivos específicos:

- Realização uma revisão bibliográfica sobre sistemas a eventos discretos, com ênfase na Teoria de Controle Supervisório (TCS);
- Realização uma revisão bibliográfica sobre o problema de escalonamento de tarefas e a utilização da metaheurística algoritmos genéticos para a sua resolução, comparando as principais representações dos cromossomos;
- Elaboração da metodologia proposta por Pena *et al.* (2015) para o emprego do problema de escalonamento de tarefas em conjunto com a TCS;
- Como forma de validação da metodologia proposta acima, propõe-se a aplicação de um estudo de caso, nos quais são necessários os seguintes passos:
    - Modelagem da planta escolhida para estudo de caso;
    - Implementação dos algoritmos genéticos com as diferentes representações de cromossomos;
    - Avaliação de resultados.

Baseado no trabalho aqui apresentado, foi submetido e aceito um artigo para o XXII Congresso Brasileiro de Automática 2018, nomeado “Algoritmo Genético para o Problema de Es-

calonamento de Tarefas em Sistemas de Manufatura com Controle Supervisório e Autômatos com Parâmetros".

1.3 Estrutura

Este trabalho está organizado como se segue:

- O capítulo 2 discute os principais conceitos sobre a Teoria de Controle Supervisório, a qual é utilizada no desenvolvimento e aplicação da metodologia proposta;
- O capitulo 3 apresenta o problema de escalonamento de tarefas junto as suas dificuldades. Neste capítulo também é apresentado o algoritmo genético, metaheurística utilizada para a resolução do problema de escalonamento.
- No capítulo 4 é, por fim, proposta a metodologia desenvolvida neste trabalho. Para a sua validação, apresenta-se então um sistema em que a metodologia é empregada demonstrando assim sua eficácia, através da análise de seus resultados.
- As conclusões do trabalho e trabalhos futuros são discutidas no capítulo 5.

Capítulo 2

Sistemas a Eventos Discretos e Teoria de Controle Supervisório

Neste capítulo são introduzidos os conceitos fundamentais da Teoria de Controle Supervisório (TCS), proposta inicialmente por Ramadge & Wonham (1987), e da abordagem modular local de Queiroz & Cury (2002). A seção 2.1 apresenta o conceito de Sistemas a Eventos Discretos (SED), abordando com maior ênfase a Teoria das Linguagens e Autômatos. Na seção 2.2 é discutida a Teoria de Controle Supervisório (TCS), em especial o Controle Supervisório Modular Local (CSML). Também na seção 2.2 é retratada a modelagem de sistemas utilizando a TCS proposta por Cury (2002). Na seção 2.3, apresenta-se a representação temporal para a TCS.

2.1 Sistemas a Eventos Discretos

Com o crescente e rápido avanço da tecnologia na área de computação, sensoriamento e comunicação, tem-se aumentado muito a utilização de sistemas que tem por objetivo a execução de tarefas. Tais sistemas percebem as ocorrências no ambiente à sua volta através de estímulos, denominados eventos. Esses eventos, por sua vez, são instantâneos, fato este que confere aos sistemas dinâmicos um caráter discreto no tempo (Costa, 2012). A ocorrência de um evento provoca uma mudança interna no sistema, podendo ou não se manifestar a um observador externo. Assim, ao perceber o acontecimento de um evento, o sistema reage imediatamente, acomodando-se em tempo nulo numa nova situação, onde permanece até que ocorra um novo evento (Cury, 2001). A evolução no tempo desses tipos de sistema pode ser representada por uma trajetória, conforme ilustrado na Figura 2.1, onde os eventos podem ser representados por

α, β e λ e os estados por x_1, x_2, x_3 e x_4. Por exemplo, no caso de ocorrência do evento α no tempo t_1, provoca uma mudança no estado do sistema, no caso de x_1 ao x_4. Sistemas que atendem as propriedades acima descritas podem ser classificados como Sistemas a Eventos Discretos (Cury, 2001).

Um Sistema a Eventos Discretos (SED) é um sistema dinâmico que evolui de acordo com ocorrências abruptas, em intervalos irregulares e geralmente desconhecidos, de eventos físicos (Ramadge & Wonham, 1989; Constain, 2011). Um SED é um tipo de sistema dinâmico, assíncrono (de intervalos não regulares), onde as transições de estados são iniciadas por eventos que ocorrem em instantes discretos do tempo (Nauom *et al.*, 1995).

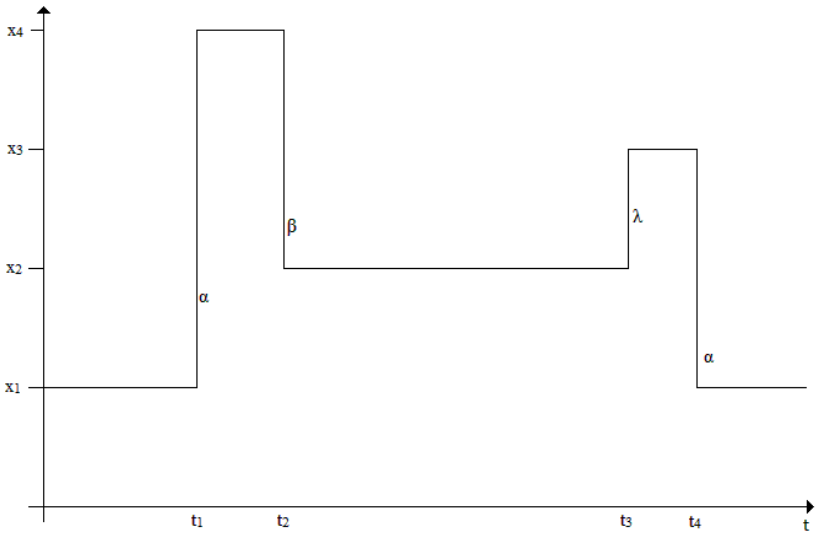

Figura 2.1 - Trajetória típica de um SED. Adaptado de Cury (2001).

Em qualquer sistema, um estado é definido como uma medida mensurável do comportamento do sistema em um instante de tempo *t* e um espaço de estado é o conjunto de todos os possíveis valores que um estado pode assumir. Um SED deve atender duas propriedades: o espaço de estados constitui um conjunto discreto e o mecanismo de transição entre esses estados é classificado como orientado a eventos, ou seja, a ocorrência de uma transição entre estados apenas acontece em pontos discretos do tempo (Cassandras & Lafortune, 2008).

SEDs estão presentes em uma série de aplicações, como na robótica, supervisão de sistemas de tráfego, logística, sistemas operacionais, redes de comunicação de computadores, redes de telecomunicações, concepções de software, gerenciamento de bases de dados e otimização de processos distribuídos (Nauom *et al.*,1995; Cury, 2001). Um exemplo de SED são os Siste-

mas Flexíveis de Manufatura (SFM). Um SFM é um sistema integrado e complexo, onde um computador controla dispositivos automatizados de manuseamento de materiais e ferramentas e de máquinas numericamente controladas e que podem processar simultaneamente volumes médios de diferentes tipos de peças (Browne, 1984). Devido ao crescimento rápido de tecnologias em computadores, SFMs tem recebido mais atenção as últimas décadas emergindo assim como uma das mais importantes chaves para que uma indústria encontre o sucesso organizacional.

Um modelo é uma ferramenta que permite replicar e descrever o comportamento de um sistema e para isso é necessário o desenvolvimento de alguns meios matemáticos (Cassandras & Lafortune, 2008). Devido as suas características específicas, e principalmente à sua natureza discreta, um SED não pode ser tratado pelos modelos matemáticos baseados em equações diferenciais ou a diferenças, normalmente utilizados para se tratar sistemas de variáveis contínuas ou discretas no tempo (Costa, 2012). O espaço de estados de um SED é limitado a um conjunto enumerável, ao passo que em sistemas contínuos ou discretos modelados através de equações diferenciais e a diferenças, o espaço de estado é contínuo ou em intervalos regulares e, portanto, infinito. Sistemas contínuos, em geral, mudam de estado a cada instante, tendo o seu comportamento descrito por uma função que relaciona o estado (variável dependente) ao tempo (variável independente) (Cury, 2001).

A tendência de crescimento do setor de serviços nos cenários nacional e mundial, aliada à complexidade geralmente atribuída aos SEDs, tem estimulado a realização de pesquisas (Sakurada & Miyade, 2009). São encontrados na literatura vários trabalhos que objetivam encontrar modelos matemáticos para a representação precisa de SEDs. Alguns trabalhos podem ser citados, como Modelo RW (Ramadge & Wonham, 1987), Redes de Petri (Murata, 1989), Cadeias de Markov (Glynn, 1989), entre outros.

Neste trabalho será utilizado o modelo RW para a representação de um SED. Este modelo proposto por Ramadge & Wonham (1987) se baseia na Teoria de Linguagens e Autômatos. As seções seguintes apresentam os conceitos sobre a Teoria de Linguagens e Autômatos.

Existem algumas operações que podem ser realizadas com as linguagens. Dentre elas, e apresentadas a seguir, encontram-se: concatenação e prefixo-fechamento. Nas operações a seguir, considera-se a *string s*, constituída por três símbolos *t, u* e *v*, apresentada na expressão 2.2:

$$t\ u\ v = s \tag{2.2}$$

onde,

- *t* é chamado de prefixo de *s;*
- *u* é chamado de subcadeia de *s*;
- *v* é chamado de sufixo de *s*.

- **Concatenação:**

A operação de concatenação consiste na construção de cadeias (Cassandras & Lafortune, 2008). Sejam duas linguagens L_1 e L_2 sobre Σ, a operação de concatenação dessas duas linguagens é definida na expressão 2.3:

$$L_1 L_2 \coloneqq \{s \in \Sigma^* : (s = s_1 s_2)\ e\ (s_1 \in L_1)\ e\ (s_2 \in L_2)\} \quad (2.3)$$

- **Prefixo-fechamento:**

Define-se o conceito de prefixo-fechamento de uma linguagem *L*, denotado por $\bar{L}$ e formalizado pela expressão 2.4 (Costa, 2012):

$$\bar{L} \coloneqq \{s \in \Sigma^* : \exists t \in \Sigma^* (st \in L)\} \quad (2.4)$$

Uma linguagem *L* é dita prefixo-fechada se $L \subseteq \bar{L}$ (Cury, 2001; Sivovella, 2005; Costa, 2012). Dessa forma, o comportamento de um SED pode ser descrito por uma linguagem *L* prefixo-fechada sobre um alfabeto Σ, ou seja, considerando a evolução sequencial de um SED e um alfabeto que corresponde ao conjunto de eventos que afetam ao sistema, para que um sistema produza uma cadeia qualquer *s*, ele deve produzir anteriormente todos os seus prefixos (Cury, 2001; Hopcroft, 2002).

2.1.1 Modelos de Linguagem de Sistemas à Eventos Discretos

Como a dinâmica dos SEDs é regida pela ocorrência de eventos, é conveniente que seu comportamento seja representado por linguagens (Cassandras & Lafortune, 2008). Abaixo seguem alguns conceitos preliminares para o entendimento (Hopcroft *et al.*, 2002; Cassandras & Lafortune, 2008; Costa, 2012): Um símbolo σ é uma entidade gráfica abstrata, podendo representar números, letras, dentre outros;

- Um alfabeto é um conjunto de símbolos finito e não vazio, convencionalmente denotado pelo símbolo Σ;
- Uma palavra *s*, ou uma cadeia (*string*), é uma sequência finita de símbolos sobre um alfabeto Σ;
- Σ^* denota todos os conjuntos possíveis de cadeias finitas compostos pelo alfabeto Σ, incluindo a cadeia vazia ε (cadeia que não contém nenhum evento);

- Uma palavra *s*, ou uma cadeia (*string*), é uma sequência finita de símbolos sobre um alfabeto Σ;
- Σ^* denota todos os conjuntos possíveis de cadeias finitas compostos pelo alfabeto Σ, incluindo a cadeia vazia ε (cadeia que não contém nenhum evento);
- Um conjunto de *strings*, todos escolhidos a partir de algum Σ^*, onde Σ é um alfabeto específico, é chamado de linguagem. A única restrição importante sobre o que pode ser uma linguagem é que todos os alfabetos são finitos;
- O tamanho da cadeia é definido pelo número de símbolos que ela contém, sendo incluídas as ocorrências múltiplas que ela contém.

O comportamento de um SED pode ser descrito através de um par de linguagens, considerando um alfabeto Σ como o conjunto de eventos que afetam o sistema. A evolução sequencial de um SED, ou seu comportamento lógico, pode ser então modelado através da dupla apresentada na expressão 2.1:

$$D = (L, L_m) \tag{2.1}$$

onde, L é a linguagem que descreve o comportamento gerado pelo sistema que representa o conjunto de todas as cadeias de eventos fisicamente possíveis de ocorrerem no sistema e L_m, a linguagem que descreve o comportamento marcado do sistema, ou seja, o conjunto de cadeias em L que correspondam a tarefas completadas que o sistema pode realizar (Cury, 2001).

2.1.2 Autômatos

Um autômato é um modelo que é capaz de representar uma linguagem de acordo com regras bem definidas através de uma representação gráfica, ou seja, através de um diagrama de transição de estados. Pode ser chamado também de gerador ou máquina de estados (Cassandras & Lafortune, 2008).

O autômato utilizado para a geração da linguagem representando o conjunto de possíveis comportamentos de um SED é representado por uma quíntupla, apresentada na equação 2.5 (Hopcroft *et al*., 2002; Cassandras & Lafortune, 2008):

$$G = (X, \Sigma, \delta, x_0, X_m) \tag{2.5}$$

$$\delta: X \times \Sigma \rightarrow X \tag{2.6}$$

onde,

- X é o conjunto finito de estados;
- Σ é o conjunto de eventos;

- δ é a função de transição de estados parcial, definida pela expressão 2.6 se este autômato for determinístico;
- $x_0 \in X$ é o estado inicial;
- $X_m \subseteq X$ é o conjunto de estados marcados que determinam o sucesso do término do processo.

A função de transição de estados definida pela expressão 2.6 determina que, sendo o estado atual pertencente ao conjunto de estados do autômato (X), um evento do conjunto de eventos do autômato (Σ) e caso este evento possa ocorrer, o autômato evoluirá para um próximo estado pertencente ao conjunto de estados do autômato.

Um autômato finito determinístico é aquele que se encontra em um único estado depois de ler uma sequência qualquer de entradas (eventos), ou seja, o termo determinístico se refere ao fato de que, para uma determinada entrada (sequência de eventos), existe um e somente um estado ao qual o autômato pode transitar a partir de seu estado atual (Hopcroft *et al*., 2002). Um autômato pode ser representado por meio de grafos direcionados, onde as arestas representam transições, e os vértices, estados. O estado inicial neste tipo de grafo é indicado por uma seta e os estados marcados, por círculos concêntricos, como mostrado na Figura 2.2 (Alves, 2016).

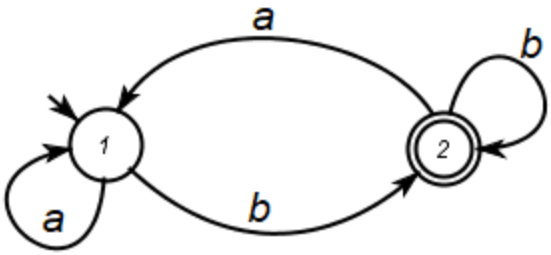

Figura 2.2 - Exemplo de autômato. Adaptado de Cury (2001).

Segundo Wonham e Ramadge (1989), um autômato G (ou gerador) é interpretado como um dispositivo que se inicia no estado x_0 e executa as transições de estado, gerando assim uma sequência de eventos. Conforme mencionado anteriormente, as transições de estado ocorrem de maneira espontânea, assíncrona e instantânea.

Um gerador G está associado a duas linguagens: a linguagem gerada $L(G)$ (equação 2.7) e a linguagem marcada $L_m(G)$ (equação 2.8). Conforme comentado anteriormente, $L(G)$ representa todas as cadeias possíveis no autômato, iniciando no estado inicial x_0 e $L_m(G)$ representa

todas as cadeias que partindo do estado inicial, chegam a um estado marcado, e são formalmente definidas, respectivamente, pelas seguintes expressões:

$$L(G) \coloneqq \{ s \in \Sigma^* : \hat{\delta}(x_0, s) \text{ é } definida\} \quad (2.7)$$

$$L_m(G) \coloneqq \{s \in L(G) : \hat{\delta}(x_0, s) \in X_m\} \quad (2.8)$$

Dois autômatos possuem linguagens equivalentes se eles geram e marcam as mesmas linguagens (Cassandras & Lafortune, 2008). Quando o sistema possui um número de estados infinitos é impossível encontrar uma representação baseada em autômatos, ainda que exista uma expressão regular que represente o sistema. O Teorema de Kleene relaciona o conjunto de linguagens regulares e o conjunto de autômatos de estados finitos, formalmente definido como:

Teorema 1 *(Teorema de Kleene)*. Se *L* é regular, existe um autômato *G* com número finito de estados tal que $\overline{Lm(G)} = L$. Se *G* tem um número finito de estados, então $L_m(G)$ é regular.

O conceito de bloqueio em autômatos se relaciona com a linguagem gerada *L(G)* e a linguagem marcada $L_m(G)$. É natural inferir que $L_m(G) \subseteq L(G)$, ou seja, a linguagem marcada está contida dentre todas as linguagens geradas pelo autômato. Conclui-se por isso, que $\overline{L_m(G)} \subseteq L(G)$, ou seja, que todo prefixo de L_m é um elemento de *L* (Cury, 2001). Quando se exige igualdade na última expressão, isto é, que cada cadeia em *L(G)* seja um prefixo de uma cadeia em $L_m(G)$, *G* é dito não-bloqueante (Ramadge & Wonham, 1989), ou seja, todas as subcadeias da linguagem são iguais as todas as subcadeias que levam o sistema a estados marcados. Assim, um SED com o comportamento descrito por *L(G)* e $L_m(G)$ é dito não bloqueante se atende a seguinte expressão (Cury, 2001):

$$L(G) = \overline{L_m(G)} \quad (2.9)$$

Porém, quando a expressão 2.9 não é atendida, ocorre a condição de bloqueio. Um autômato bloqueante é um autômato que possui estados a partir dos quais não se consegue avançar até estados marcados. Existem dois tipos de bloqueio: o *deadlock* e o *livelock*. O *deadlock* é caracterizado quando nenhum futuro evento pode ser executado e o autômato não esteja em um estado final. Por sua vez, o *livelock* ocorre quando dois estados não marcados formam um componente fortemente conectados, no qual nenhuma transição ocorre para fora desse componente. A Figura 2.3 apresenta um exemplo de um autômato que possui essas duas condições de bloqueio: a condição de bloqueio *deadlock* é a condição que leva o autômato a um estado (estado 5) em que não há possibilidade de saída (chegar a um estado marcado – o estado 2) e nem de

avanço a outros estados; e a condição de bloqueio *livelock* é a condição de bloqueio em que leva o autômato a estados não-marcados em que não se consegue alcançar estados marcados, porém consegue-se alcançar outros estados (estados 3 e 4).

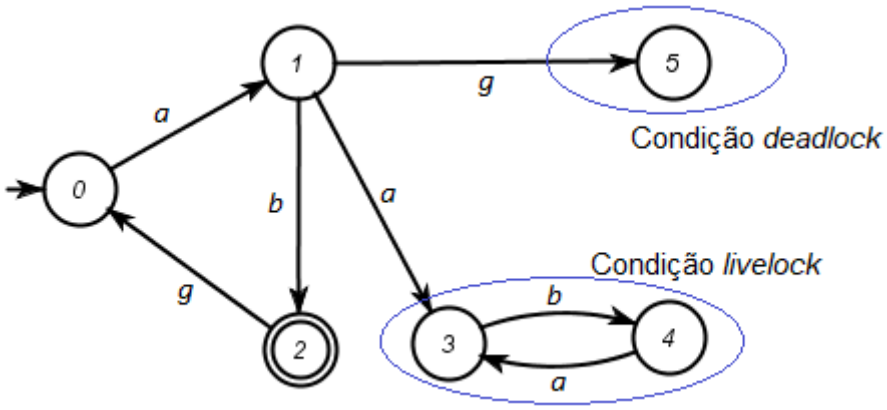

Figura 2.3 - Autômato bloqueante. Adaptado de Cury (2001).

Um autômato de estados finitos pode ter estados inacessíveis, ou seja, estados que jamais podem ser alcançados a partir do estado inicial (Cury, 2001). Um autômato é dito acessível se todos seus estados forem acessíveis (Costa, 2012). Ao aplicar a operação de parte acessível em um autômato, os estados que não podem ser alcançados a partir do estado inicial são removidos, não modificando a linguagem gerada nem a linguagem marcada do autômato, uma vez que não existe um caminho que, partindo do estado inicial leve aos estados removidos (Alves, 2016).

Um estado $x \in X$ é dito coacessível caso exista pelo menos uma *string* s que conduza o estado x até um estado marcado. Um autômato G é dito coacessível se todos os seus estados são coacessíveis (Costa, 2012; Braga, 2006; Cury, 2001). A parte coacessível de um autômato é similar ao autômato original, porém todos os estados que não permitem alcançar um estado marcado são removidos. Outro ponto importante, a aplicação dessa operação modifica a linguagem gerada do autômato, mas não modifica a linguagem marcada (Alves, 2016). A operação, ou o operador, *trim* em um autômato é o equivalente ao resultado da aplicação das operações da parte acessível e coacessível (Alves, 2016; Cury, 2001).

Existem também operações de composição de múltiplos autômatos que permitem, assim, a representação do comportamento conjunto dos autômatos originais. Dentre elas, as principais operações são: produto (composição completa paralela) e composição paralela (produto síncrono) (Alves, 2016).

A operação de produto de dois autômatos pode ser chamada também de composição completamente síncrona. Alves (2016) apresenta o produto de dois autômatos $G_1 = (X_1, \Sigma_1, \delta_1, x_{01}, X_{m1})$ e $G_2 = (X_2, \Sigma_2, \delta_2, x_{02}, X_{m2})$ como sendo $G_{12} = (X_1 \cap X_2, \Sigma_1 \cap \Sigma_2, \delta_{1x2}, x_{01} \cap x_{02}, X_{m1} \cap X_{m2})$, onde:

$$\delta_{1x2}((x_1, x_2, e) = \left\{ \begin{array}{c} \left(\delta_1(x_1, e), \delta_2(x_2, e)\right) \; se \; \delta_1(x_1, e) \; e \; \delta_2(x_2, e) \; estão \; definidos \\ indefinido \; caso \; contrário \end{array} \right\} \quad (2.10)$$

No produto, as transições dos dois autômatos devem sempre ser sincronizadas em um único evento em comum, ou seja, em um evento presente em Σ_1 e Σ_2. Por isso, o produto é considerado comutativo e associativo. O produto síncrono é utilizado na modelagem do comportamento de um SED composto, onde se deseja obter o comportamento conjunto de processos concorrentes (Sivolella, 2005). A operação de composição paralela é uma operação de caráter restritivo, já que essa operação só permitirá as transições em eventos que são comuns entre os dois autômatos.

Na modelagem de sistemas que possuem diversos componentes que interagem entre si, os eventos que são exclusivos de cada componente acabam sendo incluídos no conjunto total de eventos. A composição paralela ou produto síncrono de dois autômatos G_1 e G_2 é dado pelas expressões (Cassandras & Lafortune, 2008):

$$G_1 \,||\, G_2 := Ac(X_1 \times X_2, E_1 \cup E_2, \delta, \delta_{1||2}, (x_{01}, x_{02}), X_{m1} \times X_{m2}) \quad (2.11)$$

onde,

$$\delta_{1||2}\big((x_1, x_2), e\big) = \left\{ \begin{array}{c} \left(\delta_1(x_1, e), \delta_2(x_2, e)\right) \; se \; \delta_1(x_1, e) \; e \; \delta_2(x_2, e) \; estão \; definidos \\ (\delta_1((x_1, e), x_2) \qquad se \; \delta(x_1, e) \; está \; definido \; e \; e \notin \Sigma_2 \\ (x_1, \delta_2(x_2, e)) \qquad se \; \delta_2(x_2, e) \; está \; definido \; e \; e \notin \Sigma_1 \\ indefinido \; caso \; contrário \end{array} \right\} \quad (2.12)$$

De acordo com o formalismo de Ramadge e Wonham (1987), um Sistema a Eventos Discretos é modelado por meio de um autômato (Wonham, 2006; Costa, 2012). Na próxima seção desenvolve-se uma introdução à Teoria de Controle Supervisório (TCS) para SEDs seguindo o formalismo de Ramadge e Wonham (1989).

2.2 Teoria de Controle Supervisório

Um sistema a ser controlado corresponde a um conjunto de componentes dispostos a fim de cumprir determinado objetivo. Os componentes possuem isoladamente um comportamento

próprio e independente. Quando atuam em conjunto, devem ser controlados de forma a cumprir coordenadamente determinadas tarefas correspondentes para o sistema global (Sivolella, 2005).

A Teoria de Controle Supervisório (TCS) para SEDs possui algumas terminologias semelhantes àquelas utilizadas na Teoria de Controle clássica de sistemas contínuos e esta semelhança se explica no uso dos termos planta, supervisor e sistema em malha fechada (Braga, 2006). Uma planta é um sistema a ser controlado e que geralmente corresponde a um conjunto de subsistemas (equipamentos) que são arranjados segundo uma distribuição espacial determinada (Cury, 2001). A planta pode ser vista como um sistema de geração de eventos que possui entradas de controle que podem desabilitar eventos em determinados estados (Braga, 2006).

A TCS de Ramadge e Wonham (TCS-RW) propõe um desenvolvimento de modelos formais de sistemas de controle baseado na teoria de autômatos e linguagens, a qual é dotada de procedimentos de síntese automática de supervisores que atendem de forma menos restritiva possível as especificações de controle (Ramadge & Wonham, 1987; Constain, 2011). Esta abordagem permite encontrar um agente de controle, denominado de supervisor, capaz de desabilitar certos eventos a fim de garantir um comportamento seguro e não-bloqueante em malha fechada (Alves, 2016). A TCS possui como vantagem permitir a síntese ou obtenção automática de supervisores contrapondo ao método comum de tentativa e erro e possui a noção sobre a máxima linguagem controlável que garante a síntese de controladores que são minimamente restritivos (Queiroz, 2000).

Na TCS, o comportamento livre de um SED, ou planta, é representado pelo autômato *G* que possui um comportamento gerado *L(G)* e marcado $L_m(G)$. É possível que existam cadeias indesejáveis de eventos que levam o sistema que é representado por G a evoluir a situações de bloqueios ou que violem a sua segurança. Uma especificação ou restrição para a planta G pode ser definida como uma condição que impede que o sistema alcance situações proibidas ou indesejáveis. Especificações são também representadas por autômatos. Desse modo, as especificações impõem restrições à planta, definindo, desse modo, uma sequência de eventos que representa um comportamento desejado em malha fechada (Sivolella, 2005).

Portanto, para que sequências indesejadas de eventos não sejam geradas na planta, introduz-se um agente de controle. Na TCS, o agente de controle é chamado de supervisor e é denotado pela letra *S* (Ramadge & Wonham, 1989). O supervisor é um agente de controle externo

que possui habilidade de observar os eventos gerados pela planta e influenciar seu comportamento através da desabilitação ou proibição da ocorrência de eventos, sendo essa sua entrada de controle (Alves, 2006).

A entrada de controle é definida como o conjunto de eventos habilitados num dado instante pelo supervisor *S* e essa entrada é atualizada sempre que a ocorrência de um evento é observada na planta *G*. Um supervisor pode ser representado por um autômato (*S*), definido sobre o mesmo alfabeto da planta *G*, cuja função consiste em habilitar e desabilitar eventos (Cury, 2001). A construção de um supervisor é realizada através de um processo chamado de síntese, que constitui na composição assíncrona dos autômatos construídos para modelar as especificações.

O funcionamento do sistema controlado ou sob a ação do supervisor, denotado por S/G, pode ser descrito pelo SED resultante da composição síncrona de *S* e *G*, no caso, *S*||*G*. Assim sendo, na composição síncrona *S*||*G*, apenas as transições permitidas tanto no sistema controlado (planta *G*), como no supervisor *S* são permitidas (Cury, 2001). O supervisor, então, garante que o funcionamento da planta esteja de acordo com uma dada especificação, apenas desabilitando os eventos não permitidos no determinado instante, de acordo com a evolução de eventos ocorridos no sistema. O papel do supervisor é, então, habilitar ou desabilitar eventos de modo que o comportamento final seja aquele representado por *S/G*.

O supervisor atua na planta numa estrutura em malha fechada, como apresentada na Figura 2.4, atendendo as especificações pré-determinadas. Por este motivo, o supervisor é considerado de natureza permissiva e minimamente restritivo, uma vez que apenas impede o acontecimento de sequências indesejáveis. O supervisor garante, também, a atuação correta e coordenada de todos os subsistemas que compõem a planta. Associando-se as definições apresentadas ao ambiente de um SED, tem-se que um alfabeto Σ corresponde ao conjunto de eventos que podem ocorrer neste SED, visando a evolução do mesmo (Costa, 2012). O conjunto de eventos Σ é particionado em dois subconjuntos: o sub-conjunto de eventos controláveis Σ_c e o subconjunto de eventos não-controláveis Σ_u, determinado pela expressão 2.13:

Os eventos em Σ_c podem ser desabilitados por um agente externo, como por exemplo, um supervisor em qualquer momento do tempo, enquanto que os eventos em Σ_u modelam eventos não são possíveis de serem desabilitados (Ramadge & Wonham, 1989). Pode-se citar como que eventos não-controláveis o quebrar de uma máquina, a identificação de erros em peças através de um sensor, a perda de um canal de comunicação, entre outros. Portanto, a ação do supervisor

sobre a planta consiste em inibir a ocorrência de eventos controláveis de forma a alcançar o comportamento desejado (Costa, 2012). O supervisor é calculado para que o sistema em malha fechada obedeça a máxima linguagem controlável (todo o conjunto de linguagens que são possíveis pela TCS) contida na especificação.

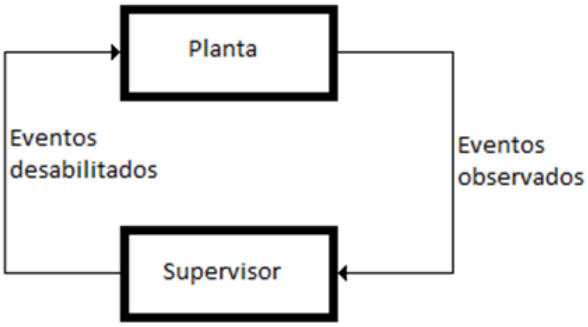

Figura 2.4 - Estrutura planta e supervisor. Adaptado de Cury (2001).

$$\Sigma = \Sigma_c \cup \Sigma_u \tag{2.13}$$

Como Cassandras & Lafortune (2008) apresentam, as linguagens que são permitidas pelo controle são conhecidas como "linguagens legais ou admissíveis". Um sistema que fica dentro da linguagem legal é chamado de sistema com comportamento seguro e obedece a relação 2.14, onde L_r é definido como mínimo comportamento requerido e L_a como o máximo comportamento admissível:

$$L_r \subset L_a \subset L(G) \tag{2.14}$$

Seja uma planta G, modelada através de um autômato determinístico de estados finitos, e que possui um comportamento dado pelo par das linguagens $L(G)$ e $L_m(G)$) definidas sobre um conjunto de eventos Σ, determina-se uma estrutura de controle Γ para G, pelo particionamento de Σ em Σ_c e Σ_u. Sendo $\gamma \in \Gamma$, uma entrada de controle que aplicada ao sistema que contém um conjunto de eventos que estão habilitados a ocorrer no sistema. Esse conjunto definido também contém o subconjunto de eventos não-controláveis, como é apresentado na definição 2.15 (Cury, 2001):

$$\Gamma=\{\gamma \epsilon 2^{\Sigma} : \gamma \supseteq \Sigma_u\} \tag{2.15}$$

onde a condição $\gamma \supseteq \Sigma_u$ indica apenas que os eventos não-controláveis não podem ser desabilitados pelo supervisor.

Assim, formalmente, um supervisor *S* é uma função definida pela expressão 2.16, ou seja, o supervisor controla a função de transição de *G*, no sentido que os eventos controláveis podem ser habilitados e desabilitados, onde se associa um conjunto de eventos habilitados a cada cadeia de eventos observadas na planta. Dessa forma, *G* não consegue executar um evento que não está no conjunto atual de eventos ativos se esse evento também não está também contido em *S* (Cassandras & Lafortune, 2008). Isto é, um evento controlável pode ser impedido de ocorrer, uma vez que fisicamente não seja possível ou está desabilitado pelo supervisor.

$$S: L(G) \rightarrow \Gamma \tag{2.16}$$

O sistema controlado, composto pela planta *G* sob ação do supervisor *S* e representado por *S/G*, é também um sistema a eventos discretos e pode ser caracterizado pela sua linguagem gerada e sua linguagem marcada. A linguagem gerada *L(S/G)* inclui a cadeia vazia ($\varepsilon \in L(S/G)$) e todos os eventos que estão presentes nas linguagens geradas do supervisor e da planta (expressão 2.17):

$$[(s \in L(S/G))\ e\ (s\sigma \in L(G))\ e\ (\sigma \in S(s))] \leftrightarrow [s\sigma \in L(S/G)] \tag{2.17}$$

A linguagem marcada $L_m(S/G)$ do sistema *S/G* é definida em 2.18, onde a linguagem marcada do supervisor sobre a planta ($L_m(S/G)$) é a intersecção da linguagem gerada do supervisor sobre a planta ($L(S/G)$) e a linguagem marcada da planta ($L_m(G)$).

$$L_m(S/G) := L(S/G) \cap L_m(G) \tag{2.18}$$

Algumas conclusões podem ser tiradas das definições 2.16, 2.17 e 2.18:

- A cadeia vazia ($\mathcal{E}$) sempre está em *L(S/G)* desde que ela está sempre contida em *L(G)*;
- As propriedades de bloqueio do *S/G* são resultantes da combinação de *S* controlando *G*. O supervisor S é dito bloqueante se *S/G* é bloqueante, e *S* não é bloqueante, caso S/G não o seja.
- O controle nos SEDs utilizando a TCS é dito monolítico quando apenas existe um supervisor para restringir o comportamento global de uma planta (Braga, 2006; Constain, 2011). Quando o sistema inclui um grande número de tarefas, a abordagem monolítica pode exigir um grande esforço computacional já que a composição síncrona acaba gerando um crescimento exponencial em número de estados. Na sequência é apresentada a abordagem modular da TCS, em particular o Controle Supervisório Modular Local (CSML), em comparação com o controle monolítico.

2.2.1 Controle Supervisório Modular e Modular Local

A primeira abordagem proposta para o Controle Supervisório de um SED foi o Controle Supervisório Monolítico (Ramadge e Wonham, 1989; Costa, 2012). O objetivo do controle supervisório monolítico é projetar um único controlador cuja função é habilitar e desabilitar eventos controláveis de acordo com a ocorrência de eventos na planta, em uma estrutura em malha fechada obedecendo algumas regras operacionais especificadas (Queiroz, 2004). A Figura 2.5 apresenta o funcionamento de um supervisor monolítico.

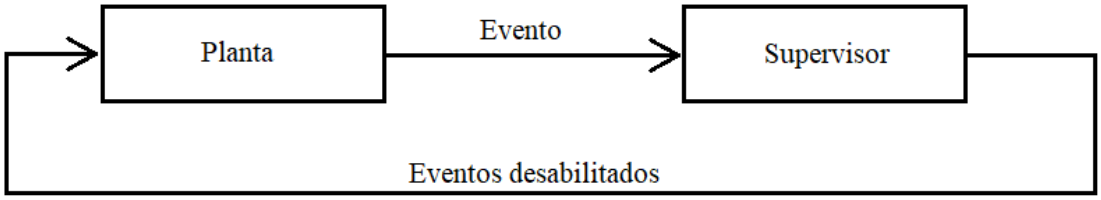

Figura 2.5 - Estrutura Planta e Supervisor Monolítico. Adaptado de Cury (2001).

A metodologia do controle supervisório monolítico sofre de alguns problemas, dentre eles, o grande número de estados da planta e/ou do supervisor, o que leva a um elevado consumo de memória no procedimento de síntese, o que inviabiliza a obtenção do supervisor quando considerados sistemas de médio a grande porte (Costa, 2012), problema conhecido como explosão de estados. Também, como apresenta Constain (2014), em plataformas com pouco poder de processamento, como Controladores Lógicos Programáveis (CLPs), a implementação de supervisores com muitos estados e transições pode ser inviável. Por estes motivos, a Teoria de Controle de Supervisório tem tão poucas aplicações práticas na indústria (Costa, 2012; Sivolella, 2005).

Diversas abordagens surgiram para tentar contornar o problema da explosão de estados, como a abordagem do Controle Supervisório Modular (CSM) (Ramadge & Wonham, 1988), do Controle Supervisório Modular Local (CSML) (Queiroz & Cury, 2000), do Controle Hierárquico (Cunha, 2003; Schimdt, Moor & Perk, 2008).

A abordagem do Controle Supervisório Modular (CSM), proposta por Ramagde & Wonham (1988) propõe a construção de um supervisor modular para cada especificação definida e esses supervisores atuam em conjunto para satisfazer a especificação global do sistema (Costa, 2012). A construção desses supervisores se dá através da composição síncrona entre a planta global e cada especificação. A Figura 2.6 mostra o funcionamento da abordagem modular local.

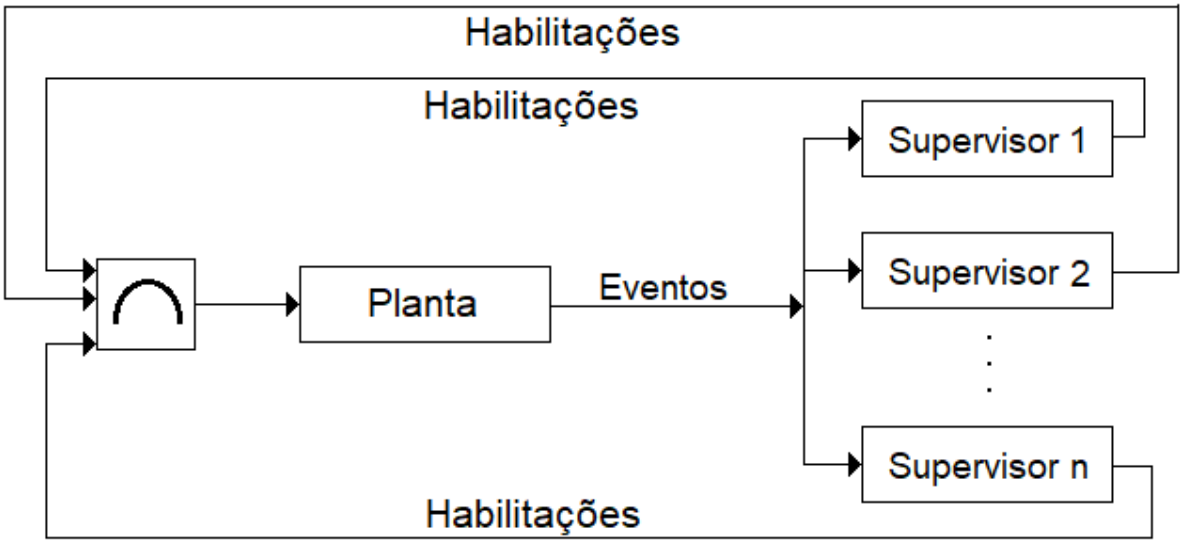

Figura 2.6 - Esquema do CSM. Fonte: Costa (2012).

A abordagem do Controle Supervisório Modular Local (CSML) (Queiroz & Cury, 2000) estende a TCS ao incluir a modularidade das especificações, como na abordagem local, mas também da planta. Nesta abordagem, cada uma das especificações do sistema possui um supervisor modular controlando a planta local ao qual é relacionado (Yuri *et al.*, 2010). A planta local diz respeito somente aos subsistemas relacionados com a especificação, dispensando a obtenção de um modelo completo da planta. Nesta abordagem, ao invés do projeto de um único supervisor monolítico que satisfaça todas as especificações, procura-se construir um supervisor para cada uma das especificações de maneira que a atuação conjunta dos supervisores atenda as especificações globais (Queiroz, 2004). A construção dos supervisores nesta abordagem se deve à composição síncrona da especificação com a planta local que se relaciona à especificação local. Assim, constrói-se a planta local através da composição assíncrona dos subsistemas que se relacionam à especificação e então, através da composição síncrona entre a planta local construída e da especificação, tem-se o supervisor local. Desse modo, evita-se a obtenção de um modelo global de supervisor (abordagem monolítica) e/ou modelo global para a planta (abordagens monolítica e modular).

Assim, a CSML permite utilizar da propriedade da modularidade das especificações e da modularidade da planta, diminuindo mais ainda a complexidade computacional da síntese de supervisores e o tamanho das soluções (Constain, 2014). A Figura 2.7 mostra o funcionamento da abordagem modular local. Os controladores modulares têm suas ações baseadas em uma versão parcial do funcionamento da planta global e, por isso, pode gerar casos de conflitos na ação de controle (Queiroz, 2000). Portanto, se faz necessária a realização do teste de não-con-

flito (modularidade local), para verificar a existência de conflito entre os supervisores modulares locais.

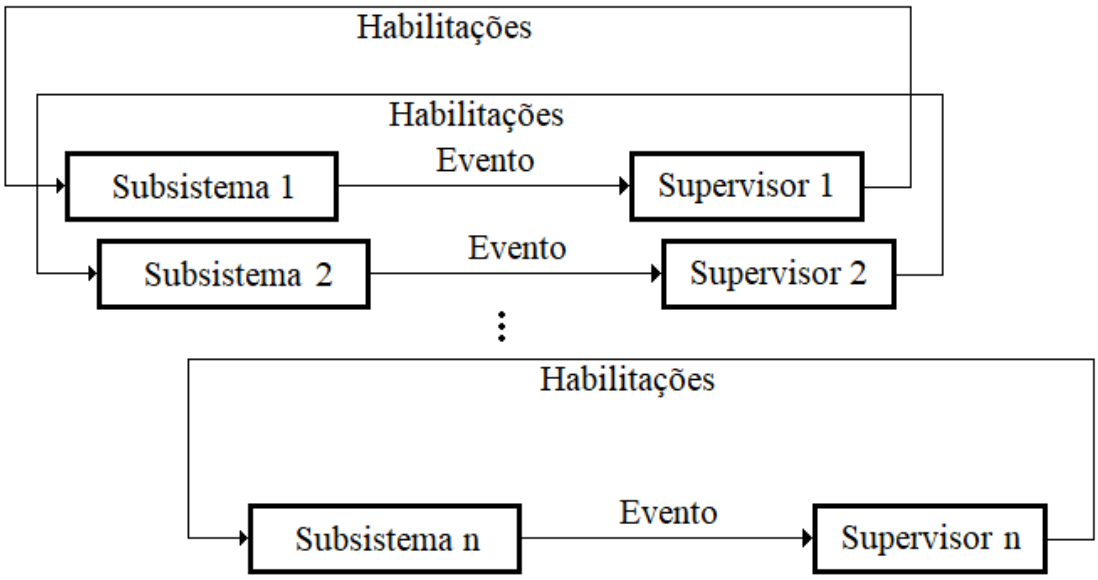

Figura 2.7 - Esquema do CSML. Fonte: Queiroz (2004).

2.2.2 Propriedades e conceitos na Teoria de Controle Supervisório

Levando em consideração que o controle exercido pelo supervisor é restritivo, no projeto e síntese dos supervisores deve-se buscar limitar a planta da menor forma possível, obtendo-se assim ação minimamente restritiva ou ótima dos supervisores sobre a planta. Um supervisor ao ser gerado pelo processo da síntese, atende alguns critérios como, por exemplo, não leva a planta a situações de bloqueio e atende o Teorema da Controlabilidade (Ramadge & Wonham, 1989). Esses critérios são apresentados como se segue.

- **Controlabilidade**

Uma linguagem $L \subseteq \Sigma^*$ é controlável (em relação a planta G) se $\overline{L} = \Sigma_u \cap \overline{L}$, ou seja, a ocorrência de um evento não controlável na planta G não gera uma sequência de eventos que desrespeite as especificações dadas (Ramadge & Wonham, 1989).

A controlabilidade é a condição necessária e suficiente para a existência do supervisor não-bloqueante que atenda uma determinada especificação desejada (Ramadge & Wonham, 1988).

- **Teorema de Ramadge e Wonham**

Dada uma linguagem $L \subseteq L_m(G)$, $L \neq 0$, então existe um supervisor não-bloqueante S tal que $L_m(S/G) = L$ se e somente se L for controlável em relação a G e L for a linguagem marcada fechada de G.

- **Não-conflito**

Quando existem diversos supervisores sobre uma planta, há a possibilidade dos supervisores implementarem comportamentos contraditórios, já que os supervisores não possuem informações a respeito das especificações implementadas pelos outros (Pena, 2007). Os supervisores podem não ser bloqueantes em relação a sua planta, porém a ação conjunta pode ser conflitante.

Na abordagem do CSML, a ação de cada supervisor é aplicado somente sobre o respectivo subsistema. Tal ação, devido ao compartilhamento de eventos, pode ter interferência nas demais plantas locais, o que pode causar bloqueio quando mais de um supervisor é aplicado simultaneamente sobre a planta (Pena *et al.*, 2010).

Uma maneira de verificar o conflito é aplicar o teste de não-conflito, também chamado teste de modularidade. Pena (2007) apresenta o teste de não-conflito descrito a seguir:

Sejam S_j linguagens que implementam a ação dos supervisores e que também são chamados de S_j, onde $j \in J = \{1, ..., m\}$. O teste baseia-se em averiguar se a igualdade mostrada na expressão 2.19 é atendida. Em resumo, o teste consiste em verificar que, para todo o prefixo "compartilhado" por duas ou mais linguagens, exista pelo menos uma cadeia "compartilhada" que o contenha.

$$\overline{\left\|_{j=1}^{m} S_j\right.} = \left\|_{j=1}^{m} \overline{S_j}\right. \tag{2.19}$$

2.2.3 Modelagem e Síntese de Supervisores

Uma metodologia básica para a síntese de supervisor ótimo baseado na abordagem CSML, baseia-se na modelagem da planta, definição das especificações e síntese do supervisor, como apresentado em (Cury, 2001) e em (Queiroz, 2004). Na sequência apresentam-se os três passos adaptados ao contexto deste trabalho.

a) Passo 1: Modelagem da Planta

Utilizando a abordagem CSML, que permite a decomposição da planta em subsistemas (plantas locais), é realizada a modelagem da planta seguindo os sequentes passos:

- Determinar os subsistemas envolvidos na planta;
- Definir o alfabeto de eventos, classificando os eventos em controláveis e não-controláveis.
- Construir um modelo G_{locx} para cada subsistema (planta local);

b) Passo 2: Modelagem das Especificações

Após a classificação dos eventos em controláveis e não-controláveis, definem-se as condições para que o sistema funcione de maneira segura e desejável, isto é, definir as especificações do sistema de controle. De forma semelhante, modelando cada uma das especificações separadamente, deve-se seguir as seguintes etapas:

- Identificar as restrições/especificações e construir os respectivos autômatos E_i;
- Realizar o teste de controlabilidade de cada uma das especificações E_i;
- Realizar o teste de modularidade ou não-conflito das especificações E_i;
- Para cada especificação E_i, realizar a composição dos autômatos da especificação com o autômato G_{locx} da planta local correspondente, obtendo a linguagem alvo $K_{locx,}$ que é a máxima linguagem ou seja, a linguagem alvo que o supervisor deseja seguir (produto síncrono da especificação com a planta);
- Atualizar K_{locx} pelo cálculo de sua componente coacessível ou não-bloqueante;

c) Passo 3: Síntese do Supervisor não bloqueante ótimo

O processo de síntese dos supervisores, desde que garantido o teste de não-conflito, além da controlabilidade das especificações, garante a obtenção de supervisores não-conflitantes e minimamente restritivos (Queiroz, 2000). Ou seja, ao garantir o funcionamento do sistema de maneira desejável através da definição e modelagem das especificações através de autômatos, atendendo tanto as condições de não-bloqueio e de controlabilidade, consegue-se que os supervisores possuam todas linguagens possíveis e legais - a máxima linguagem controlável - garantindo que a ação conjunta dos supervisores modulares seja ótima (minimamente restritiva).

Uma condição de existência do supervisor não-bloqueante para a planta consiste na composição assíncrona de todas as especificações (denotado de *E*) pertencer a linguagem marcada da planta, então existe este supervisor.

Porém, nem sempre *E* é controlável, sendo necessário, portanto, que se calcule uma linguagem que mais se aproxime de *E*, chamada de *supC(E)*, isto é, uma linguagem suprema que seja a mais adequada e ótima (máxima linguagem controlável). Este cálculo se baseia em encontrar os "maus estados" e os eliminar do autômato (Queiroz, 2004).

2.3 Representação temporal na TCS

A Teoria de Controle Supervisório que se baseia em máquinas de estados finitos (MEF), tem sido amplamente estudada, abordando questões fundamentais sobre controle de SEDs. Segundo Chen e Lin (2000), no que se diz respeito ao controle de um SED atualmente, muitas das questões fundamentais já foram resolvidas e por isso, deve-se prestar atenção aos modelos que sejam mais úteis nas aplicações reais.

Na abordagem clássica da TCS, o tempo não é considerado, apesar do tempo ser inerente em sistemas reais (Pena *et al*., 2015). O tempo introduz uma nova dimensão de controle e modelagem de SEDs, de considerável interesse de aplicação e também de significante complexidade. Existem diversos trabalhos que procuram inserir o tempo em SEDs, como Moller (1986), Kozak (1991), Brave & Heymann (1988) e Wong-Toi & Hoffmann (1991), porém a construção do modelo não parece ser passível de composição. Segundo Brandin & Wonham (1994), na área geral de métodos formais para as especificações, projeto e verificações de sistemas que dependem do tempo, existe um substancial corpo de trabalhos publicados, porém são relativamente pouco direcionados para a síntese formal de controle que sejam ótimos.

Para lidar com esse problema do tempo, foram desenvolvidos modelos para sistema a eventos discretos que levam o tempo em consideração e, entre estes trabalhos, alguns também desenvolveram métodos para síntese de estruturas de controle para este tipo de sistema (Brandin & Wonham, 1994; Penha *et al.*, 2011). O modelo proposto por Brandin & Wonham (1994) é adjacente a estrutura da TCS e é capaz de capturar os problemas de tempo em uma útil gama de problemas de controle.

Como apresenta Brandin & Wonham (1994), para a implementação do modelo desenvolvido deve-se iniciar pela quíntupla apresentada na expressão 2.20:

$$G_{act} \coloneqq (\Sigma_{act}, A, \delta_{act}, a_0, A_m) \tag{2.20}$$

onde,

- Σ_{act} é um alfabeto finito de eventos;
- *A* é o conjunto de atividades onde seus elementos são atividades. Atividades se diferem de eventos por possuírem uma duração (tempo) a elas associadas, enquanto eventos são considerados instantâneos. Substituiu o conjunto *Q* (conjunto de estados). Este conjunto é obrigatoriamente finito;
- δ_{act} é a função de transição de atividades, que é parcial, e apresenta-se na expressão 2.21 que define que na ocorrência de um evento, leva o sistema a uma nova atividade. Também se considera que uma transição de atividade é uma tripla apresentada na expressão 2.22, onde *a* é atividade, α é o evento relacionado à atividade e *a'* é a nova atividade definida pela expressão 2.23.
- a_0 é a atividade inicial;
- A_m é o subconjunto de atividades marcadas.

$$\delta_{act}: \Sigma_{act} \times A \rightarrow A \tag{2.21}$$

$$[a, \alpha, a'] \tag{2.22}$$

$$a' = \delta_{act}(\alpha, a) \tag{2.23}$$

Brandin &Wonham (1994) introduzem um evento adicional chamado de "*tick"* que representa o tique-taque do relógio global. As expressões temporais sempre serão especificadas em termos do tempo do relógio digital global. Sendo assim, o evento *tick* sempre ocorre exatamente em momentos de tempo real. Dessa forma, o conjunto total de eventos do autômato passa a ser representado pela expressão 2.24, ou seja, o novo conjunto de eventos do sistema é constituído pelos eventos mais o evento *tick.* A função parcial de transição de estados do autômato passa a ser 2.25, onde na ocorrência de um evento, o sistema evolui para um novo estado. O autômato está, por fim, descrito em 2.26, caracterizando assim esses sistemas como SEDs, podendo ser modelados e controlados através da TCS. Esse novo tipo de autômato que inclui o tempo modela um tipo de sistema que foi chamado por Brandin & Wonham de Sistemas a Eventos Discretos Temporizados (SEDT).

$$\Sigma \coloneqq \Sigma_{act} \cup \{tick\} \tag{2.24}$$

$$\delta : \Sigma \times X \rightarrow X \tag{2.25}$$

$$G = (X, \Sigma, \delta, x_0, X_m) \tag{2.26}$$

Dentro da proposta dos SEDT, representando os sistemas por autômatos, foram desenvolvidas duas estruturas, a primeira chama de Gráficos de Transição de Atividade (GTA) e Gráficos de Transição Temporizada (GTT) (Brandin & Wonham, 1995; Penha, 2011).

- GTA: são estruturas modeladas por autômatos no qual em cada evento σ no conjunto Σ_{act} onde adiciona-se as informações relativas no tempo. Essas informações constituem-se de um tempo de limite inferior (l_a) e um tempo de limite superior (u_a) que correspondem, respectivamente aos limites minimos e máximos de tempos do evento a quando for habilitado pelo supervisor. Na Figura 2.8 é apresentado um exemplo da estrutura GTA. Neste exemplo, tem-se um autômato com dois estados e dois eventos, o evento *a* e o evento *b*. Para o evento *a*, os limites l_a e u_a são respectivamente 1 e 2 para o evento *b*, os limites l_a e u_a são, respectivamente, 2 e infinito. Nessa estrutura, o evento *tick* é suprimido e as restrições das ordens de atividade são incorporadas pelas margens de tempo (Brandin & Wonham, 1994);

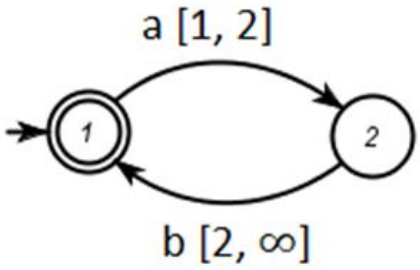

Figura 2.8 - Exemplo de um SEDT modelado através da estrutura GTA. Fonte: Brandin & Wonham (1994).

- GTT: são estruturas de transição também modeladas por autômato, porém, neste caso, o tempo (evento *tick*, geralmente representado por *t*) é explicitamente incluido na estrutura adicionando eventos que modelam o avanço de uma unidade de tempo. Na Figura 2.9 é apresentado um exemplo da estrutura GTT.

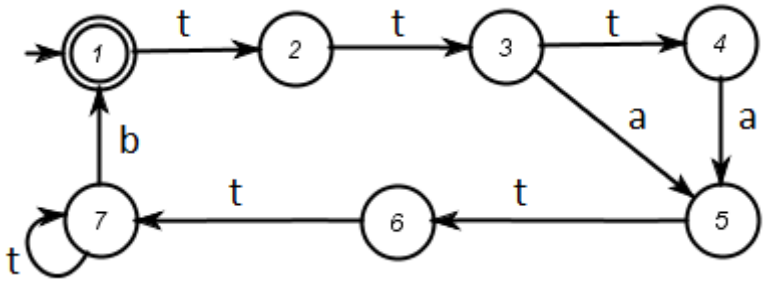

Figura 2.9 - Exemplo de um SEDT modelado através da estrutura GTT. Fonte: Brandin & Wonham (1994).

Em contrapartida ao trabalho apresentado por Brandin e Wonham (1994) que propõe a utilização de SEDT, na inserção do tempo em SEDs, Chen e Lin (2000) propõe a modelagem usando máquinas de estados finitos com parâmetros.

Ao modelar em máquinas de estados finitos, o problema pode se tornar exponencialmente complexo devido à explosão de estados, tornando a modelagem inadequada para aplicações práticas onde o número de estados e eventos é relativamente grande. Porém, ao inserir um parâmetro para descrever algum contexto, permite-se reduzir drasticamente o número de estados (Chen & Lin, 2000). Por exemplo, em um sistema que contém um *buffer* (armazém) que possui 10 partições, onde cada partição pode estar ou não preenchida, apenas para esse *buffer* teríamos 1024 estados. Porém, ao empregarmos, por exemplo, numerais de 0 a 1, consegue-se descrever o *buffer* da mesma maneira, reduzindo assim, drasticamente o número de estados na modelagem desse sistema.

Por este motivo, Chen e Lin propõem empregar máquinas de estado finito juntamente com conjuntos de parâmetros para modelar SEDs. Tal modelo é denominado de Máquinas de Estados Finitos com Parâmetros (MEFcP).

Uma MEF constitui de uma quíntupla de mesma constituição da expressão 2.7. Para a introdução de parâmetros em uma MEF é necessário expandir o modelo, incluindo novos conceitos, descritos a seguir (Chen e Lin, 2000). Seja $p \in P$ um vetor de parâmetros e P um espaço vetorial e seja $g \in G$ as guardas que são condições de ocorrência de um evento e são os predicados dos parâmetros p. A função de transição δ, anteriormente definida pela expressão 2.27, passa a ser definida pela expressão 2.28, ou seja, a função de transição passa a depender dos parâmetros:

$$\delta: \Sigma \times Q \to Q \tag{2.27}$$

$$\delta: \Sigma \times Q \times G \times P \to Q \times P \tag{2.28}$$

A Figura 2.10 mostra a representação de uma transição de uma MEF com parâmetros, que passa a ser interpretada da seguinte maneira: se no estado x_1 a guarda g é verdadeira e o evento α ocorrer, então o próximo estado é x_2 e os parâmetros são atualizados de acordo com a função de atualização dos valores dos parâmetros ($f(p)$).

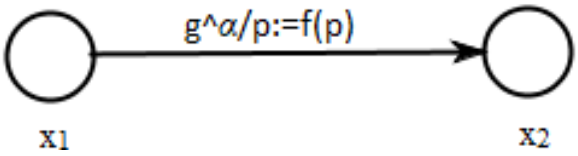

Figura 2.10 - Transição de uma MEF com parâmetro. Fonte: Chen & Lin (2000).

Um parâmetro que é comumente utilizado em sistemas é o valor global do relógio, ou seja, o segundo (s) e muitas vezes denotado por *t*. Em Sistemas Flexíveis de Manufatura (SFM) que dependem do tempo, as guardas e funções de ação são então adicionadas às transições dos autômatos de forma a modelar não apenas o comportamento lógico do sistema, mas também a duração de cada operação (Pena *et al*., 2015; Costa, 2012). Nessa abordagem, um evento não-controlável modela uma resposta a um comando modelado por um evento controlável. Um evento não-controlável continua não sendo possível de ser impedido de ocorrer. Os autômatos que representam os supervisores não são alterados e, dessa forma, a cada par de eventos (comandos e respostas) do sistema é associado a uma guarda (Costa, 2012). O modelo da guarda é apresentado na Figura 2.11, onde,

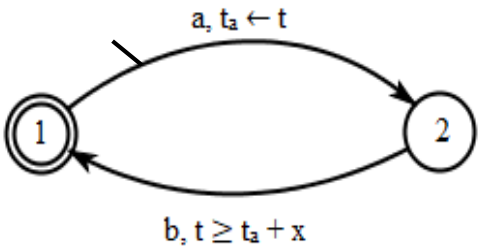

Figura 2.11- Autômato com guardas. Fonte: Costa (2012).

- *a*: evento controlável;
- *b*: evento não-controlável;
- *x*: duração, considerada fixa e conhecida. Cada par de eventos se relaciona pelas guardas;
- t_a: tempo de simulação.

Nesse tipo de representação, não existem condições temporais para eventos controláveis, isto é, se um evento controlável é logicamente factível e permitido pelo supervisor, não existe condição temporal que adiará sua execução (Pena, 2015). Também, se existir concorrência entre um evento controlável e um não-controlável em um instante de tempo, o evento não-controlável tem prioridade.

Esta forma de modelagem se adequa aos sistemas de manufatura, pois são modelados com um nível de abstração no qual cada evento não-controlável indica o fim de uma operação que foi anteriormente iniciada pelo seu evento controlável correspondente.

2.4 Considerações finais

Neste capítulo foi apresentado um tipo de sistema baseado na execução de tarefas, conhecido por Sistemas à Eventos Discretos. Diversos sistemas que existem atualmente podem ser classificados como SEDs, tais como: sistemas de fila, sistemas flexíveis de manufatura, entre outros. Existem diversas abordagens propostas para a modelagem de SEDs, em particular, neste capítulo foi apresentada a abordagem baseadas em linguagem e autômatos. Como forma de inserir o controle em SEDs, foi apresentada a Teoria de Controle Supervisório (TCS) que é um método para a obtenção de estruturas de controle para sistemas a eventos discretos, em particular, a abordagem modular local, uma vez que esta aproveita a modularidade do sistema.

Por fim, foram apresentadas duas diferentes abordagens para a resolução da inserção do tempo em sistemas a eventos discretos: através da ideia proposta do Sistema à Eventos Discretos Temporizados ou da modelagem de máquinas finitas com parâmetros.

Capítulo 3

Escalonamento de Tarefas

Um sistema flexível de manufatura (SFM) é um sistema no qual um grupo de máquinas de controle numérico e um conjunto de dispositivos de manuseio de materiais trabalham juntos sob o controle de um computador central (Groover, 2011; Costa, 2012). Esses SFMs, em sua maioria altamente automatizados, buscam integrar seus diversos níveis gerenciais e operacionais, destacando-se os sistemas de gestão da produção e os sistemas de controle de chão de fábrica. Neste contexto, aparecem os sistemas de supervisão e aquisição de dados (SCADA), que monitoram e supervisionam o funcionamento correto dos equipamentos e também possibilitam a integração entre sistemas de gerenciamento de produção e o controle do chão de fábrica (Groover, 2011).

Devido ao grande avanço e rápido desenvolvimento de tecnologias em computação, SFMs tem recebido aumento de atenção nos últimos anos, pois são sistemas extremamente complexos constituindo-se de vários componentes de hardware e software que estão interconectados (Chan & Chan, 2004). Um dos problemas relacionados aos SFMs que receberam e recebem muita atenção são os problemas de escalonamento de tarefas.

Neste capítulo será apresentada uma pequena introdução sobre os níveis de automação presentes em sistemas de manufatura e como os sistemas de supervisão conseguem realizar uma parte da integração desses níveis. No quesito do nível de gerenciamento da produção, apresenta-se o clássico problema do escalonamento de tarefas que é discutido na sequência. Para a resolução deste problema, diversos pesquisadores desenvolveram e utilizaram diversas técnicas, como metodologias analíticas ou até a utilização de metaheurísticas. Para esse trabalho foi escolhido a metaheurística conhecida como Algoritmos Genéticos. Por fim, introduz-se o algoritmo genético e seus conceitos, abordando principalmente a questão de sua utilização para a resolução de problemas de escalonamento de tarefas.

3.1 Níveis em Sistemas de Manufatura Automatizados

Sistemas de manufatura automatizados visam integrar os seus diversos níveis gerenciais e operacionais (ISO-95, 2000), entre eles, os sistemas de gestão da produção e os sistemas de controle de chão de fábrica. Particularmente, os sistemas supervisórios, também conhecidos como sistemas SCADA (*Supervisory Control and Data Acquisition Systems*) são sistemas que adquirem dados em tempo real, supervisionam processos, apresentando essas informações em uma interface de fácil interpretação e utilização, apresentando ao usuário um conjunto de informações e opções de controle.

Devido a hierarquia presente em sistemas automatizados, apresentada na Figura 3.1, pode-se notar os diferentes níveis dentro de um sistema automatizado. Estes sistemas buscam integrar os seus diversos níveis gerenciais e operacionais, destacando-se para esse trabalho, os níveis de chão de fábrica (controle e dispositivos) e os níveis de gerenciamento (planta e produção).

Figura 3.1 - Níveis de automação. Adaptado: Santos (2012).

No nível de gerenciamento, o foco é na execução das atividades de produção, ou seja, as operações centrais de agregação de valor de uma empresa de manufatura e que se constituem o seu objetivo central (Lopes, 2012). Esses níveis possibilitam a otimização das atividades dentro da produção. O SCADA não é um sistema completo de controle, mas apenas realiza a supervisão e suas funções se resumem a acesso, análise de tendências, armazenamento de dados, automação de tarefas e apresentação das informações em uma interface homem-máquina (IHM) (Lopes, 2012).

Os sistemas de supervisão realizam suas funções através da troca de informações e comandos com os sistemas de controle das plantas, muitas vezes implementados em controladores industriais, entre eles os controladores lógicos programáveis (CLP) (Groover, 2011). Por esse motivo, devido a sua hierarquia, é possível a integração entre as ferramentas de gerenciamento da produção, entre elas os planos de produção contendo o escalonamento das tarefas, com os níveis operacionais de execução das atividades.

3.2 O Problema do Escalonamento de Tarefas

Os Sistemas Flexíveis de Manufatura são o resultado de dois fatores: o crescimento na demanda da quantidade do produto e a preocupação pela qualidade do mesmo e são projetados para conseguir combinar a eficiência com a flexibilidade para a produção de um lote de médio volume da melhor maneira possível (Chan & Chan, 2004). Estes sistemas são altamente automatizados e por isso são necessários investimentos iniciais consideráveis para viabilizar a sua implantação, podendo ter benefícios como aumento da produção, redução nos custos, redução nos tempos de finalização das tarefas, redução de estoques, entre outros (Costa, 2012). E para que estes benefícios sejam alcançados, torna-se necessário a utilização de alguma técnica de escalonamento da produção (ou escalonamento de tarefas) que consiste na ordenação e distribuição de tarefas para atingir desempenhos melhores. Muitos dos problemas de produção dentro de um SFM são atribuídos à função do escalonamento, muitas vezes feita por um operário no chão de fábrica.

A importância do escalonamento tem aumentado nos recentes anos devido ao crescimento na demanda do consumidor pela variedade, na diminuição do ciclo de vida dos produtos, nas mudanças de mercado ocasionadas pela competição global e no rápido desenvolvimento de novos processos e tecnologias. Neste contexto, o problema do escalonamento de tarefas tem atraído muitos pesquisadores devido à sua ampla aplicabilidade e complexidade (Ho *et al.*, 2007). Também, os problemas de escalonamento constituem uma classe muito importante dentro da otimização combinatória e tem sido uma área de pesquisa muito ativa durante diversos anos (Becerra & Coello, 2005).

O escalonamento pode ser tratado como problema de otimização, tendo como foco otimizar uma função objetivo encontrando caminhos onde tarefas atribuem recursos obedeçam determinados critérios. Ou seja, o propósito do escalonamento de produção dentro de um SFM

é alocar um conjunto de recursos limitados para tarefas ao longo do tempo. No caso particular do *job-shop scheduling*, as tarefas podem ser chamadas também de trabalhos e os recursos são as máquinas que realizam certos trabalhos. Uma tarefa ou trabalho é definida por uma sequência definida de operações ou passos que levam a um objetivo final, por exemplo, a produção de uma peça.

O problema *job-shop scheduling*, ou seja, o problema de sequenciamento e distribuição de recursos dentro de uma célula flexível de manufatura ou um SFM, abreviado frequentemente por JSP, possui uma grande importância industrial já que, ao ser solucionado, é possível obter produção livre de falhas de alto custo, com uma utilização de alta eficiência de recursos do SFM disponíveis (Silva *et al*., 2011). Assim, ao encontrar um escalonamento de produção que seja o melhor possível, consegue-se aumentar a capacidade de produção da planta ao maximizar o uso de recursos críticos de equipamentos de processamento, deslocando tarefas para equipamentos menos ocupados, quando possível, e tarefas de produção oportunamente adequadas para outras janelas de tempo não utilizadas (Shi & Pan, 2005).

Segundo Becerra & Coello (2005), algumas considerações para o entendimento do JPS são necessárias:

- Cada trabalho (tarefa) tem uma sequência lógica e, por isso, requer ser processado nas máquinas seguindo uma certa ordem, fixa, ou seja, existem restrições de precedência para cada operação;
- As máquinas não podem processar mais de um trabalho por vez;
- E por fim, uma vez que a máquina tenha começado um certo trabalho, ele não pode ser interrompido antes de seu fim.

Assim, o problema de JSP pode ser definido da seguinte forma (Croce *et al.*, 1995):

- Existe um conjunto de *m* máquinas e um conjunto de *n* trabalhos (tarefas), constituindo-se, portanto, um problema *n x m*;
- Uma tarefa *n* consiste em um conjunto de operações que devem ser processadas em uma sequência específica;
- Cada operação o_{nm} deve ser processada em uma máquina *m* e tem um tempo de processamento t_{nm} que é deterministicamente conhecido;
- A função objetivo é minimizar o *makespan*, ou seja, minimizar o tempo máximo de conclusão dos trabalhos obedecendo as restrições que são previamente indicadas.

Os planos e as correspondentes ações de produção devem ainda respeitar determinadas restrições impostas, não apenas pela demanda, mas também pela própria estrutura interna do sistema produtivo (Pinha, 2004). Por meio da TCS, pode-se apresentar um formalismo para a descrição das restrições do sistema que permite descrever a estrutura do sistema produtivo, conforme será visto na seção 4.1.

A dificuldade maior do JSP consiste, principalmente, no grande número de restrições que não podem ser evitadas em aplicações do mundo real (Falkenauer & Bouffouix, 1991), ou de outro modo, a grande dificuldade do JSP se encontra na descrição analítica do conjunto de restrições do problema (Silva *et al.*, 2011). Portanto, a representação do sistema de produção através de um formalismo como a TCS permite, além da implementação das estruturas de controle e consequente garantia de especificações, uma base semântica para descrição analítica do problema em si e suas restrições, a partir da qual é possível a aplicação de métodos específicos de otimização.

O espaço de escalonamentos factíveis cresce exponencialmente com o aumento dos trabalhos que devem ser processados, com o número de operações requeridas para cada trabalho, com o tamanho do lote e com as restrições da instalação (Holsapple *et al*., 1993). Por isso, conforme Becerra & Coello (2005), o JSP é um problema *NP-hard*[2] sendo sua classe de problemas de otimização uma das menos tratáveis. Como exemplo de complexidade de resolução desse tipo de problema, os autores apresentam um problema formulado em 1963 por Muth & Thompson, de instância 10 x 10 (10 trabalhos e 10 máquinas), só foi resolvido em 1989 por Carlier & Penson ao utilizar o algoritmo de *Branch and Bound*.

Os JSPs foram resolvidos por métodos analíticos e forneceram soluções ótimas para problemas que sejam pequenos (Chan & Chan, 2004), no caso, poucas tarefas, máquinas e operações. Entretanto, quando o problema se torna de maior, o esforço computacional cresce exponencialmente com o número de trabalhos, operações e máquinas e como o JSP geralmente envolve centenas de máquinas e milhares de trabalhos para serem atribuídos e distribuídos, o problema se torna exponencial e é improvável que algum algoritmo exato de tempo polinomial exista (Hoitmt *et al*., 1993). Então, devido a esse problema de crescimento exponencial surgiu o interesse da aplicação de algoritmos baseados em inteligência artificial (IA) no JSP (metaheurísticas), pois, ao contrário dos algoritmos convencionais e abordagens de simulação, com

[2] Problemas classificados como *NP-hard* são problemas de difícil solução. É um problema que é improvável a existência de algum algoritmo exato de tempo polinomial para este tipo de problema (Rodrigues, 2000).

segue-se gerar escalonamentos de alta qualidade com menos esforço computacional (Holsapple *et al*., 1993). Métodos heurísticos procuram alcançar os resultados para os problemas rapidamente, mas não garantem que a solução obtida seja a solução ótima, mas apresentam resultados satisfatórios. Metaheurísticas também são métodos que procuram alcançar os resultados, porém utilizam de processos capazes de escapar de mínimos/máximos locais, na busca de soluções ótimas.

Ao comparar o problema do escalonamento de tarefas utilizando a otimização tradicional e métodos iterativos, as metaheurísticas usualmente podem encontrar boas soluções com menos esforço computacional e conseguem amenizar a dificuldade que alguns métodos heurísticos tem de escapar de ótimos locais (Gao *et al*., 2014). O sucesso dessas metaheurísticas tem como base a capacidade de proporcionar um equilíbrio entre a diversificação e intensificação, possuir habilidades de computação rápida e encontrar boas soluções com menor esforço computacional. Diversas metaheurísticas têm sido usadas em muitos JSPs, como algoritmos evolucionários, *Tabu Search*, *Simulated Annealing*, Algoritmo da Formiga, Sistemas Imunes Artificiais, entre outros (Becerra & Coello, 2005).

Em 1991, Nakano & Yamada (1991) e Falkenauer & Bouffouix (1991) apresentaram abordagens para o problema de *job-shop scheduling* baseada na utilização de algoritmos genéticos. Um algoritmo genético é uma abordagem que sistematicamente permuta uma população inicial geralmente aleatória gerando uma solução baseada em atributos pré-definidos e que durante a evolução do algoritmo, busca retornar uma melhor solução (Chan & Chan, 2004).

3.3 Algoritmos Genéticos

Nas décadas de 50 e 60, muitos cientistas da computação estudaram sistemas evolucionários com a ideia de que a evolução poderia ser usada como uma ferramenta de otimização para problemas de engenharia. A ideia em todos esses sistemas é envolver uma população de candidatos de soluções para um problema dado, usando operadores inspirados pela variação genética e seleção natural (Mitchell, 1998).

Algoritmos genéticos (AGs) são uma família de modelos computacionais inspirados pela evolução e que codificam uma solução em potencial para um caso específico em uma estrutura de dados, chamada de cromossomo e aplicam operadores nessa estrutura para preservar informações críticas (Whitley, 1994). Utilizam mecanismos baseados na genética para iterativa-

mente gerar novas soluções de atuais soluções acessíveis (Holsapple *et al.*, 1993).

Os AGs foram propostos por John Holland em 1975 em seu trabalho "*Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*" (Holland, 1975), baseado no princípio da seleção natural e recombinação genética. Holland apresentou o AG como uma abstração da evolução biológica e os operadores de *crossover*, inversão e mutação foram suas maiores inovações. Atualmente, os pesquisadores frequentemente utilizam o termo de algoritmo genético para descrever algo muito longe da concepção original de Holland (Mitchell, 1998).

O AG baseia-se no princípio da seleção natural ou também conhecido como Darwinismo. O Darwinismo tem como conceitos básicos:

- Indivíduos competem por recursos que são limitados;
- Devido às características de cada indivíduo, alguns possuem uma maior probabilidade de sobrevivência e reprodução;
- Um grande número de características passa de pai para filho;
- No processo de reprodução pode ocorrer a falha;
- Uma população se constitui de indivíduos, onde cada indivíduo, no AG, constitui uma unidade fundamental que codifica uma possível solução;
- Cada indivíduo é codificado em uma cadeia finita de um alfabeto finito, onde cada cadeia representa um ponto no espaço de busca e é chamado de cromossomo (Falkenauer & Bouffoix, 1991);
- Um símbolo na cadeia é chamado de gene (Croce *et al.*, 1995). Um gene no AG é ou um simples bit ou pequenos blocos de bits adjacentes que codificam um elemento particular de um candidato à solução;
- A aptidão ou *fitness* é tipicamente definida como a probabilidade em que o organismo irá viver ou se reproduzir (Mitchell, 1998).

A noção de avaliação e aptidão, às vezes, são usadas indiferentemente, mas é útil distinguir a função de avaliação da função de aptidão (Whitley, 1994). A função de avaliação, também chamada de função objetivo, fornece uma medida de performance com respeito a um grupo particular de parâmetros. Já a função de aptidão é a função que transforma a medida de performance em uma alocação de oportunidades reprodutivas.

Mitchell (1998), em seu livro, apresenta o seguinte pseudocódigo para um AG simples.

Algoritmo 3.1: Pseudocódigo de um AG simples (Mitchell, 1998).

1. Comece com uma população de *n* cromossomos.
2. Calcule a aptidão *f(x)* de cada cromossomo *x* na população.
3. Repita os seguintes passos até que *n* descendências sejam criadas:
    a. Selecione um par de cromossomos da população atual com a probabilidade de seleção sendo uma função crescente da aptidão.
    b. Com uma probabilidade p_c, recombine o par escolhendo aletoriamente um ou mais pontos para formar 2 descendentes.
    c. Realize a mutação dos dois descendentes com probabilidade p_m.
4. Substitua a população atual com a nova população.
5. Se o número de iterações for menor que o número de gerações, vá para o passo 2. Senão, finalize e apresente o melhor cromossomo.

O primeiro passo para a construção de um AG se constitui na definição de uma codificação que permite mapear as soluções como um todo em um conjunto de cadeias de símbolos (Croce *et al*., 1995).

3.3.1 Representação para o problema de escalonamento de tarefas

Uma das características básicas do algoritmo genético é trabalhar em um espaço de codificação e um espaço de solução de maneira alternada. Assim, no que diz respeito a evolução, o AG trabalha no espaço da codificação, enquanto no quesito de avaliação, o AG trabalha com o espaço da solução (Cheng *et al*., 1996). Uma codificação é escolhida de modo que cada ponto no espaço de busca é representado por exatamente uma cadeia chamada cromossomo, permitindo que o AG trabalhe com essas representações (cromossomos) em vez das próprias soluções (Falkenauer & Bouffouix, 1991).

Holland (1975) propôs a codificação utilizando cadeias binárias, porém, nem sempre essa codificação se mostra útil para a resolução de diversos tipos de problemas. Durante a década de 80 e 90, surgiram diversas técnicas de codificação que foram criadas para problemas particulares no qual a técnica criada por Holland em 1975 dificilmente pode ser aplicada (Cheng *et al*., 1996). Nos problemas de escalonamento e ordenamento, a codificação é, em geral, tal que os cromossomos são permutações de cadeias básicas (Croce *et al*., 1995). Conforme Cheng *et al*. (1995), várias técnicas de codificação não-binárias foram desenvolvidas e três questões devem ser consideradas na escolha da codificação do cromossomo:

- *a viabilidade do cromossomo:* refere-se ao fenômeno da solução decodificada do problema esteja em uma região do espaço de busca que seja viável para a resolução do problema e relaciona-se com a natureza do problema de otimização;
- *a legalidade do cromossomo:* refere-se ao fenômeno em que o cromossomo representa uma solução do problema e relaciona-se com as técnicas de codificação;
- *a singularidade do cromossomo:* refere-se ao fenômeno de que a codificação represente apenas uma solução.

Em JSPs de tamanho $n \times m$, diversos tipos de codificações têm sido propostos na literatura (Becerra & Coello, 2005).

No presente trabalho, as codificações do cromossomo para o JSP baseiam-se em "*A Tutorial Survey of Job-Shop Scheduling Problems Using Genetic Algorithms - I Representation*" de Cheng, Gen e Tsujimura (Cheng *et al.*,1995). Segundo os autores, devido a existência das restrições de precedência das operações do JSP, não é fácil encontrar ou desenvolver uma codificação e não existe uma representação das restrições de precedência através de um sistema de inequações. Mesmo utilizando uma abordagem de penalização para cromossomos ilegais, esse tipo de abordagem não consegue ser facilmente aplicado em alguns tipos de restrições.

Cheng *et al.* (1995) apresenta nove técnicas de representação (codificação e decodificação) do cromossomo e estas estão divididas em dois grupos: abordagem direta e abordagem indireta. Na classe da abordagem direta, a representação dos cromossomos é construída de maneira direta, ou seja, o escalonamento é codificado em um cromossomo, porém os operadores genéticos são mais complexos de se aplicar. Na classe de abordagem indireta, as representações constituem-se de regras de despacho que atribuem as tarefas, mas não constituem uma lista. Na abordagem indireta, essas regras são codificadas no cromossomo e os cromossomos evoluem para encontrar melhor sequência de regras de despacho e é, então, construído um escalonamen-

to com tais regras de despacho. Esta última forma de abordagem necessita de um tradutor. Na Tabela 3.1 apresenta-se as codificações apresentadas de cada abordagem proposta por Cheng *et al*. (1995).

Tabela 3.1 – Lista de representações de cromossomos para JSPs.

Abordagem Direta	**Abordagem Indireta**
Operation-based representation	*Preference list-based representation*
Job-based representation	*Priority rule-based representation*
Job pair relation-based representation	*Disjunctive graph-based representation*
Completion time-based representation	*Machine-based representation*
Random keys representation	-

Para o entendimento das representações e de como é realizada a codificação das mesmas, utiliza-se um exemplo de um JSP de tamanho 3x3, apresentado na Tabela 3.2, onde u.t. constituem as unidades de tempo, J_i são as tarefas/trabalhos, M_i são as máquinas. Na parte intermediária da tabela são apresentadas as sequências de máquinas que são utilizadas para a realização da tarefa. Por fim, na parte inferior apresentam-se as restrições de precedência de cada uma das tarefas.

- ## Abordagem direta

Para a abordagem direta, seguindo a Tabela 3.1, são apresentadas 5 representações: *operation based-representation* (representação baseada em operação), *job-based representation* (representação baseada em trabalhos), *job pair relation-based representation* (representação baseada na relação de par de trabalho), *completion time-based representation* (representação baseada no tempo de conclusão) e *random keys representation* (representação em chaves aleatórias).

a) Representação baseada em operação

Essa representação codifica um escalonamento como uma sequência de operações, onde cada gene representa uma dessas operações e, geralmente, esses genes são representados utilizando-se os números naturais (Cheng *et al*., 1995). Alguns trabalhos apresentam essa codificação, como por exemple Gen & Tsujimara (1994), Kubota (1995), Michaliwick (1995), Silva et al (2011), Alves (2012), entre outros.

Tabela 3.2 - Problema exemplo.

Trabalho	**Operações**		
	1	2	3
Tempo de processamento (u.t.)			
J1	3	3	2
J2	1	5	3
J3	3	2	3
Sequência de máquinas			
J1	M1	M2	M3
J2	M1	M3	M2
J3	M2	M1	M3
Restrições de precedência			
Máquina	Trabalho		
M1	J2	J1	J3
M2	J3	J1	J2
M3	J2	J1	J3

Como é fácil perceber, a permutação desses números pode gerar escalonamentos ilegais devido às restrições de precedência. Para contornar esse problema das restrições de precedência, é proposta a nomeação de todas as operações de um trabalho com o mesmo símbolo e que a interpretação é de acordo com a ordem de ocorrência (Gen & Tsujimara, 1994).

Essa codificação é fácil de se ler, porém já foram encontrados problemas com vários experimentos considerando-se mais de uma máquina, já que a sequência definida pelo cromossomo pode ser incompatível com as restrições e, também, nem todos os cromossomos definem soluções factíveis (Cheng *et al*., 1995).

A codificação consiste em um cromossomo tamanho definido pelo número de trabalhos (denominado *n*), pelo número de máquinas (denominado *m*) e/ou pelo número de operações (*j*). Assim, um cromossomo terá *n x j* genes, onde cada trabalho/tarefa aparece no cromossomo exatamente *m* vezes.

Um cromossomo é decodificado sendo primeiramente traduzido para uma ordem de operações. A primeira operação no cromossomo é agendada primeiro, na sequência, a segunda e assim sucessivamente. Cada operação é agendada na máquina para que o tempo de processamento total seja o menor possível. Por exemplo, o cromossomo [2 1 1 1 2 2 3 3 3], representa a alocação de tarefas da Figura 3.2 (cromossomo decodificado), e indica que a primeira operação da tarefa 2 foi agendada na máquina M_1 respeitando a ordem de precedência da tarefa 2. Em seguida a primeira operação da tarefa 1 foi agendada na máquina M_1, respeitando a ordem de precedência da tarefa 2. Agendou-se então a segunda e terceira operação da tarefa 1 nas má-

quinas M_2 e M_3, respectivamente, respeitando a ordem de precedência. Agendou-se, então, a segunda e terceira operação nas máquinas M_3 e M_2, respectivamente. Ao agendar as operações da tarefa 3, percebe-se a que a primeira operação deve ser agendada na M_2 e há tempo hábil para o agendamento antes da segunda operação da tarefa 1, então agenda-se a primeira operação da tarefa 3 antes da segunda operação da tarefa 1. As outras duas operações são agendadas assim que possível nas máquinas M_1 e M_3, respectivamente.

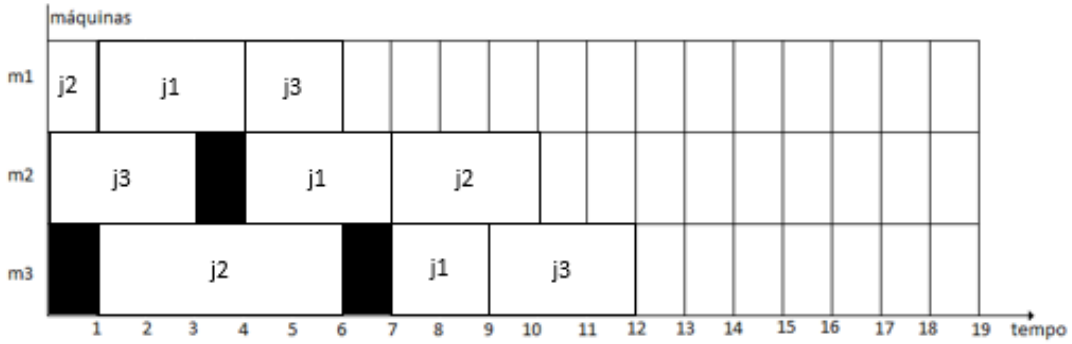

Figura 3.2 - Cromossomo decodificado do exemplo proposto utilizando a representação baseada em operação.

b) Representação baseada em trabalho

Consiste de uma lista de *n* trabalhos (tarefas) e um escalonamento é construído de acordo com uma sequência de trabalhos (Cheng *et al.*, 1995). O cromossomo terá *n* genes que representam as operações.

Essa codificação é apresentada no trabalho de Holsapple *et al*. (1993). É importante salientar que essa codificação sempre resultará em um escalonamento factível ou legal. Nessa codificação, todas as operações do primeiro trabalho são agendadas primeiro, depois todas as operações do segundo trabalho são agendadas no melhor tempo de processamento da máquina que a operação requer e assim, até serem agendadas todas as operações.

Para o exemplo proposto na Tabela 3.2, a decodificação do cromossomo [2 3 1] apresenta-se na Figura 3.3. Para esse exemplo de cromossomo, primeiramente se agendou todas as operações da tarefa 2 seguindo a ordem de precedência das operações. Em seguida, agendou-se todas as operações da tarefa 3 no melhor tempo hábil possível e obedecendo as regras de precedência. Por fim, agendou-se todas as operações da tarefa 1 no melhor tempo hábil possível e seguindo as regras de precedência.

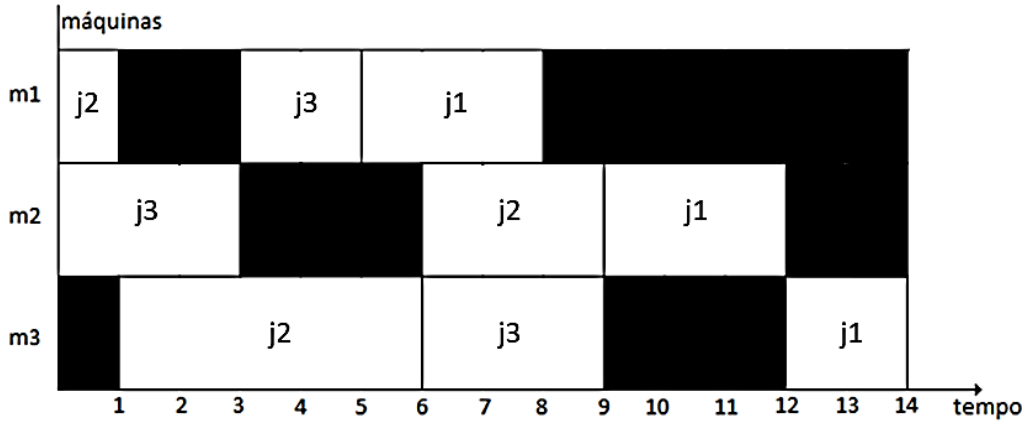

Figura 3.3 - Cromossomo decodificado do exemplo proposto utilizando a representação baseada em tarefas.

c) Representação baseada na relação de par de trabalho

Essa representação foi proposta por Nakano & Yamada (1991), e utiliza uma matriz binária para codificar um escalonamento. A matriz é construída de acordo com a relação de precedência de um par de trabalhos nas *m* máquinas correspondentes. Uma variável binária x_{ijm} é definida para indicar a precedência para um par de trabalhos e sendo *i* o trabalho *i*, *j* o trabalho *j* e *m* a máquina, uma variável da matriz binária é definida pela seguinte Expressão 3.1:

$$x_{ijm} = \begin{Bmatrix} 1,\ se\ a\ tarefa\ i\ \text{é}\ processada\ antes\ de\ j\ na\ máquina\ m \\ 0,\ outras \end{Bmatrix} \qquad (3.1)$$

Essa representação é provavelmente a mais complexa das apresentadas em (Cheng *et al.*, 1995), altamente redundante e quase todos cromossomos produzidos são ilegais. Para o exemplo proposto, analisando as restrições de precedência apresentadas na Tabela 3.2, o cromossomo codificado na matriz binária é apresentado na Figura 3.4. Por exemplo, para o gene do cromossomo x_{121}: na máquina M_1, a tarefa J_1 não tem preferência para a tarefa J_2, sendo então representado por um 0 na matriz binária. No caso do gene x_{122} cromossomo: na máquina M_2, a tarefa J_1 tem preferência em relação a tarefa J_2, então é representado por um 1 na matriz binária. Assim, todas os genes do cromossomo são calculados e preenchem a matriz binária.

$$\begin{pmatrix} x_{121} & x_{122} & x_{123} \\ x_{131} & x_{132} & x_{133} \\ x_{231} & x_{233} & x_{232} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Figura 3.4 - Cromossomo codificado do exemplo proposto utilizando a representação baseada na relação de pares de tarefas.

d) Representação baseada no tempo de conclusão

Essa representação foi proposta por Yamada & Nakano (1992) e se baseia no tempo de conclusão das operações. A codificação do cromossomo é feita na forma de uma lista ordenada de tempo de conclusão das operações. Essa representação não é apropriada para a maioria dos operadores genéticos, já que gera um escalonamento ilegal (Cheng *et al*., 1995). Por isso, Yamada & Nakano criaram operadores genéticos especiais para essa representação. Na Expressão 3.2 é apresentado a forma do cromossomo que possui tamanho *j* x *m* (número de trabalhos *j* versus número de máquinas *m*), onde c_{ijm} representa o tempo de finalização de cada operação *i* da tarefa *j* na máquina *m*.

$$[\, c_{ijm}\, c_{ijm}\;\; c_{ijm}\;\; c_{ijm}\;\; c_{ijm}\;\; c_{ijm} \ldots] \tag{3.2}$$

e) Representação de chaves aleatórias

A representação por chaves aleatórias foi proposta inicialmente por Bean (1994), na qual as operações genéticas podem produzir descendência factível sem criar a sobrecarga adicional. Posteriormente, Norman & Bean (1995) propuseram melhorias a essa representação.

Essa representação codifica a solução através de números aleatórios nos quais são usados como chaves de classificação para decodificar a solução (Cheng *et al*., 1995). Sendo um JSP de *n* trabalhos e *m* máquinas, cada gene consiste em um número aleatório, constituído de duas partes: um inteiro entre 1 a *m* e uma fração gerada aleatoriamente entre 0 e 1. A parte inteira é interpretada como uma máquina determinada para a tarefa e a parte fracionária provê a sequência de tarefa em cada máquina.

Seja por exemplo, a codificação do cromossomo para o problema exemplo apresentado na Expressão 3.3. A decodificação para esse cromossomo apresenta-se na Expressão 3.4, onde cada elemento é caracterizado em o_{jm}, onde *j* é o tarefa e *m* a máquina. A ordem dos genes dá a ordem de agendamento de cada trabalho nas máquinas. É importante salientar que essa representação pode violar as condições de precedência.

$$[1.34\ 1.09\ 1.88\ 2.66\ 2.91\ 2.01\ 3.23\ 3.21\ 3.44] \tag{3.3}$$

$$[o_{21}\ o_{11}\ o_{31}\ o_{32}\ o_{12}\ o_{22}\ o_{23}\ o_{13}\ o_{33}] \tag{3.4}$$

- **Abordagem indireta**

Para a abordagem indireta, seguindo a Tabela 3.2, são apresentadas 4 representações: *preference list-based representation* (representação baseada em lista de preferência), *priority rule-based representation* (representação baseada em regra de prioridade), *disjunctive graph-based representation* (representação baseada em gráficos disjuntivos) e *machine-based representation* (representação baseada em máquina).

a) Representação baseada em lista de preferência

Essa representação foi primeiramente proposta por Davis (1985). Ela também aparece em trabalhos como o de Falkenauer & Bouffoix (1991), Croce *et al.* (1995) e Kobayashi & Yamamura (1995).

Em um JSP de tamanho *j* trabalhos por *m* máquinas, um cromossomo é formado de m subcromossomos, sendo cada um de uma máquina. Cada subcromossomo é constituído de uma cadeia de símbolos de tamanho *j* e cada um desses símbolos identifica uma operação que deve ser processada na máquina requisitada (respectivo subcromossomo) (Cheng *et al*., 1995). É importante saber que um subcromossomo não descreve uma sequência de operações na máquina, mas apenas uma lista de preferência. Por isso, deve-se identificar a máquina crítica para que se possa iniciar mais cedo e então selecionar a operação a ser processada.

Utilizando o exemplo proposto na Tabela 3.2 e tendo como um exemplo de subcromossomo o apresentado na relação 3.5.

$$[(2\ 3\ 1)\ (1\ 3\ 2)\ (2\ 1\ 3)] \tag{3.5}$$

Na Figura 3.5 apresenta-se a decodificação desse cromossomo. A decodificação desse exemplo será da seguinte maneira: desse cromossomo pode-se deduzir que as primeiras operações que tem preferência são operação da J_2 na máquina M_1, operação da J_1 na máquina M_2 e operação J_2 na máquina M_3. De acordo com as restrições de precedência, é possível agendar apenas a operação da tarefa J_2 na máquina M_1. Então, as novas preferências são a operação de J_3 na máquina M_1, operação de J_1 na máquina M_2 e operação J_2 na máquina M_3. De acordo com as restrições de precedência só se consegue agendar a operação de J_2 na máquina M_3. Agenda-se daí as operações J_1 na máquina M_1 e J_3 na máquina M_2. Agora, as preferências são as operações J_3 na máquina M_1 e a J_1 em M_2, e de acordo com as regras de precedência podem se agendar as duas operações. As novas preferências são as operações de J_2 na máquina M_2 e J_1 em M_3 e

e de acordo com as regras de precedências, essas operações são agendadas. Por fim, a única preferência é a operação J_3 na máquina M_3 e ela é agendada.

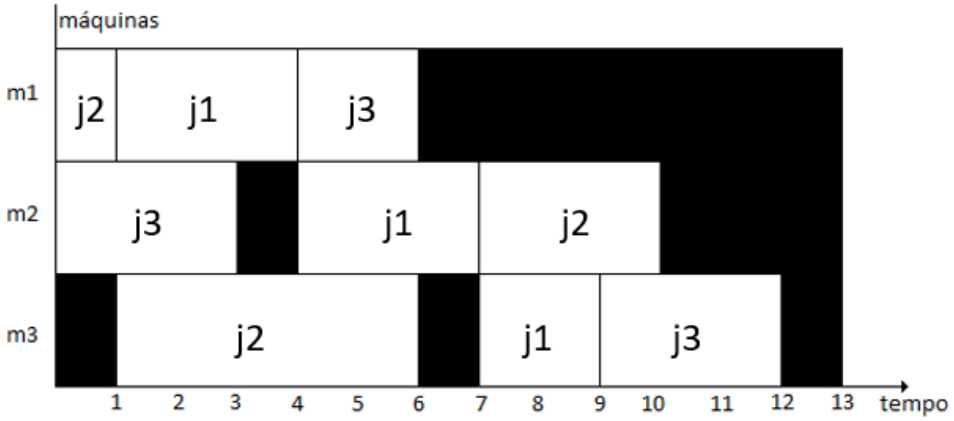

Figura 3.5 - Cromossomo decodificado do exemplo proposto utilizando a representação baseada em lista de preferência.

b) Representação baseada em regra de prioridade

Essa representação foi proposta por Dorndorf & Pesch (1995). Nessa representação, um cromossomo codifica uma sequência de regras de despacho para a atribuição de um trabalho e essa sequência é construída com uma heurística de despacho prioritário que se baseia na sequência de regras de despacho (Dorndorf & Pesch, 1995). Essas regras de despacho prioritário provavelmente são as heurísticas mais comuns aplicadas para resolver JSP, pois são fáceis de implementar e não possuem muita complexidade no que diz a respeito ao tempo (Gli ffler & Thompson, 1960). Porém, o problema dessa representação é a identificação de uma regra de prioridade que seja eficiente (Cheng *et al*., 1995).

Em um JSP de tamanho *j* trabalhos e *m* máquinas, um cromossomo é uma cadeia de entrada de tamanho *n* x *m*, sendo apresentada na Expressão 3.6 (Cheng *et al*, 1995). Uma entrada p_i representa uma regra de um conjunto de regras que foram anteriormente especificadas e essa entrada p_i refere-se a entrada da posição *i*, que estabelece que um conflito na iteração *i* deve ser resolvido utilizando a regra de prioridade *i*.

$$(p_1, p_2, \dots, p_{nm}) \tag{3.6}$$

Conforme Cheng *et al.* (1995), o procedimento para a dedução da descendência de um cromossomo $(p_1, p_2, \dots, p_{nm})$ é mostrado a seguir. Seja as seguintes variáveis:

- PS_t: uma sequência parcial contendo *t* operações agendadas;
- S_t: o conjunto de operações agendáveis na iteração t, correspondente ao PS_t dado;
- σ_i: o menor tempo que a operação $i \epsilon S_t$ pode ser iniciada;
- ϕ_i: o menor tempo que a operação $i \epsilon S_t$ pode ser finalizada;
- C_t: o grupo de operações conflitantes na iteração t.

O procedimento para a decodificação do cromossomo é apresentado a seguir:

- **Passo 1:** Seja t=1 e comece com PS_t como uma sequência vazia e S_t inclui todas as operações com nenhuma precedência;
- **Passo 2:** Determine $\phi_t^* = \min_{i \epsilon St}\{\phi_i\}$ e a máquina m* em que ϕ_t^* poderá ser realizada;
- **Passo 3:** Forme o conjunto conflitante C_t que inclui todas as operações $i \epsilon S_t$, com $\sigma_i < \phi_t^*$ que requer a máquina *m**. Selecione uma operação de C_t pela regra de prioridade p_t e adicione essa operação em PS_t o mais cedo possível, então crie uma nova sequência parcial PS_{t+1};
- **Passo 6:** Atualize PS_{t+1} removendo a operação selecionada de S_t e adicionando o sucessor direto da operação para S_t. Incremente t.
- **Passo 5:** Retorne ao passo 2 até que uma sequência completa é gerada.

Para um melhor entendimento, utiliza-se o exemplo proposto na Tabela 3.3. Para o cromossomo de exemplo mostrado na Expressão 3.7, sendo o_{jim} o gene, j a tarefa, i a operação e m a máquina, a decodificação será da seguinte maneira: primeiramente, as três operações que competem são a operação 1 da tarefa 1 na máquina 1 (o_{111}), operação 1 da tarefa 2 na máquina 1 (o_{211}) e operação 1 da tarefa 3 na máquina 2 (o_{312}). Os tempos de processamento finais de processamento das operações que seriam processadas por tarefas são 3, 1 e 3 u.t. (unidades de tempo), das o_{111}, o_{211} e o_{312}, respectivamente. Visto que o objetivo é a minimização do tempo, escolhemos a máquina 1 (to_{211}=1) e obedecendo a regra de prioridade determinada pelo cromossomo (regra 1), agenda-se a operação 1 da tarefa 2 na máquina 1 (o_{111}). Então, avança-se para o próximo agendamento. Sendo as três operações disputadas (o_{111}, o_{223} e o_{312}) e os tempos finais de processamento de cada uma dessas tarefas (4, 6, 3) escolhe-se a máquina 2. Atendendo

a regra de prioridade estipulada, agenda-se a operação 1 da tarefa 3 na máquina 2 (o_{312}). Novamente, avança-se para o próximo agendamento e segue-se os mesmos passos. O cromossomo inteiramente codificado está na Figura 3.6.

Tabela 3.3 - Exemplo de tabela de regras de despacho ou de prioridade.

Regra	Descrição
1	Selecione a operação com menor tempo de processamento
2	Selecione a operação com maior tempo de processamento
3	Selecione a operação para o trabalho com o maior tempo de processamento restante
4	Selecione a operação para o trabalho com o menor tempo de processamento restante

$$[1\ 2\ 2\ 1\ 4\ 4\ 2\ 1\ 3] \tag{3.7}$$

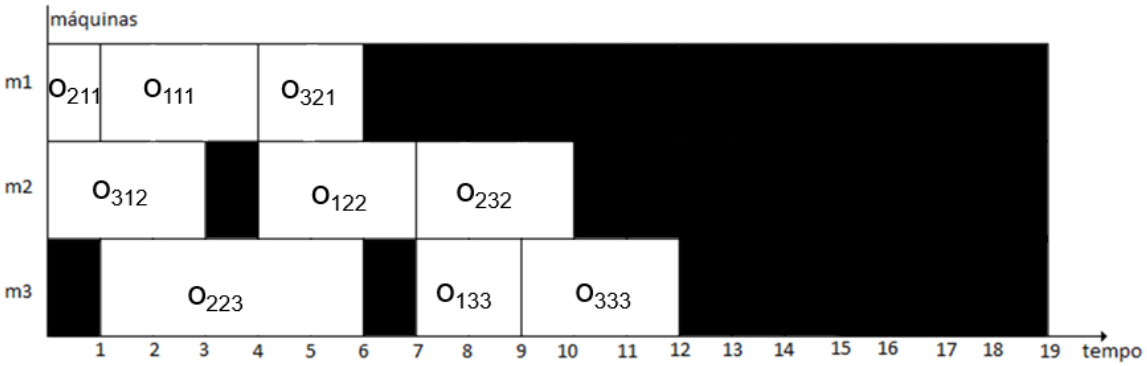

Figura 3.6 - Cromossomo decodificado do exemplo proposto utilizando a representação baseada em regra de prioridade.

c) Representação baseada em gráficos disjuntivos

Essa representação foi proposta por Tamaki & Nishikawa (1992), sendo considerada uma representação baseada na relação de pares de trabalhos. O gráfico disjuntivo é definido por uma tripla.

$$G = (N, A, E) \tag{3.8}$$

O JSP pode ser representado com um gráfico disjuntivo (Roy & Sussman, 1982; Ballas, 1969). Por definição, *N* contém os nós que representam as operações, A contém os arcos conectando as operações consecutivas e contém os arcos disjuntivos conectando as operações que serão processadas na mesma máquina. As restrições se encontram em *E*. O arco disjuntivo possui duas orientações e a construção de um escalonamento indicará as orientações de todos os arcos disjuntivos para determinar as sequências de operações (Cheng *et al*, 1995). O cromosso-

mo para essa representação constituirá em uma cadeia binária que corresponde em uma ordem de arcos disjuntivos de *E*. A cadeia é constituída de símbolos e_{ij} que tem seus valores determinados pela relação 3.8.

$$e_{ij} = \begin{cases} 1, orientação\ do\ arco\ disjuntivo\ do\ nó\ j\ para\ o\ nó\ i \\ 0, orientação\ do\ arco\ disjuntivo\ do\ nó\ i\ para\ o\ nó\ j \end{cases} \quad (3.9)$$

d) Representação baseada em máquina

Para o entendimento dessa representação é necessário o entendimento de uma heurística conhecida como *shifting bottleneck heuristic* (heurística do "gargalo móvel") proposta por Adams *et al.* (1988). Essa heurística é um dos mais poderosos procedimentos conhecidos entre todas as heurísticas para o JSP, e funciona da seguinte maneira: sequenciar as máquinas uma por uma, sucessivamente, tomando o tempo de cada máquina como um gargalo entre as máquinas que não são sequenciadas e cada vez que uma máquina é sequenciada, todas as sequências são novamente otimizadas (Cheng *et al*., 1995).

Dorndorf & Pesch (1995) propuseram um algoritmo genético baseado em máquina, onde o cromossomo decodifica a sequência das máquinas e a sequência é construída com uma heurística do "gargalo móvel". Eles apresentam o cromossomo com uma lista de máquinas ordenadas e AGs são utilizados para melhorar os cromossomos para encontrar uma melhor sequência de máquinas para a heurística do gargalo móvel.

Tanto a identificação do gargalo quanto os procedimentos locais de reotimização são ba seados repetidamente na solução do problema de escalonamento de uma máquina que é um relaxamento do problema original.

3.3.2 Operadores genéticos

AGs processam populações de cromossomos substituindo sucessivamente um membro da população por outro e na sua forma mais simples (Mitchell, 1998).

Um AG opera em uma população de tamanho fixo de maneira iterativa (Holsapple *et al*, 1993). O primeiro passo para a construção do AG se resume na síntese da população inicial. O primeiro passo para a construção do AG se resume na síntese da população inicial. Uma população inicial de soluções é selecionada e é considerada a primeira geração, frequentemente produzida de maneira aleatória. Um AG, ao contrário das técnicas padrões de otimização, executa

uma pesquisa em conjunto de pontos no espaço de busca, diminuindo assim a probabilidade de estar preso em um ótimo local (Falkenauer & Bouffouix, 1991). Há alternativas que buscam suprir deficiências que existem na criação aleatória de indivíduos de representação mais complexa buscando uma melhor performance e isto aparece na utilização de procedimentos heurísticos como geradores de populações iniciais.

Após a geração da população inicial, deve-se calcular a aptidão da população para seguir para o próximo passo. Existem 3 operadores genéticos básicos que são usados por um AG: seleção, *crossover* e mutação (Holsapple *et al*., 1993).

a) Seleção

Após o cálculo da aptidão para as cadeias na população atual, a operação de seleção é realizada (Whitley, 1994). O operador de seleção é o operador que seleciona os cromossomos para a reprodução levando em consideração a qualidade do cromossomo (Mitchell, 1998).

A aptidão é calculada refletindo a maneira como cada indivíduo é em comparação aos outros. Os indivíduos com maior aptidão são selecionados de acordo com uma "seleção ruidosa". Nessa "seleção ruidosa", os indivíduos são selecionados aleatoriamente mas com a probabilidade aumentando de acordo com a aptidão, sendo assim, os AGs são essencialmente, uma técnica de otimização aleatória (Falkenauer & Bouffouix, 1991). Esses indivíduos selecionados são chamados de pais.

Os cromossomos são selecionados para serem pais de um cruzamento, porém o problema se encontra em como selecionar esses cromossomos (Obitko, 1998). Segundo a teoria de Dar win, o melhor indivíduo sobrevive para criar a descendência e por isso, diversos métodos foram criados para realizar essa operação de seleção, como o método da seleção por roleta, seleção por torneio, seleção por classificação, entre outros. Abaixo segue a explicação sobre os principais métodos de seleção:

- *Seleção por roleta:* nesse método cada indivíduo é representado por um espaço que proporcionalmente corresponde à sua aptidão e ao "girar a roleta", os indivíduos são escolhidos (Whitley, 1994);
- *Seleção por torneio:* esse método foi desenvolvido por Goldberg (1991) e a seleção é feita sorteando-se um número fixo de indivíduos e o melhor dentre eles é selecionado para ser o pai.

- *Seleção por classificação:* o método da seleção por roleta apresenta problemas quando existem grandes diferenças entre os valores da aptidão. O método da seleção por classificação seleciona a população e atribui a cada cromossomo um valor de adequação determinado pela sua classificação, onde o pior terá adequação igual a 1, o segundo pior igual a 2 e assim por adiante (Obitko, 1998). Esse método pode apresentar uma menor convergência que o método de seleção por roleta.

Esses indivíduos ou cromossomo selecionados serão utilizados para a geração de seus descendentes. Dependendo dos operadores e do contexto do problema, cada pai irá gerar um ou mais descendentes (Holsapple *et al*., 1993). Após a operação de seleção de ter sido realizada, a recombinação (*crossover*) pode acontecer (Whitley, 1994).

b) *Crossover*

Os indivíduos selecionados formam um conjunto de pais e eles são cruzados para produzir seus descendentes (Falkenauer & Bouffouix, 1991). O operador de *crossover* padrão é chamado de crossover simples ou de 1 ponto, mas há inúmeras variações, como o operador de 2 pontos, multipontos, entre outros (Holsapple *et al*, 1993).

- *Operador de crossover de 1 ponto (conhecido por 1PX):* esse operador de *crossover* produz dois descendentes, onde um ponto de corte é escolhido aleatoriamente e os descendentes são obtidos por usar a primeira parte de um pai, até o ponto, e a segunda parte do outro pai (Croce *et al*., 1995); *Operador de crossover de 2 pontos (conhecido por 2PX):* nesse operador são escolhidos dois pontos aleatoriamente e os descendentes são obtidos dos pais ao trocar os bits entre os dois pontos de corte (Croce *et al*., 1995);
- *Operador de multipontos de corte (conhecido por MPX):* nesse tipo de *crossover* é sorteado um número fixo de n pontos de corte e os descendentes ou filhos são as combinações alternadas. Os operadores 1PX e 2PX apresentados anteriormente são casos particulares desse operador;
- *Operador de crossover parcialmente mapeado (conhecido por PMX):* muitas vezes, os operadores genéticos podem gerar cromossomos que sejam ilegais e as vezes é preciso redefinir esses operadores para produzir soluções factíveis. Desenvolvido por Goldberg (1985), neste operador são sorteados dois pontos de corte. Os descendentes recebem na íntegra os genes dos pais situados entre os dois pontos de corte e então, o cromossomo é preenchido de maneira que seus valores sejam os mais adequados. Esse operador é a-

adequado para operar em permutações (Croce *et al*. 1995).

- *Operador de crossover de ordem (conhecido por OX):* desenvolvido por Goldberg (1989) é uma variante do operador PMX. No OX, o cromossomo é considerado circular e esse operador trabalha com posições relativas ao invés de posições absolutas, como é o caso do PMX.
- *Operador de crossover de ordem linear (conhecido por LOX):* esse operador foi apresentado por Falkenauer & Bouffouix (1991) e é uma adaptação do operador OX, uma vez que para o JSP o cromossomo não pode ser considerado circular. Ele funciona da maneira que se segue: são sorteados dois pontos de corte, os símbolos que aparecem no pai 1 entre esses dois pontos de corte são apagados no pai 2 formando "buracos". Esses "buracos" deslizam para o interior dos dois pontos de corte e nesses "buracos" são adicionados os genes do primeiro pai. O mesmo acontece para o segundo descendente, mas apenas alterando a ordem dos pais.

Após a geração da descendência através dos operadores de seleção e *crossover*, aplica-se o operador de mutação (Whitley, 1994).

c) Mutação

As mutações ocasionais asseguram que genes que são úteis e não estão presentes na população atual tenham uma chance de entrar no contexto genético (Croce *et al*., 1995). O obje- tivo da mutação é introduzir novo material genético em um indivíduo, em consequência, adicionar diversidade para as características genéticas da população (Engelbrecht, 2007).

Após o *crossover*, aplica-se o operador de mutação, isto é, para cada gene na população, ele será mutado com uma probabilidade p_m. Essa probabilidade, geralmente, apresenta-se em um valor menor que 1. A mutação é usada como um suporte ao *crossover* para garantir a gama completa de todos os genes (Engelbrecht, 2007).

Existem diversos operadores de mutação. O operador de mutação mais simples apenas troca de forma aleatória um dos genes (Falkenauer & Bouffouix, 1991).

Para o caso do JSP, Croce *et al*. (1995) propõem a utilizam do operador de mutação chamado *swap*, uma vez que esse operador escolhe aleatoriamente 2 genes e os troca de lugar, garantindo assim que a mutação não leve a cromossomos que sejam ilegais.

Existe também um outro operador genético que foi proposto por Holland, o operador da inversão. Esse operador já não é mais tão utilizado atualmente (Mitchell ,1998). Esse operador de inversão apenas inverte a ordem de uma seção continua de um cromossomo.

Após o processo de seleção, recombinação (*crossover*) e mutação, os descendentes podem ser avaliados e inseridos na população, já que o processo de avaliação, seleção, recombinação e mutação formam uma geração na execução de um AG (Whitley, 1991). Se uma geração é criada, a descendência substitui os pais na população e essa substituição pode ser baseada em restrições, como por exemplo, a restrição que a descendência deve ser melhor que o pior pai, ou por exemplo, a restrição forçada que todos os descendentes substituem os pais (Engelbrecht, 2007).

Em um AG existe um número de detalhes a decidir, como o tamanho da população, probabilidades de crossover e mutação e número de gerações e o sucesso do algoritmo depende muito desses detalhes (Mitchell, 1998). A performance de um AG é influenciada pela taxa de mutação p_m e pela taxa de *crossover* p_c (Engelbrecht, 2007). Não existem valores padrões para esses parâmetros, pois dependem muito da aplicação do problema. Segundo Mitchell (1998), um AG tem tipicamente entre 50 a 500 gerações.

3.4 Considerações finais

Neste capítulo foram, primeiramente, apresentados os níveis de automação existentes em sis-temas de manufatura, abordando os níveis gerenciais e de controle de chão de fábrica. É discutido também sobre o sistema de supervisão, o SCADA, e como este é de fundamental importância e pode possibilitar a integração dos níveis de automação dentro de um SFM.

No contexto do gerenciamento da produção, é apresentado um problema clássico conhecido como escalonamento de tarefas, problema esse alvo de muitas pesquisas. O escalonamento de tarefas busca encontrar sequências ótimas de tarefas para alcançar os objetivos da produção.

Foram desenvolvidas e aplicadas diversas técnicas de otimização para a resolução desse problema, técnicas como a utilização de métodos analíticos, procedimentos heurísticos, entre outros. Dentre muitos dos procedimentos heurísticos aplicados ao problema de escalonamento, neste trabalho destaca-se a metaheurística conhecida como Algoritmos Genéticos.

O algoritmo genético, primeiramente formulado por Holland em 1975, é muito aplicado em resoluções de problemas de escalonamento de tarefas de inúmeras maneiras. A escolha dos

AGs para a resolução do problema de escalonamento de tarefas consiste na possibilidade de contornar problemas de limitação de *hardware* e conseguir resultados com um tempo viável, ainda que esses sejam subótimos. Para a construção do AG é necessário escolher uma codificação/representação para o cromossomo. Neste trabalho empregou-se três representações: a representação baseada em operação, a representação em chaves aleatórias e a representação baseada em regras de prioridade.

No próximo capítulo, propõe-se uma metodologia para a implementação de um AG para a resolução de problemas de escalonamento de tarefas dentro de um SFM utilizando a TCS para a descrição formal do sistema. Como forma de validar a metodologia proposta é, então, aplicado a metodologia a um estudo de caso e posteriormente, os resultados da aplicação são apresentados.

Capítulo 4

Metodologia Proposta e Estudo de Caso

O escalonamento em sistemas de manufatura busca encontrar sequências ótimas de tarefas, de modo que os objetivos de produção sejam alcançados, entre eles, o aumento da produtividade e maximização do uso de equipamentos (Hoitomt, Luh & Pattipati, 1993; Shi & Pan, 2005; Pinha, Queiroz & Cury, 2011).

Os problemas de escalonamento de tarefas são resolvidos por métodos analíticos e fornecem soluções ótimas para problemas que sejam pequenos, porém, quando a dimensão do problema aumenta, o esforço computacional cresce de maneira exponencial com o número de tarefas, operações e máquinas envolvidas (Chan & Chan, 2004). O alto custo para a obtenção de soluções ótimas pode ser proibitivo, pois o problema de escalonamento pertence à classe de problemas NP-complexo, cuja solução polinomial não é conhecida (Oliveira *et al*., 2013).

Neste capítulo será apresentada, primeiramente, uma metodologia para a resolução de problemas de escalonamento de tarefas empregando algoritmos genéticos juntamente com a Teoria do Controle Supervisório como forma de descrição analítica das restrições do sistema. Posteriormente, como forma de validação da metodologia, será utilizado um estudo de caso e que são implementados três diferentes representações do cromossomo. Por fim, são apresentados os resultados encontrados.

4.1 Metodologia Proposta

Ao comparar a otimização tradicional e métodos iterativos, as metaheurísticas usualmente podem encontrar boas soluções com menor esforço computacional e conseguem amenizar a dificuldade que alguns métodos heurísticos tem de escapar de ótimos locais (Gao *et al*., 2014).

Outra dificuldade na abordagem do escalonamento de tarefas está relacionada com a descrição analítica de restrições do problema. Restrições são necessárias para garantir que as sequências de produção sejam consideradas "legais" ou factíveis.

Através de abordagens baseadas em sistemas a eventos discretos (SED), é possível encontrar uma representação formal para o problema, em termos de ações e eventos, possibilitando descrever matematicamente o problema e restrições associadas. Por exemplo, a Teoria do Controle Supervisório (TCS) objetiva a modelagem, análise e controle do SED através de estruturas de controle (supervisores) que sejam minimamente restritivos, ou seja, evitando somente trajetórias de eventos consideradas proibidas ou indesejáveis (Silva *et al*., 2011). Visto a grande dificuldade, já citada, encontrada na descrição analítica das restrições dos sistemas para o problema do escalonamento de tarefas, uma vez que não se consegue descrevê-las através de um conjunto de inequações, emprega-se o controle supervisório. Devido ao caráter minimamente restritivo, o controle supervisório fornece o conjunto de todas as sequências possíveis, porém não escolhe uma sequência em particular para execução das tarefas. Assim, o AG é empregado como técnica de otimização, escolhendo um bom caminho, ou sequência de eventos, entre todos habilitados pelo controle supervisório. Portanto, garante-se que a sequência encontrada no escalonamento via AG seja uma boa solução, senão a ótima, além de ser factível, "legal" ou possível de ser realizada pela planta. Dessa forma, é possível combinar a natureza ótima da TCS com algoritmos e métodos de otimização para o escalonamento de sistemas de manufatura automatizados (Oliveira *et al*., 2013).

Assim, este trabalho propõe uma adaptação da metodologia de Pena *et al*. (2015) e emprega algoritmos genéticos como metaheurística para obtenção de soluções ótimas de escalonamento, combinado com a teoria do controle supervisório (TCS) na geração automática de sequências de operações na produção de peças em um sistema didático de manufatura. Para modelagem do sistema utiliza-se a abordagem CSML. Por questão de representação do parâmetro tempo utilizou-se a abordagem de máquinas de estados finitos com parâmetros. A metodologia empregada para o desenvolvimento do trabalho divide-se em três etapas, descritas a seguir.

- **Etapa 1 – Modelagem da planta e síntese de supervisores.** Nesta etapa são modelados os subsistemas da planta juntamente com as especificações para realizar a síntese dos supervisores utilizando o *software* Supremica (Akesson *et al*., 2006). O objetivo dessa etapa é garantir que o SED apresente o comportamento desejado ao propor um sistema de controle (supervisores) de modo a atender requisitos de funcionamento lógico e segurança (especificações operacionais).
- **Etapa 2 – Aplicação do algoritmo genético (AG) ao problema do escalonamento em plantas representadas por SED, em particular, segundo a TCS.** Nessa etapa, primeiramente, é escolhida a representação do cromossomo para o AG. Então, é implementado o AG para a busca da melhor sequência de produção do sistema em conjunto com os modelos obtidos na etapa 1 para a descrição das restrições do sistema. Desse modo, encontra-se uma sequência de eventos que atenda a um determinado plano de produção, ao mesmo tempo em que especificações para o funcionamento da planta são atendidas.

4.2 Validação da Metodologia: estudo de caso

Para a validação da metodologia proposta foi escolhido o seguinte estudo de caso: sistema didático de manufatura que se encontra no Laboratório de Robótica do curso de Engenharia Elétrica da Universidade Estadual do Oeste do Paraná – Unioeste, campus de Foz do Iguaçu, Paraná. Ele é composto pelos seguintes equipamentos:

- Servo Robô 5250 da empresa Lab-Volt;
- Um carrossel rotativo;
- Dois magazines (*gravity feeders*) ou *buffers* para a alimentação de peças no sistema e funcionam como um depósito de peças. Eles possuem dois contatos, sendo um normalmente aberto e outro, normalmente fechado. Os contatos normalmente abertos mantêm desligada uma carga até o momento em que a chave de fim de curso detecta peças. Já os contatos normalmente fechados permanecem conduzindo corrente para a carga até o momento em que a chave de fim de curso detecta peças, neste momento eles abrem e não permitem a passagem de corrente.

As Figuras 4.1 e 4.2 apresentam, respectivamente, o servo robô e a mesa giratória com os magazines de peças.

Figura 4.1 - Servo Robô 5250.

Figura 4.2 - *Buffers* e Mesa Giratória.

4.2.1 Descrição do funcionamento da célula de manufatura

O funcionamento da célula consiste na produção de duas diferentes peças, chamadas neste trabalho de peça 1 e peça 2.

Em cada um dos *buffers* ou magazines guardam-se diferentes materiais. O *buffer* 1, chamado de B_1, armazena o material para a produção da peça 1 e neste trabalho é chamado de material 1. O mesmo ocorre para o *buffer* 2 (chamado de B_2), que guarda o material para a produção da peça 2 e neste caso chamado de material 2. O servo robô, chamado de R, retira o material dos armazéns e os deposita no carrossel rotativo, chamado de G.

Para a mesa giratória, foi admitido que ela possui três posições conhecidas como P_1, P_2 e P_3. Na posição P_1 ocorre o depósito dos materiais. Na posição P_2 encontra-se a estação de manufatura 1, chamada de M_1, que realiza o processo 1 para a fabricação da peça 1 e o processo 2 para a fabricação da peça 2. O material 2, após o processo 2 para a fabricação da peça 2, já se encontra pronto e passa a ser chamado de peça 2. Esta deve ser retirada pelo servo robô e depositada na posição correta de depósito de peça 2, chamado de D_2. Porém, o material 1 após o processo 1 para a fabricação da peça 1 deve seguir para a estação de manufatura 2, chamada de M_2, que ocorre na posição P_3 e após esse processo, o servo robô deve retirar a peça 1 e a depositar em seu depósito adequado, chamado de D_1.

Então, para a fabricação da peça tipo 1, o procedimento é o seguinte:

1 – O material 1 para a produção da peça tipo 1, entra no sistema pelo *buffer* 1 (B_1);

2 – No *buffer* 1 (B_1), existe um sensor que indica a chegada de material 1 na extremidade do *buffer*;

3 – Quando o sensor do *buffer* 1 (B_1) indica a chega de material 1, o robô (R) pode retirar o material 1 do *buffer* 1 e depositá-lo na posição P_1 da mesa giratória (G);

4 – Após o depósito do material 1 na mesa giratória (G), a mesa giratória pode girar até a posição P_2, onde se encontra a estação de manufatura 1 (M_1);

5 – Quando o material 1 se encontra na posição P_2 da mesa giratória (G), ele passa pelo processo de manufatura 1 para o seu tipo de material;

6 – Após o final do processo de manufatura 1 para o material 1 que estão na posição P_2 da mesa giratória (G), a mesa giratória novamente gira levando o material 1 até a posição P_3 da mesa giratória;

7 – Na posição P_3 se encontra a estação de manufatura 2 (M_2), onde o material 1 passa pelo processo de manufatura 2;

8 – Ao término do processo de manufatura 2, o robô (R) retira o material 1 (que agora é

chamado de peça tipo 1) e o deposita no depósito 1 (D_1), encerrando assim o processo de produção da peça tipo 1.

E para a fabricação da peça tipo 2, o procedimento é o seguinte:

1 – O material 2 para a produção da peça tipo 2, entra no sistema pelo *buffer* 2 (B_2);

2 – No *buffer* 2 (B_2), existe um sensor que indica a chegada e a saída de material 2 na extremidade do *buffer*;

3 – Quando o sensor do *buffer* 1 (B_2) indica a chega de material 2, o robô (R) pode retirar o material 2 do *buffer* 2 e depositá-lo na posição P_1 da mesa giratória (G);

4 – Após o depósito do material 2 na mesa giratória (G), a mesa giratória pode girar até a posição P_2, onde se encontra a estação de manufatura 1 (M_1);

5 – Quando o material 2 se encontra na posição P_2 da mesa giratória (G), ele passa pelo processo de manufatura 1 para o seu tipo de material;

6 – Ao término do processo de manufatura 1, o robô (R) retira o material 2(que agora é chamado de peça tipo 2) e o deposita no depósito 2 (D_2), encerrando assim o processo de produção da peça tipo 2.

Na Figura 4.3 apresenta-se o diagrama de funcionamento da célula de manufatura. É importante salientar que se considera que os processos M_1 e M_2 são executados diretamente sobre as peças depositadas na mesa giratória, nas respectivas posições P_2 e P_3.

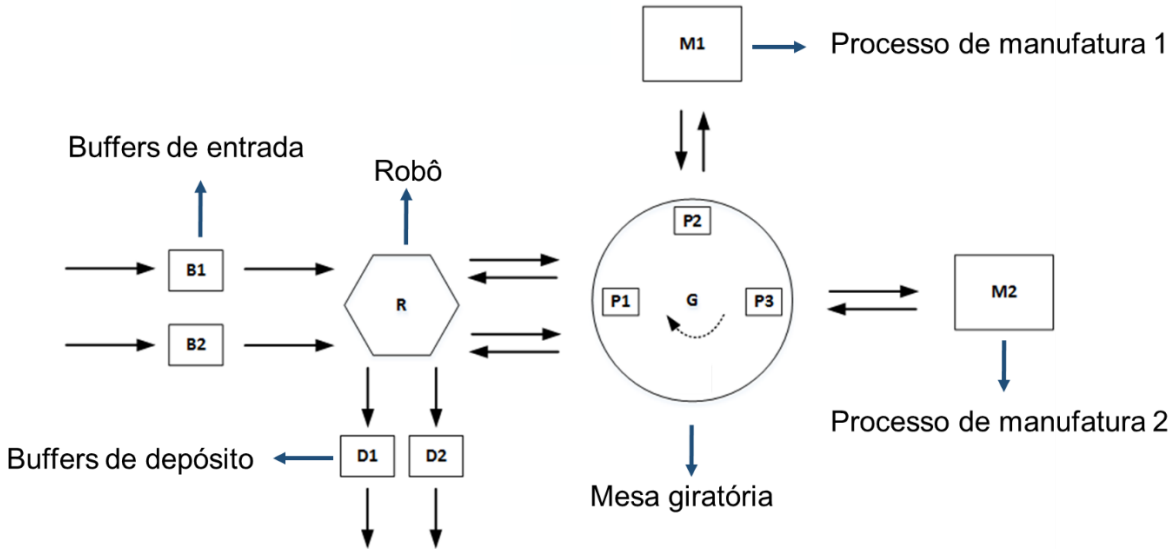

Figura 4.3 - Diagrama de Funcionamento da Célula de Manufatura.

4.2.2 Etapa 1 – Modelagem da planta e síntese dos supervisores

Para facilitar a modelagem de grandes sistemas, geralmente realiza-se uma abordagem local, em vez de uma abordagem global, com objetivo de evitar o problema de explosão de estados. Além de facilitar a criação do modelo, a abordagem local traz como vantagem de quando forem necessárias modificações e alterações em um subsistema ou restrição, somente ocorrerá mudanças no modelo específico correspondente.

Para a modelagem foi utilizado o *software* Supremica. Como apresenta Akesson (2003), o Supremica é uma tentativa de construção de um ambiente integrado capaz de resolver verificações de supervisores de grande escala e problemas com a sínteses destes. Nas modelagens desenvolvidas, o estado inicial do autômato é determinado por uma flecha, os estados marcados por um duplo círculo, eventos controláveis são indicados através de flechas tracejadas com uma reta e os eventos não-controláveis por flechas apenas tracejadas.

- **Modelagem da Planta**

Seguindo a abordagem modular do CSML, para cada subsistema associado à planta é modelado um autômato. O modelo global do comportamento discreto do sistema é dado pelo produto síncrono de cada um dos autômatos modulares. A planta global do sistema está composta pelos seguintes subsistemas: robô (R), carrossel rotativo (G), processos de manufatura 1 e 2 (M_1 e M_2) e *buffers* ou magazines de entrada (B_1 e B_2).

a) Subsistema Robô (R)

O funcionamento do servo robô consiste em pegar os materiais 1 e 2 dos *buffers* e depositá-los na mesa giratória, bem como pegá-las peças 1 e 2 e depositá-las em seus respectivos depósitos. As Tabelas 4.1 e 4.2 apresentam os estados e eventos que foram utilizados para a modelagem do subsistema.

Tabela 4.1 - Estados do autômato do subsistema Robô (R).

Estado	Nome do estado
Parado	1
Com material 1	2
Com material 2	4
Com peça 1	3
Com peça 2	5

Tabela 4.2 - Eventos do autômato do subsistema Robô (R).

Evento	Classificação	Nome
Pegar material 1 em B1 e depositá-lo em G	Controlável	E_1
Término de depósito do material 1 em G	Não-controlável	E_2
Pegar material 2 em B2 e depositá-lo em G	Controlável	E_3
Término do depósito do material 2 em G	Não-controlável	E_4
Pegar peça 1 e depositá-la em D1	Controlável	E_5
Término do depósito da peça 1 em D1	Não-controlável	E_6
Pegar peça 2 e depositá-la em D2	Controlável	E_7
Término do depósito da peça 2 em D2	Não-controlável	E_8

A Figura 4.4 apresenta o autômato do Robô (R), com quatro possíveis trajetórias: pegar material do tipo 1 no *Buffer* 1 (B_1) e depositá-lo na Mesa Giratória (G); pegar material do tipo 2 no *Buffer* 2 (B_2) e depositá-lo na Mesa Giratória (G); retirar peças do tipo 2 da posição P_2 da mesa e depositar em D_2; retirar peças do tipo 1 da posição P_3 da mesa e depositar em D_1. O estado inicial e o estado marcado são os mesmos, consistindo no estado 1, no qual o robô está parado.

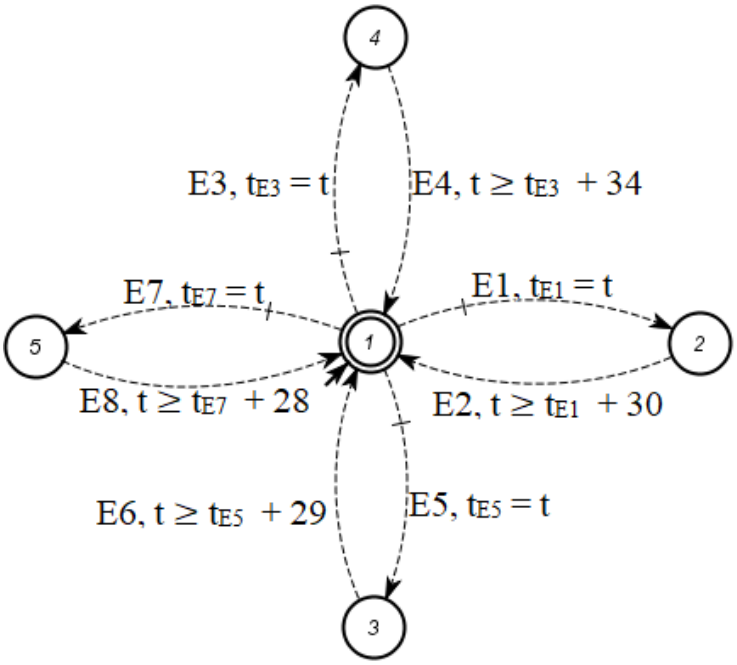

Figura 4.4 - Autômato que representa o Robô (R).

Na Figura 4.4, apresentam-se as guardas associadas ao autômato como forma de representar o parâmetro de tempo. Esse parâmetro representa o tempo que foi determinado para a execução dos pares de eventos controláveis seguidos dos eventos não-controláveis, isto é, esse parâmetro apresenta a duração de tempo em que o conjunto de eventos (controlável e não-controlável) demoram para ocorrer. Por exemplo, a ação de pegar o material 1 no *buffer* 1 (e-

vento controlável E_1) e depositar na mesa giratória (evento não-controlável E_2) demora 30 unidades de tempo. Na Tabela 4.3 apresentam-se os valores de duração de ocorrência medidos em unidades de tempo (u.t.) associados as guardas presentes no subsistema robô. Esses valores foram aferidos da planta didática de manufatura, onde uma unidade de tempo equivale a um segundo.

Tabela 4.3 - Parâmetros de tempo associados as guardas do subsistema Robô (R).

Eventos da guarda	**Duração (u.t)**
$E_1 – E_2$	30
$E_3 – E_4$	34
$E_5 – E_6$	29
$E_7 – E_8$	28

b) Subsistema Mesa Giratória (G)

O funcionamento do carrossel rotativo ou mesa giratória consiste apenas em girar e levar os materiais para outras posições. Considera-se que a mesa gira somente no sentido horário. As Tabelas 4.4 e 4.5 apresentam os estados e eventos que foram utilizados para a modelagem do sistema.

Tabela 4.4 - Estados do autômato do subsistema Mesa Giratória (G).

Estado	**Nome**
Está girando	2
Está parado	1

Tabela 4.5 - Eventos do autômato do subsistema Mesa Giratória (G).

Evento	**Classificação**	**Nome**
Girar o carrossel	Controlável	E_9
Terminou o giro	Não-controlável	E_{10}

O estado marcado, ou seja, aquele que indica o fim do processo, é o estado parado. O estado inicial também é o estado 1. Na Figura 4.5 é mostrado o autômato que representa a mesa giratória, indicando se a mesa está ou não em rotação.

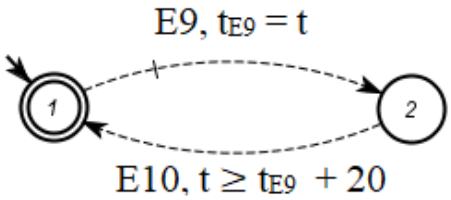

Figura 4.5 - Autômato que representa a Mesa Giratória (G).

Na Figura 4.5, também é mostrada a guarda associada ao parâmetro. Esse parâmetro determina a duração de tempo em que o conjunto de eventos (controlável e não-controlável) demoram para ocorrer, ou seja, o tempo para a mesa iniciar e finalizar um giro. Na Tabela 4.6 apresenta-se o valor de duração de ocorrência medido em unidades de tempo (u.t.) associado a guarda presente no subsistema mesa giratória (G).

Tabela 4.6 - Parâmetros de tempo associados a guarda do subsistema Mesa Giratória (G).

Eventos da guarda	**Duração (u.t)**
$E_9 - E_{10}$	20

c) Subsistema *Buffer* 1 (B_1)

O funcionamento do Magazine 1 ou *Buffer* 1 (B_1) consiste em indicar a presença ou não do material 1. As Tabelas 4.7 e 4.8 apresentam os estados e eventos que foram utilizados para a modelagem do subsistema.

Tabela 4.7- Estados do autômato do subsistema do *Buffer* 1 (B_1).

Estado	**Nome**
Sem material 1	1
Com material 1	2

Tabela 4.8 - Eventos do autômato do subsistema *Buffer* 1 (B_1).

Evento	**Classificação**	**Nome**
Sensor indica presença de material 1	Não-controlável	E_{11}
Sensor indica ausência de material 1	Não-controlável	E_{12}

O estado marcado é o estado sem material 1. O estado inicial também é o estado 1. Na Figura 4.6 é mostrado o modelo do *buffer* B_1, indicando a presença ou não de material 1. Esse autômato não possui parâmetros, e, consequentemente, guardas, em virtude de ser composto apenas por eventos não-controláveis que apenas são respostas de um sensor.

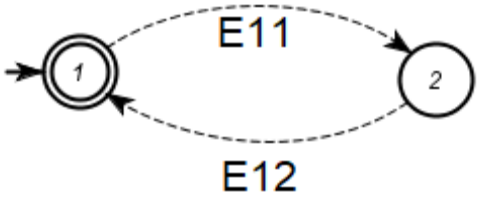

Figura 4.6 - Autômato que representa o *Buffer* 1 (B_1).

d) Subsistema *Buffer* 2 (B_2)

O funcionamento do Magazine 2 ou *Buffer* 2 (B_2) consiste em indicar a presença ou não do material 2, e é modelado de maneira similar ao subsistema *Buffer* 1. As Tabelas 4.9 e 4.10 apresentam os estados e eventos que foram utilizados para a modelagem do subsistema.

Tabela 4.9 - Estados do autômato do subsistema *Buffer* 2 (B_2).

Estado	Nome
Sem material 2	1
Com material 2	2

Tabela 4.10 - Eventos do autômato do subsistema *Buffer* 2 (B_2).

Evento	Classificação	Nome
Sensor indica presença de material 2	Não-controlável	E_{13}
Sensor indica ausência de material 2	Não-controlável	E_{14}

O estado marcado é o estado sem material 2. O estado inicial é o estado sem material 2, correspondendo ao estado 1. Na Figura 4.7 é mostrado o modelo do *Buffer* B_2, indicando a presença ou não de material 2. Esse autômato não apresenta nem parâmetros ou guardas em virtude de ser composto apenas por eventos não-controláveis que apenas são respostas de um sensor.

Figura 4.7 - Autômato que representa o *Buffer* 2 (B_2).

e) Subsistema Processo de Manufatura 1 (M_1)

O funcionamento do subsistema Processo de Manufatura 1(M_1) consiste em realizar um processo de manufatura no material 1 e outro processo de manufatura no material 2. As Tabelas 4.11 e 4.12 apresentam os estados e eventos que foram utilizados para a modelagem do subsistema.

Tabela 4.11 - Estados do autômato do subsistema Processo de Manufatura 1 (M_1).

Estado	Nome
Nada acontece	1
Realizando processo de manufatura no material 1	2
Realizando processo de manufatura no material 2	3

Tabela 4.12 - Eventos do autômato do subsistema Processo de Manufatura 1 (M_1).

Evento	Classificação	Nome
Iniciar o processo de manufatura no material 1	Controlável	E_{15}
Término do processo de manufatura no material 1	Não-controlável	E_{16}
Iniciar o processo de manufatura no material 2	Controlável	E_{17}
Término do processo de manufatura no material 2	Não-controlável	E_{18}

O estado marcado, ou seja, aquele que indica o fim do processo, é o estado onde nada acontece. Os estados inicial e final correspondem ao mesmo estado. Na Figura 4.8 é mostrada o autômato que representa o Processo de Manufatura M_1.

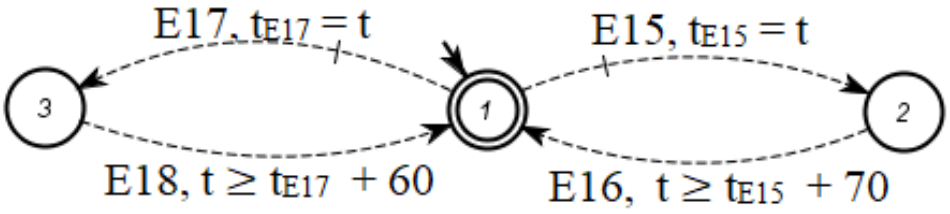

Figura 4.8 - Autômato do subsistema M_1.

Na Figura 4.8, encontram-se também representadas as duas guardas associadas aos parâmetros, indicando o tempo de processamento do material 1 e material 2, respectivamente. Na Tabela 4.13 apresentam-se os valores de duração de ocorrência medidos em unidades de tempo (u.t.) associados às guardas presentes no subsistema processo de manufatura 1 (M1).

Tabela 4.13 - Parâmetros de tempo associados as guardas do subsistema Processo de Manufatura 1 (M1).

Eventos da guarda	Duração (u.t)
$E_{15} - E_{16}$	80
$E_{17} - E_{18}$	70

f) Subsistema Processo de Manufatura 2 (M_2)

O funcionamento do Processo de Manufatura 2 consiste em realizar um processo de manufatura no material 2. As Tabelas 4.14 e 4.15 apresentam os estados e eventos que foram utilizados para a modelagem do subsistema.

Tabela 4.14 - Estados do autômato do subsistema Processo de Manufatura 2 (M_2).

Estado	Nome
Nada acontece	1
Realizando processo de manufatura no material 1	2

Tabela 4.15 - Eventos do autômato do subsistema Processo de Manufatura 2 (M_2).

Evento	Classificação	Nome
Iniciar o processo de manufatura no material 1	Controlável	E_{19}
Término do processo de manufatura no material 1	Não-controlável	E_{20}

O estado marcado, ou seja, aquele que indica o fim do processo, é o estado onde nada acontece. Na Figura 4.9 é mostrado o autômato que representa o Processo de Manufatura 2 (M_2), indicando que nada está acontecendo ou se iniciou o processo de manufatura 2.

Na Figura 4.9, apresenta-se também a guarda associada ao parâmetro, e representa o tempo necessário para processamento do material 1. Na Tabela 4.16 apresenta-se o valor de duração de ocorrência medido em unidades de tempo (u.t.) associado a guarda presentes no subsistema processo de manufatura 2.

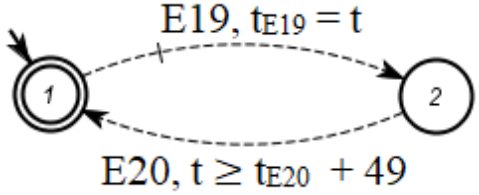

Figura 4.9 - Autômato que representa o subsistema M_2.

Tabela 4.16 - Parâmetro de tempo associado a guarda do subsistema Processo de Manufatura 2 (M_2).

Eventos da guarda	Duração (u.t)
$E_{19} - E_{20}$	49

- **Modelagem das Especificações**

O comportamento da planta, descrito pelos 6 geradores mostradas nas Figuras 4.4 a 4.9, representam o comportamento em malha-aberta do sistema. Porém, para a operação correta e segura do sistema, algumas restrições se impõem ao comportamento livre da planta. Um conjunto de especificações é definido a partir do qual são obtidas as estruturas de supervisão que irão garantir que a planta funcione em malha-fechada de acordo com os requisitos desejados. Alguns cuidados e considerações devem ser feitas:

- Cada restrição diz respeito a um sistema subconjunto de subsistemas;
- Às vezes, as restrições indicam subsequências de eventos do sistema que não podem ser alteradas;
- Não é necessário considerar o comportamento global do sistema.

Na construção do autômato que representa uma especificação/restrição, se todos os seus estados estão marcados não há alteração na linguagem marcada do sistema, mantendo-se, assim, as propriedades da planta. Tal característica é decorrente do produto síncrono entre o autômato do subsistema e da especificação, empregado no processo de síntese dos supervisores na TCS (Ramadge & Wonham, 1988; Cassandras & Lafortune, 2008). Posteriormente, se houver algum ou alguns estados ilegais (também chamados de proibidos), devem ser eliminados do gerador resultante da composição das restrições com a planta ou subsistema.

As especificações consideradas nesse trabalho são:

- Evitar o *underflow* (número de peças negativo) e o *overflow* (número de peças superior ao máximo) dos *Buffers* B_1 e B_2;
- Evitar que ações de manufatura sejam realizadas ou que o robô deposite ou retire peça enquanto o carrossel estiver girando;
- Impedir que peças do tipo 1 sejam usinadas no processo M_2 antes de terem sido processadas em M_1;
- Evitar o giro do carrossel quando ele estiver vazio;
- Evitar que peças do tipo 1 passem pelo processo 2 em M_1 e vice-versa.

A seguir, apresentam-se as 17 especificações modeladas para a planta didática considerada neste trabalho.

a) Especificação 1

Esta restrição envolve os subsistemas *Buffer* 1 (B_1) e Robô (R). Ela especifica que ao constar a chegada de material 1 no *Buffer* 1 (evento E_{11}), o Robô (R) poderá realizar a retirada do material 1 de B_1 (evento E_1). Na Figura 4.10 é apresentado o autômato da Especificação 1.

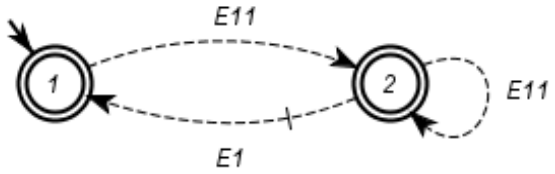

Figura 4.10 - Autômato da Especificação 1.

b) Especificação 2

Esta restrição envolve os subsistemas *Buffer* 2 (B_2) e Robô (R). Ela especifica que ao constar a chegada de material 2 no *Buffer* 2 (evento E_{13}), o Robô (R) poderá realizar a retirada do material 2 de B_2 (evento E_3). Na Figura 4.11 é apresentado o autômato da Especificação 2.

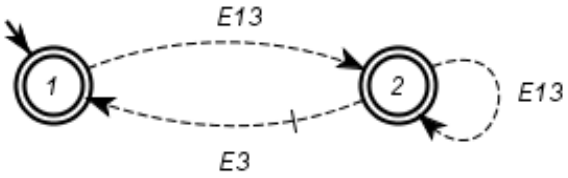

Figura 4.11 - Autômato da Especificação 2.

c) Especificação 3

Esta restrição envolve os subsistemas Processo de Manufatura 2 (M_2) e Robô (R). Ela especifica que ao constar o término do Processo de Manufatura 2 sobre a peça 1 (evento E_{20}), o Robô (R) poderá realizar a retirada da peça 1 da Mesa Giratória (G) (evento E_5). Na Figura 4.12 é apresentado o autômato da Especificação 3.

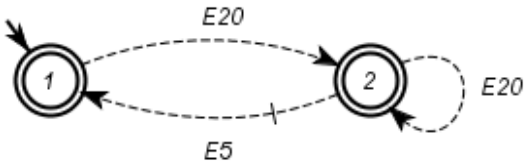

Figura 4.12 - Autômato da Especificação 3.

d) Especificação 4

Esta restrição envolve os subsistemas Processo de Manufatura 1 (M_1) e Robô (R). Ela especifica que ao constar ao término do Processo de Manufatura 1 sobre a peça 2 (evento E_{18}), o Robô (R) poderá realizar a retirada da peça 2 da Mesa Giratória (G) (evento E_7). Na Figura 4.13 é apresentado o autômato da Especificação 4.

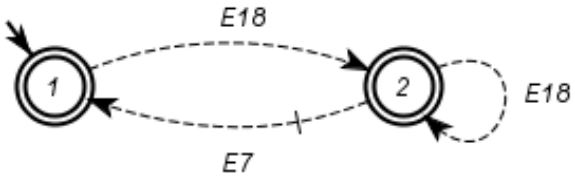

Figura 4.13 - Autômato da Especificação 4.

e) Especificação 5

Esta restrição envolve os subsistemas Mesa Giratória (G) e Robô (R). Ela impede o *over-flow* em G: toda vez que ocorrer o depósito de um material (eventos E_1 e E_3), G deverá girar para que possa ocorrer outro depósito de material (evento E_{19}). Na Figura 4.14 é apresentado o autômato da Especificação 5.

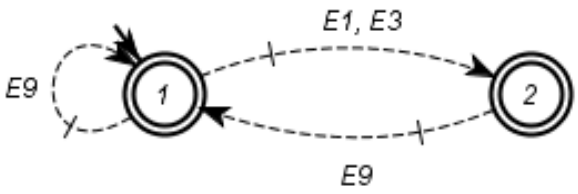

Figura 4.14 - Autômato da Especificação 5.

f) Especificação 6

Esta restrição envolve os subsistemas Mesa Giratória (G) e Processo de Manufatura 2 (M_2). Ela impede o giro de G durante o Processo de Manufatura 2: toda vez que se iniciar o Processo de Manufatura 2 (evento E_{19}), deve-se aguardar o término desse processo (evento E_{20}) para que G execute o giro (evento E_9). Na Figura 4.15 é apresentado o autômato da Especificação 6.

Figura 4.15 - Autômato da Especificação 6.

g) Especificação 7

Esta restrição envolve os subsistemas Mesa Giratória (G) e Processo de Manufatura 1 (M_1). Ela impede o giro de G durante o Processo de Manufatura 1 para a peça 1: toda vez que se iniciar M_1 para peça 1 (evento E_{15}), deve-se aguardar o término desse processo (evento E_{16}) para que G execute o giro (evento E_9). Na Figura 4.16 é apresentado o autômato da Especificação 7.

Figura 4.16 - Autômato da Especificação 7.

h) Especificação 8

Esta restrição envolve os subsistemas mesa giratória (G) e Processo de Manufatura 1 (M_1). Ela impede o giro de G durante o Processo de Manufatura 1 para a peça 2: toda vez que se iniciar o M_1 para peça 2 (evento E_{17}), deve-se aguardar o término desse processo (evento E_{18}) para que G execute o giro (evento E_9) Na Figura 4.17 é apresentado o autômato da Especificação 8.

Figura 4.17- Autômato da Especificação 8.

i) Especificação 9

Esta restrição envolve os subsistemas mesa giratória (G), o Robô (R) e Processo de Manufatura 1 (M_2). Essa especificação relaciona a posição da peça 2 em G (evento E_9) com o Processo de Manufatura 1 da peça 2 (evento E_{17}). Para que essa relação ocorra, a restrição também depende do Robô (evento E_4). Na Figura 4.18 é apresentado o autômato da Especificação 9.

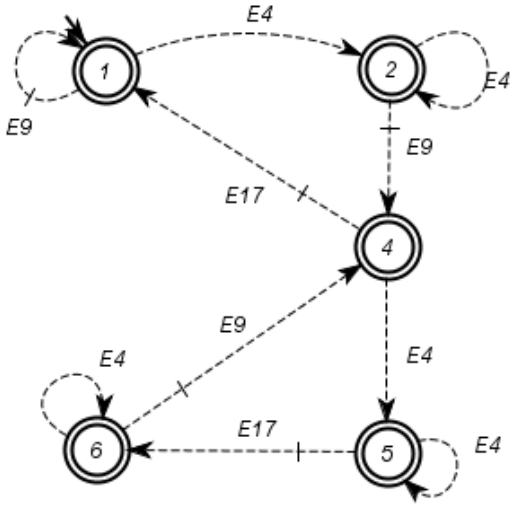

Figura 4.18 - Autômato da Especificação 9.

j) Especificação 10

Esta restrição envolve os subsistemas mesa giratória (G) e o Robô (R). Ela impede o giro da mesa durante a retirada da peça 2 (evento E_7), deve-se aguardar o término desse processo (evento E_8) para que G execute o giro (evento E_9) Na Figura 4.19 é apresentado o autômato da Especificação 10.

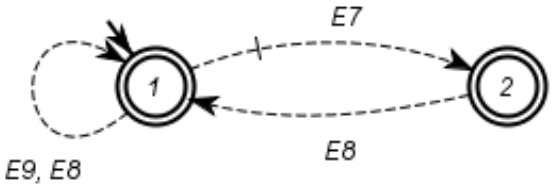

Figura 4.19 - Autômato da Especificação 10.

k) Especificação 11

Esta restrição envolve os subsistemas mesa giratória (G), o Robô (R) e Processo de Manufatura 1 (M_1). Essa especificação relaciona a posição da peça 1 na mesa giratória (evento E_9) com o Processo de Manufatura 1 da peça 1 (evento E_{15}). Para que essa relação ocorra, a restrição também depende do Robô (evento E_2). Na Figura 4.20 é apresentado o autômato da Especificação 11.

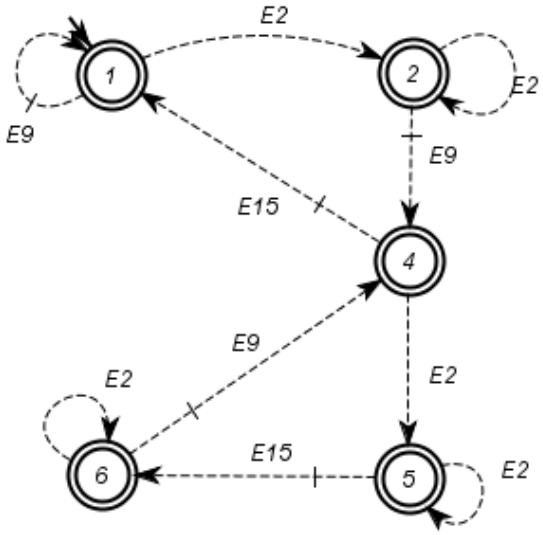

Figura 4.20 - Autômato da Especificação 11.

l) Especificação 12

Esta restrição envolve os subsistemas mesa giratória (G) e o Robô (R). Ela impede o giro da mesa durante a retirada da peça 1 (evento E_5), deve-se aguardar o término desse processo (evento E_6) para que G execute o giro (evento E_9) Na Figura 4.21 é apresentado o autômato da Especificação 12.

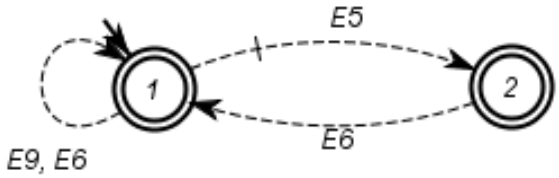

Figura 4.21 - Autômato da Especificação 12.

m) Especificação 13

Esta restrição envolve os subsistemas mesa giratória (G), Processos de Manufatura 1 e 2 (M_1 e M_2) e o Robô (R). Ela impede que inicie a retirada da peça 1 (evento E_5), a retirada da peça 2 (evento E_7), o depósito do material 1 (evento E_1), o depósito do material 2 (evento E_3), o processo de manufatura 1 e 2 (eventos E_{15} e E_{17}) enquanto a mesa giratória executa o giro (eventos E_9 e E_{10}). Na Figura 4.22 é apresentado o autômato da Especificação 13.

Figura 4.22 - Autômato da Especificação 13.

n) Especificação 14

Esta restrição envolve os subsistemas mesa giratória (G), Processo de Manufatura 1 (M_1) e o Robô (R). Ela impede que a mesa gire ao vazio (evento E_9), pois sempre deve haver algo na mesa (eventos E_2, E_4 e E_{16}). Na Figura 4.23 é apresentado o autômato da Especificação 14.

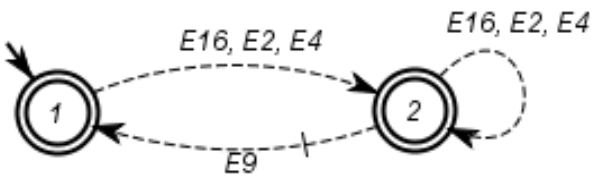

Figura 4.23 - Autômato da Especificação 14.

o) Especificação 15

Esta restrição envolve os subsistemas mesa giratória (G) e os Processos de Manufatura 1 e 2 (M_1 e M_2). Essa especificação relaciona a ordem de processamento da peça 1, onde sempre deve haver um giro da mesa giratória (evento E_9) entre o Processo de Manufatura 1 (evento E_{15}) e o Processo de Manufatura 2 (evento E_{19}). Na Figura 4.24 é apresentado o autômato da Especificação 15.

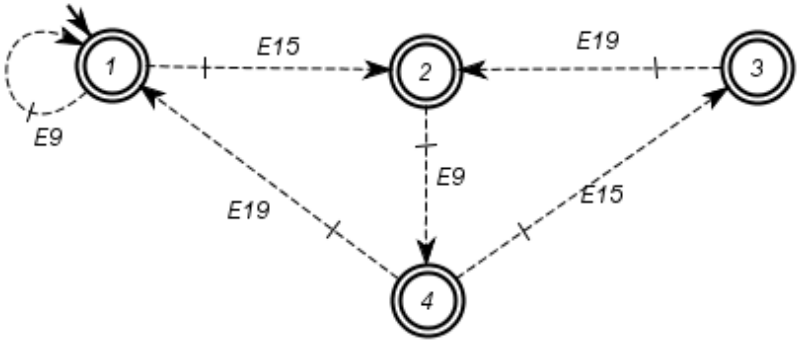

Figura 4.24 - Autômato da Especificação 15.

p) Especificação 16

Esta restrição envolve os subsistemas mesa giratória (G) e o Processo de Manufatura 1 para a peça 2 (M_1). Essa especificação impede que após o término do Processo de Manufatura 1 para a peça 2 (evento E_{18}), a mesa giratória gire (evento E_9) antes que o Robô retire a peça 2 (evento E_7). Na Figura 4.25 é apresentado o autômato da Especificação 16.

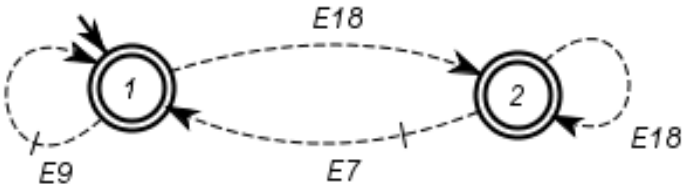

Figura 4.25 - Autômato da Especificação 16.

q) Especificação 17

Esta restrição envolve os subsistemas mesa giratória (G) e o Processo de Manufatura 2 (M_2). Essa especificação impede que após o término do Processo de Manufatura 2 para a peça 1 (evento E_{20}) G gire (evento E_9) antes que o Robô retire a peça 1 (ao evento E_5). Na Figura 4.26 é apresentado o autômato da Especificação 17.

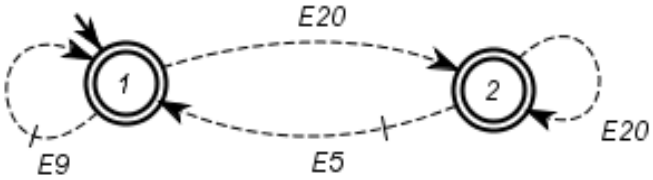

Figura 4.26 - Autômato da Especificação 17.

O *software* Supremica permite a síntese e construção dos supervisores e ainda fornece dados desses supervisores, como nome e número de estados, eventos e números e nomes das transições. Para o estudo de caso proposto foi construída tanto a abordagem monolítica (um supervisor global) como a abordagem CSML (vários supervisores modulares). A ferramenta Supremica também permite realizar a redução dos supervisores.

Na abordagem monolítica, o processo de síntese é realizado sobre o modelo completo da planta e o modelo completo das especificações. Para isso, primeiro são obtidos, pela composição síncrona, o autômato que representa a planta e outro para a representação das especificações. O teste de controlabilidade (apresentado na seção 2.2.2) das especificações é realizado também pela ferramenta. Ao final, a ferramenta encontra o supervisor monolítico, que pode ser reduzido para diminuir a quantidade de estados. A Tabela 4.17 apresenta os dados dessa abordagem.

Tabela 4.17 - Dados do supervisor monolítico construído.

Autômato	**Estados**	**Eventos**	**Transições**
Planta	240	20	2144
Especificações	34818	18	375800
Supervisor monolítico	14640	20	88432
Supervisor monolítico reduzido	10480	20	63872

Para a abordagem modular local, o processo de obtenção dos supervisores envolve os seguintes passos. Para cada subsistema, encontra-se o modelo local ou autômato G_k que o representa, onde k representa o índice do subsistema em questão. Para cada uma das especificações E_x, onde x indica o índice da especificação, encontra-se a planta local G_{locx}. A planta local G_{locx} é obtida pelo produto síncrono de todas as plantas G_k envolvidas com a especificação E_x. Para obtenção do supervisor modular, é necessário, primeiro, encontrar a linguagem alvo ou especificação local E_{locx}, obtida pelo produto síncrono entre G_{locx} e E_x. Finalmente, pelo processo de síntese da TCS, encontra-se o supervisor modular local S_x. Novamente, a ferramenta

Supremica realiza o teste de controlabilidade das especificações e o não-conflito dos supervisores modulares locais. A Tabela 4.18 apresenta os dados do processo de síntese dos supervisores modulares, contendo o número estados e transições em cada um dos autômatos.

Nota-se a diferença pelo número de estados e de transições entre o supervisor monolítico e pelos supervisores construídos pela CSML, e tendo em vista, além do esforço computacional, a implementação dentro do ambiente industrial se torna bem mais fácil.

A partir da construção dos supervisores do sistema, então aplica-se o algoritmo genético (AG) ao problema do escalonamento na planta que está representada por SED, em particular, segundo a TCS.

Tabela 4.18 - Dados do processo de síntese dos supervisores modulares.

Número da Especificação	**Especificação E_x**	**Planta local G_{locx}**	**Especificação local K_{locx}**	**Supervisor reduzido S_x**
1	02/03[3]	10/26	20/50	20/50
2	02/03	10/26	20/50	20/50
3	02/03	10/26	20/50	20/50
4	02/03	15/44	30/85	30/85
5	02/04	10/26	20/48	20/48
6	02/04	04/08	04/07	04/07
7	02/04	06/14	06/13	06/13
8	02/04	06/14	06/13	06/13
9	05/10	30/118	150/530	120/422
10	02/04	10/26	10/25	10/25
11	05/10	30/118	150/530	120/422
12	02/04	10/26	10/25	10/25
13	02/07	60/296	60/237	60/237
14	02/07	30/118	20/221	20/221
15	04/06	12/48	48/128	48/128
16	02/04	15/44	60/215	60/215
17	02/04	10/26	40/130	40/130

4.2.3 Etapa 2 – Aplicação do algoritmo genético (AG)

Conforme a metodologia proposta, primeiramente, escolhe-se uma representação para o cromossomo no algoritmo genético. Para esta planta, foram escolhidas três representações, sendo duas representações de abordagem direta e uma da abordagem indireta. As representações escolhidas foram: abordagem direta – representação baseada em operação, representação

[3] Indica o número de estados / transições do respectivo autômato.

de chaves aleatórias; e abordagem indireta – representação baseada em regras de prioridade. Os algoritmos desenvolvidos são baseados no pseudocódigo apresentado na seção 3.1.

Nessas representações, conforme o estudo de caso, considera-se uma tarefa ou trabalho, as operações em determinada sequência que levam a produção de uma peça, seja ela do tipo 1 ou 2. No caso da utilização da abordagem TCS, a fabricação de qualquer produto pode ser modelada por uma sequência específica de eventos (Pena *et al.*, 2015). Dentro dessa sequência de eventos, existem subsequências ou subcadeias que não podem ser alteradas, chamadas de subsequências invariantes. Por isso, a fabricação de um produto pode ser modelada como uma sequência de subsequências de eventos, relação esta apresentada na expressão 4.1:

$$\varepsilon = \{\varepsilon_1 \varepsilon_2 \dots \varepsilon_n\} \tag{4.1}$$

onde ε_i corresponde ao evento controlável ou não-controlável que compõe a subsequência invariante ε que precisa ser executada sequencialmente para que o produto seja fabricado corretamente. As subsequências invariantes estão diretamente relacionadas com a modelagem do sistema. O sistema em questão foi modelado utilizando autômatos e para representação do parâmetro tempo, utilizou-se guardas, onde cada ação, representada por um evento controlável, tem uma reação (evento não-controlável) correspondente e as respectivas guardas associadas representam condições temporais para que os eventos de reação ocorram. Portanto, cada sequência de ação (evento controlável) e reação (evento não-controlável) corresponde a uma subsequência invariante e cada subsequência invariante corresponde a uma operação a ser realizada.

Para o estudo de caso, as sequências invariantes podem ser obtidas definindo as etapas para a produção de peças tipo 1 e 2, ou seja:

- **Peça 1:** sensor detecta presença do material 1 (evento não-controlável E_{11}); robô pega material 1 (evento controlável E_1) e o deposita na mesa giratória (evento não-controlável E_2); sensor indica ausência de material 1 (evento não-controlável E_{12}); mesa gira (evento controlável E_9 – evento não-controlável E_{10}); o processo de manufatura 1 ocorre (evento controlável E_{15}); processo de manufatura 1 termina (evento não-controlável E_{16}); a mesa gira novamente (evento controlável E_9 – evento não-controlável E_{10}); processo de manufatura 2 começa (evento controlável E_{19}); processo de manufatura 2 termina (evento não-controlável E_{20}); robô retira a peça 1 da mesa giratória (evento controlável E_5) e a deposita no buffer de depósito 1 (evento não-controlável E_6);

- **Peça 2:** sensor detecta presença do material 2 (evento não-controlável E_{13}), robô pega material 2 (evento controlável E_3) e o deposita na mesa giratória (evento não-controlável E_4), sensor indica ausência de material 2 (evento não-controlável E_{14}), mesa gira (evento controlável E_9 – evento não-controlável E_{10}), o processo de manufatura 1 ocorre (evento controlável E_{17} – evento não-controlável E_{18}), o robô retira a peça 2 da mesa giratória (evento controlável E_7) e a deposita no buffer de depósito 2 (evento não-controlável E_8).

Traduz-se então, essa produção nas seguintes sequências de eventos apresentadas na expressão 4.2 para a peça tipo 1 e na expressão 4.3 para a produção da peça tipo 2.

$$P_1 = \{E_{11} \quad E_1\ E_2 \quad E_{12} \quad E_9\ E_{10} \quad E_{15}\ E_{16} \quad E_9\ E_{10} \quad E_{19}\ E_{20} \quad E_5\ E_6\} \quad (4.2)$$

$$P_2 = \{E_{13} \quad E_3\ E_4 \quad E_{14} \quad E_9\ E_{10} \quad E_{17}\ E_{18} \quad E_7\ E_8\} \quad (4.3)$$

Nota-se que, excluindo o primeiro evento (E_{11}: chegada de material 1; e E_{13}: chegada de material 2) e o quarto evento (E_{12}: ausência de material 1; e E_{14}: ausência de material 2), temos subsequências que relacionam as sequências de ações e reações agora chamadas de sequências invariantes. Por exemplo, o robô sempre ao pegar o material 2 no *buffer* 2 tem que sempre depositá-lo na mesa giratória (E_3 – E_4) ou sempre que se iniciar o processo de manufatura 2, esse processo sempre acaba (E_{19} – E_{20}). As subsequências invariantes encontradas neste sistema são:

- Robô pega material 1 e deposita na esteira: eventos E_1 E_2, sendo essa subsequência invariante chamada de ε_1;
- Giro da mesa: eventos E_9 E_{10}, sendo essa subsequência invariante chamada de ε_2;
- Processo de manufatura M_1 da peça 1: eventos E_{15} E_{16}, sendo essa subsequência invariante chamada de ε_3;
- Processo de manufatura M_2: eventos E_{19} E_{20}, sendo essa subsequência invariante chamada de ε_4;
- Remoção da peça 1: eventos E_5 E_6, sendo essa subsequência invariante chamada de ε_5;
- Robô pega peça 2 e deposita na esteira: eventos E_3 E_4, sendo essa subsequência invariante chamada de ε_6;
- Processo de manufatura M_1 da peça 2: eventos E_{17} E_{18}, sendo essa subsequência invariante chamada de ε_7;

- Remoção da peça 2: eventos E_7 E_8, sendo essa subsequência invariante chamada de ε_8.

Então, as novas sequências de eventos para a produção da peça do tipo 1 e da peça do tipo 2, utilizando as subsequências invariantes, são apresentadas em 4.4 (produção de peça tipo 1) e 4.5 (produção de peça tipo 2).

$$P_1 = \{E_{11} \quad \varepsilon_1 \quad E_{12} \quad \varepsilon_2 \quad \varepsilon_3 \quad \varepsilon_2 \quad \varepsilon_4 \quad \varepsilon_5\} \tag{4.4}$$

$$P_2 = \{E_{13} \quad \varepsilon_6 \quad E_{14} \quad \varepsilon_2 \quad \varepsilon_7 \quad \varepsilon_8\} \tag{4.5}$$

Cada subsequência invariante corresponde a uma operação a ser escalonada dentro do problema do *job-shop scheduling*. Observando as expressões 4.4 e 4.5, tanto na peça do tipo 1, quanto na peça do tipo 2, o primeiro evento (E_{11} ou E_{13}) indica a chegada de peças nos *buffers* e o quarto (E_{12} ou E_{14}) evento indica a saída de peças dos *buffers*. Esses eventos são não-controláveis e, de acordo com as especificações, os primeiros eventos (E_{11} e E_{13}) são condições de início do processo de produção e os eventos E_{12} e E_{14} correspondem a uma consequência de retirada do material pelo robô. Por este motivo, foi escolhido que a primeira operação corresponde a uma subsequência invariante que passa a ser a apresentada em 4.6 para peças do tipo 1 e 4.7 para peças do tipo 2.

$$\varepsilon_{11} = \{E_{11}\ \varepsilon_1\ E_{12}\} \tag{4.6}$$

$$\varepsilon_{21} = \{E_{13}\ \varepsilon_6\ E_{14}\} \tag{4.7}$$

Então, a sequência de subsequências invariantes para a produção de peça do tipo 1 passa a ser a representada na expressão 4.8 e a da produção de peça tipo 2, a da expressão 4.9. Por fim, na produção da peça 1 tem-se então 6 operações e na produção da peça 2, 4 operações.

$$P_1 = \{\varepsilon_{11} \quad \varepsilon_2 \quad \varepsilon_3 \quad \varepsilon_2 \quad \varepsilon_4 \quad \varepsilon_5\} \tag{4.8}$$

$$P_2 = \{\varepsilon_{21} \quad \varepsilon_2 \quad \varepsilon_7 \quad \varepsilon_8\} \tag{4.9}$$

No caso considerado existem alguns casos especiais de operações concorrentes ou de situações em que um par de operações representa a mesma operação. No que se diz respeito a operações concorrentes, são operações que podem ocorrer ao mesmo tempo e que não influenciam uma na outra. Por exemplo, se uma peça tipo 1 passa pelo processo de manufatura 2, pode ocorrer de uma peça qualquer estar passando pelo processo de manufatura 1. Também, neste sistema, existe o caso especial do giro da mesa, consistindo na mesa girar com mais de uma pe-

ça em cima, no qual esse movimento corresponde a uma operação para cada peça, mas é traduzido apenas como o par de eventos, uma vez que as duas operações ocorrem de maneira simultânea.

É necessário a identificação de algumas variáveis dentro do problema do escalonamento: uma tarefa é constituída das operações necessárias para a produção das peças. No caso da tarefa para produzir uma peça do tipo 1 são necessárias seis operações e a tarefa para produzir uma peça do tipo 2, são necessárias quatro operações. São identificadas quatro máquinas dentro do sistema proposto: o robô (M_1), a mesa giratória (M_2), o processo de manufatura 1 (M_3) e processo de manufatura 2 (M_4). Nas Tabelas 4.19 e 4.20 são apresentadas as relações entre máquinas, sequência de operações, subsequências invariantes, subsequências de eventos e tempo de execuções das operações para a peça do tipo 1 e para a peça do tipo 2, respectivamente. Os tempos das operações foram idealizados durante o processo de modelagem do sistema e encontram-se quantificados em unidades de tempo (u.t.).

Tabela 4.19 - Relação entre sequência de operações e de máquinas e subsequências invariantes e de eventos e tempos de execuções das operações para a peça tipo 1.

Sequência de Operações	**1a**	**2a**	**3a**	**4a**	**5a**	**6a**
Sequência de máquinas	M_1	M_2	M_3	M_2	M_4	M_1
Subsequência invariantes	ε_{11}	ε_2	ε_3	ε_2	ε_4	ε_5
Subsequência de eventos	*"E_{11} E_1 E_2 E_{12}"*	*"E_9 E_{10}"*	*"E_{15} E_{16}"*	*"E_9 E_{10}"*	*"E_{19} E_{20}"*	*"E_5 E_6"*
Tempo de execuções das operações (u.t.)	30	20	60	20	49	29

Tabela 4.20 - Relação entre as sequências de operações, as sequência de máquinas, as subsequências invariantes, as subsequências de eventos e os tempos de execuções das operações para a peça tipo 2.

Sequência de Operações	**1b**	**2b**	**3b**	**4b**
Sequência de máquinas	M_1	M_2	M_3	M_1
Subsequência invariantes	ε_{21}	ε_2	ε_7	ε_8
Subsequência de eventos	*"E_{13} E_3 E_4 E_{14}"*	*"E_9 E_{10}"*	*"E_{17} E_{18}"*	*"E_7 E_8"*
Tempo de execuções das operações (u.t.)	34	20	70	28

O problema do escalonamento de tarefas visa encontrar sequências de operações que façam o sistema alcançar uma melhor performance de acordo com o objetivo do escalonamento. Neste trabalho, buscou-se a redução do tempo de produção de lotes de peças do tipo 1 ou do tipo 2 em relação ao processamento sequencial, isto é, a função objetivo desse trabalho consiste

na minimização do tempo de produção dos lotes propostos. O processamento sequencial consiste na produção de peça por peça. No caso desse sistema, utilizando os dados das Tabelas 4.19 e 4.20, para a produção de uma peça do tipo 1 são necessário 208 unidades de tempo e da peça tipo 2, 152 unidades de tempo. Então, por exemplo, para a produção sequencial de um lotede 10 peças do tipo 1 e 10 peças do tipo 2, o tempo necessário para a produção seria 3600 unidades de tempo. Utilizou-se também a abordagem da TCS para garantir que as sequências encontradas no algoritmo implementado para a resolução do problema de escalonamento, sejam "legais" e factíveis.

Após a escolha das representações do AG, implementou-se os respectivos algoritmos no *software* Matlab, versão 2017a, em um *notebook* com a seguinte configuração: processador Intel Core i7 3ª geração 2GHz, 8GB de memória RAM e sistema operacional de 64 bits. Essa implementação objetiva encontrar o menor tempo de produção de lotes de peças.

- **Caso 1: Representação baseada em Operação**

A representação baseada em operação ocorre com a permutação de símbolos que representam as operações (uma operação é uma subsequência invariante) que ocorreram. Sendo assim, o tamanho do cromossomo corresponderá ao número de operações (genes) e obedece a expressão 4.10, ou seja, cada gene, nesta representação, corresponde a uma operação a ser escalonada e cada gene é representado por um símbolo:

$$N_{genes} = n_{pecas1} \times 6 + n_{pecas2} \times 4 \quad (4.10)$$

onde,

- n_{genes}: corresponde ao número de genes do cromossomo a ser construído;
- n_{pecas1}: corresponde ao número de peças do tipo 1;
- n_{pecas2}: corresponde ao número de peças do tipo 2.

Também cada gene, ao ser traduzido em sequência de eventos, corresponde a uma subsequência invariante. Por exemplo, utilizando o sistema apresentado para validação da metodologia, deseja-se a produção de 2 peças tipo 1 e 3 pecas tipo 2. O cromossomo terá neste caso 24 genes, pois são necessárias 24 operações para a produção dessas peças.

Um cromossomo é constituído de genes. Em um cromossomo que utiliza a representação baseada em operação, cada gene é representado por um símbolo que identifica sua tarefa e pela leitura desse cromossomo e repetição dos símbolos é identificada a operação dessa tarefa. Já u-

ma operação é associada a uma subsequência invariante e essa subsequência invariante é associada pela sua respectiva subsequência de eventos. Um exemplo de cromossomo nessa representação é apresentado na expressão 4.11, onde seus genes são representados por números naturais excluindo o número zero:

$$[\,1\ 2\ 1\ 1\ 2\ 2\ 3\ 6\ 1\ 6\ 4\ 1\ 2\ 6\ 1\ 4\ 6\ 3\ 3\ 4\ 4\ 3\ 3\ 3\,] \tag{4.11}$$

Para a fabricação de um produto, existe uma ordem de precedência das operações, ou seja, existe uma ordem de sequência das operações. Ao alterar essa ordem pode levar uma produção incorreta de uma peça, ou até mesmo a problemas no sistema. A fim de contornar problemas de precedência, utiliza-se o mesmo símbolo para representar as operações da mesma tarefa, sendo que a operação é interpretada de acordo com a ordem de ocorrência na leitura do cromossomo da esquerda para direita. Por exemplo, observando o cromossomo apresentado na expressão 4.11: o primeiro gene (símbolo 1) no cromossomo representa a primeira operação da tarefa 1, uma vez que é a primeira vez que esse símbolo ocorre na leitura do cromossomo e por isso, é a primeira operação a ser agendada. Já o terceiro gene, representa a segunda operação da tarefa 1, já que aparece pela segunda vez na leitura do cromossomo e como é o terceiro gene, é a terceira operação a ser agendada. Os símbolos escolhidos para a codificação do cromossomo foram os pertencentes ao conjunto de números naturais excluindo o número zero. Para a identificação do tipo de peça produzida, os símbolos para representar as peças de tipo 1 são codificados com números ímpares e para as peças de o tipo 2, com números pares.

Um cromossomo é considerado legal quando a sequência de eventos no qual o cromossomo é vinculado é uma sequência possível na planta e habilitada pelos supervisores. Então, para realizar a comprovação da legalidade do cromossomo, é necessário realizar a decodificação desse cromossomo de uma sequência de genes para uma sequência de eventos e, então, essa sequência de eventos deve ser submetida a uma comprovação de legalidade utilizando a modelagem do sistema pela TCS.

A decodificação do cromossomo é realizada da seguinte forma: seja o exemplo do cromossomo na expressão 4.11 e utilizando o sistema proposto e modelado na seção 4.2.1.Para a produção das peças do tipo 1 são necessárias 6 operações, então a repetição dos símbolos que identificam a tarefa (números ímpares) ocorrerá 6 vezes dentro do cromossomo. Já para a produção das peças tipo 2 são necessárias 4 operações, portanto, a repetição dos símbolos que identificam essa tarefa (números pares) ocorrerá 4 vezes.

A construção da sequência de eventos que é associada a esse cromossomo é feita a partir da leitura do cromossomo da esquerda para a direita. O primeiro gene é representado pelo símbolo 1. Este símbolo identifica a tarefa 1. A operação que é associada a tarefa 1, por ser a primeira repetição no cromossomo, identifica a primeira operação da tarefa 1. A primeira operação da tarefa 1 é associada a primeira subsequência invariante da produção da peça 1 apresentada na expressão 4.8 (ε_{11}) que corresponde a subsequência de eventos "$E_{11}\ E_1\ E_2\ E_{12}$". O segundo gene é representado pelo símbolo 2 que identifica a tarefa 2. A operação que é associada a essa tarefa é a primeira operação da tarefa 2 que é associada a primeira subsequência invariante da produção da peça 2 apresentada na expressão 4.9 (ε_{21}) que corresponde a subsequência de eventos "$E_{13}\ E_3\ E_4\ E_{14}$". Até então, a sequência de eventos que está sendo construída é descrita na expressão 4.12. O terceiro gene corresponde ao símbolo 1, identificando a tarefa 1, porém, como é a segunda repetição do símbolo 1, a operação corresponderá a segunda operação da tarefa 1 que se associa a subsequência invariante ε_2 que corresponde a "$E_9\ E_{10}$". Então, a sequência de eventos construída até o terceiro gene é descrita na expressão 4.13. A decodificação do cromossomo é feita deste modo até a leitura e decodificação do último gene. Neste sistema proposto, existe o caso especial em que a mesa gira com duas peças em cima, o que caracteriza o mesmo par de eventos ou subsequência invariante. Então, as duas operações são consideradas, porém, apenas é agendado um par de eventos na sequência.

$$Sequência\ de\ eventos = "E_{11}\ E_1\ E_2\ E_{12}\ E_{13}\ E_3\ E_4\ E_{14}" \tag{4.12}$$

$$Sequência\ de\ eventos = "E_{11}\ E_1\ E_2\ E_{12}\ E_{13}\ E_3\ E_4\ E_{14}\ E_9\ E_{10}" \tag{4.13}$$

Para realizar essa decodificação do cromossomo no algoritmo que foi implementado para o problema do escalonamento, foi desenvolvido um *script* chamado de *decoding* (no português, decodificação) que realiza essa decodificação do cromossomo para uma sequência de eventos. A partir dessa sequência de eventos decodificada é possível comprovar a legalidade do cromossomo a partir da TCS e para isso, também foi desenvolvido um *script* chamado de *controlesupervisorio*. Estes dois *scripts* encontram-se nos Apêndices A e B. O Algoritmo 4.1 apresenta o pseudocódigo para a representação baseada em operação.

Algoritmo 4.1 – Pseudocódigo proposto para o desenvolvimento do algoritmo para a representação baseada em operação.

% Geração da população inicial totalmente legal

1. **Enquanto** (cromossomo atual < tamanho da população)
2. **Enquanto** (gene atual < tamanho do cromossomo)

3. Criação aleatória do gene no cromossomo; Decodificação do cromossomo;
4. Verificação da legalidade do cromossomo;
5. **Se** o cromossomo é legal
6. **Se** o cromossomo é legal
7. Decodificação do cromossomo;
8. Verificação da legalidade do cromossomo;
9. **Se** o cromossomo é legal
10. Avança para o próximo gene;
11. **Senão**
12. Volta-se para passo 3;
13. **Fim se**
14. **Fim enquanto**
15. **Fim enquanto**

% Início da evolução

16. **Enquanto** (geração atual < número de gerações)
17. Avaliação da população;
18. Seleção dos pais para o *crossover* levando em consideração a qualidade de cada indivíduo;
19. *Crossover* dos pais para a geração de herdeiros;
20. Decodificação dos herdeiros pelo *script decoding*;
21. Verificação da legalidade dos herdeiros pelo *script* controlesupervisorio;
22. **Se** os herdeiros são possíveis pela TCS
23. Com uma probabilidade de mutação (p_m), um dos herdeiros é submetido ao operador de mutação;
24. Decodificação dos herdeiros pelo *script decoding*;
25. Verificação da legalidade dos herdeiros pelo *script* controlesupervisorio;
26. **Se** os herdeiros são possíveis pela TCS
27. Inserção dos herdeiros na população, tomando lugar dos piores indivíduos;
28. Avanço para a próxima geração;
29. **Senão** Volta-se ao passo 15;
30. **Fim se**
31. **Senão**
32. Volta-se ao passo 15;

33. **Fim se**

34. **Fim enquanto**

35. Apresentação dos resultados;

% Fim do Algoritmo

Seguindo o pseudocódigo proposto, na fase de geração da população inicial, são gerados os indivíduos e nessa representação, um indivíduo corresponde a um cromossomo. Durante a construção de um indivíduo (cromossomo), cada gene é inserido de maneira aleatória, decodificado e avaliado. Se ao acrescentar o gene, o cromossomo se tornar ilegal, esse gene é descartado e se realiza novamente o sorteio de outro gene, sendo ele decodificado e avaliado novamente. Por esse motivo, afirma-se que a população inicial é constituída de cromossomos legais. Essa população é construída dessa maneira devido ao número de cromossomos legais ser considerado pequeno em relação a toda possibilidade de permutações de cromossomos possíveis de construção, garantindo assim, uma maior chance de que os cromossomos encontrados como solução final sejam factíveis ou legais. Após a construção dos cromossomos que formam a população, todos os cromossomos (indivíduos) são avaliados e classificados em ordem decrescente para serem selecionados (passo 14). Utilizou-se então, no processo de seleção (passo 15), o operador de seleção por classificação, onde os melhores possuem maiores chances de serem selecionados para o processo de cruzamento (*crossover*). São selecionados dois pais para realizarem o cruzamento.

Após a seleção dos pais, realiza-se o *crossover* (passo 16). Foi escolhido para esse algoritmo o operador de *crossover* de dois pontos de corte. Nesse operador, utilizam-se dois pontos sorteados entre 1 e a quantidade de genes dos cromossomos para gerar dois herdeiros. Por exemplo, sejam as expressões 4.14 e 4.15, os cromossomos selecionados para serem pais no processo de *crossover* e por sorteio, os dois pontos de corte sorteados sejam as posições 3 e 6. Portanto, os dois herdeiros gerados são apresentados em 4.16 e 4.17. Os herdeiros após serem gerados são avaliados novamente para poderem avançar para o processo de mutação. Se os herdeiros forem ilegais, novos pais são selecionados e cruzados.

[**1 1 2 2 1 2 1 2 1 1**] (4.14)

[1 1 1 1 2 2 1 2 2 1] (4.15)

[**1 1 2** 1 2 2 **1 2 1 1**] (4.16)

[1 1 1 **2 1 2** 1 2 1 1] (4.17)

O processo de mutação (passo 20) ocorre com uma probabilidade determinada. Neste algoritmo foram utilizados os valores de 1% e 25%, para demonstrar a influência da taxa de mutação sob o algoritmo. Utilizou-se o operador de mutação *swap*, uma vez que ele diminui as chances de geração de um cromossomo ilegal. Por exemplo, seja a expressão 4.14 o cromossomo que irá se submeter ao processo de mutação e os dois pontos de troca de genes sejam 2 e 7. O novo herdeiro após a mutação é apresentado em 4.18. Por fim, os herdeiros são avaliados e se legais, são inseridos na população no lugar dos indivíduos de pior qualidade. Se forem ilegais, retorna-se ao processo de seleção.

[1 **2** 2 2 1 2 1 **1** 1 1] (4.18)

Após a implementação do algoritmo, este foi testado e executado. Foi escolhido então, dois lotes de produção, um de10 peças (5 peças do tipo 1 e 10 peças do tipo 2) e 3 peças (1 peça tipo 1 e 2 peças do tipo 2). Os parâmetros escolhidos para o tamanho da população foram 10 e 20 indivíduos, para o número de gerações foram 100 e 500 e para probabilidade de mutação foram 1% e 25%. Em cada variação dos parâmetros, o algoritmo foi executado 10 vezes. A seguir, na Tabela 4.21, apresenta-se os resultados obtidos na execução para o lote de 10 peças, na Tabela 4.22 os resultados obtidos para a execução do lote de 3 peças

Tabela 4.21 - Resultados da implementação com a representação baseada em operação para lote de 10 peças.

Tamanho da população	**10**				**20**			
Número de gerações	**100**		**500**		**100**		**500**	
Probabilidade de mutação (%)	**1**	**25**	**1**	**25**	**1**	**25**	**1**	**25**
Média de tempo de produção do lote	1704	1704	1700	1700	1700	1700	1700	1700
Melhor resultado	1700	1700	1700	1700	1700	1700	1700	1700
Desvio Padrão	6,4	6,4	0	0	0	0	0	0
Processamento Sequencial	1800							

O Matlab também fornece o tempo de execução total em segundos e o número de vezes de iterações de cada etapa, codificada em uma função, implementada, valores esses importantes para a análise da performance do algoritmo. Esses valores foram obtidos a cada execução, calculadas as médias e são apresentados na Tabela 4.23 os resultados do lote de 10 peças e na Tabela 4.24 os resultados do lote de 3 peças.

Tabela 4.22 - Resultados da implementação com a representação baseada em operação para lote de 3 peças.

Tamanho da população	**10**				**20**			
Número de gerações	**100**		**500**		**100**		**500**	
Probabilidade de mutação (%)	**1**	**25**	**1**	**25**	**1**	**25**	**1**	**25**
Média de tempo de produção do lote	492	492	492	492	492	492	492	492
Melhor resultado	492	492	492	492	492	492	492	492
Desvio Padrão	0	0	0	0	0	0	0	0
Processamento Sequencial	512							

Tabela 4.23- Tabela de dados de performance da implementação para o lote de 10 peças (5 peças tipo 1 e 5 peças tipo 2).

Tamanho da população	**10**				**20**			
Número de gerações	**100**		**500**		**100**		**500**	
Probabilidade de mutação	**1**	**25**	**1**	**25**	**1**	**25**	**1**	**25**
Média de tempo (segundos)	777	942	902	1030	1332	1486	1014	1236
Média de iterações: *decoding*	13878	14192	15338	16325	31908	33686	28892	32373
Média de iterações: *controlesupervisorio*	13877	14191	15227	16324	31907	33685	28891	32373

Tabela 4.24- Tabela de dados de performance da implementação para o lote de 3 peças (1 peça tipo 1 e 2 peças tipo 2).

Tamanho da população	**10**				**20**			
Número de gerações	**100**		**500**		**100**		**500**	
Probabilidade de mutação	**1**	**25**	**1**	**25**	**1**	**25**	**1**	**25**
Média de tempo (segundos)	472	606	545	586	630	545	550	628
Média de iterações: *decoding*	753	790	882	898	1149	1217	1307	1475
Média de iterações: *controlesupervisorio*	752	789	881	897	1148	1216	1306	1474

A análise das Tabelas 4.21, 4.22, 4.23 e 4.24permite concluir que:

- O algoritmo implementado encontra resultados superiores ao processamento sequencial (1800 u.t. para lote de 10 peças e 512 para lote de 3 peças), isto é, obtém-se escalonamentos melhores do que ao processamento de uma peça por vez;
- A probabilidade de mutação (p_m) ao ser alterada, influencia na diversidade e na qualidade da população, de modo que todos os parâmetros de performance aumentaram, seja o tempo médio de execução do algoritmo, como a quantidade de iterações dos *scripts*;
- O tempo de execução do algoritmo e a quantidade de iterações dos *scripts* de decodificação (*decoding*) e do teste do controle supervisório (*controlesupervisorio*) reflete ao esforço computacional alto necessário para a obtenção de resultados;
- O esforço computacional alto de processamento do algoritmo é consequência de dois fatores:
    - Devido à explosão de estados presente na TCS. Por mais que a abordagem CSML reduza o número de estados e transições em comparação com a abordagem monolítica, ainda assim, para o problema exemplo proposto tem-se um número alto: 614 estados e 2141 transições;
    - Devido a geração da população inicial aleatória. Nota-se que o tamanho do cromossomo cresce de acordo com o número de peças obedecendo a expressão 4.6 e o tamanho do espaço de busca do algoritmo cresce exponencialmente. O espaço de busca do algoritmo corresponde ao espaço onde todas as possibilidades de permutação do cromossomo, incluindo cromossomos que sejam “ilegais” ou inválidos e este cresce de modo exponencial com o aumento do número de genes. Entretanto, o número de cromossomo que são possíveis ou legais pela TCS são de número reduzido e, por esse motivo, o esforço computacional é alto;
- Por fim, a metodologia proposta se prova válida, apesar do alto esforço computacional.

- **Caso 2: Representação em Chaves Aleatórias**

Na representação em chaves aleatórias, a solução é codificada através de um número aleatório que é utilizado como chave de codificação, onde cada gene consiste em dois símbolos:

um número natural de um dígito igual ao número de máquinas (m), que determina a máquina a ser utilizada, e uma fração gerada aleatoriamente entre zero e um. Essa parte fracional será usada para determinar a sequência de tarefas em cada máquina, através da permutação da posição dos genes, como será explicado mais adiante. O cromossomo tem tamanho também determinado pela expressão 4.10. Utilizando o sistema proposto para a validação da metodologia, um exemplo de cromossomo para um lote de peças constituído de uma peça tipo 1 e uma peça tipo 2 pode ser visualizado na expressão 4.19.

$$[1.157 \quad 1.814 \quad 1.934 \quad 1.276 \quad 2.706 \quad 2.427 \quad 2.485 \quad 3.438 \quad 3.276 \quad 4.751] \qquad (4.19)$$

No exemplo apresentado em 4.19, o cromossomo será interpretado da seguinte maneira: agenda-se na máquina 1 na seguinte ordem, a primeira operação da tarefa 1, a quarta operação da tarefa 2, a sexta operação da tarefa 1 e a primeira operação da tarefa 2. Na máquina 2, agenda-se na seguinte ordem, a segunda operação da tarefa 1, a segunda operação da tarefa 2 e a quarta operação da tarefa 1. Na terceira máquina, agenda-se primeiramente a terceira tarefa 2 e depois a terceira tarefa da tarefa 1. Por fim, na máquina 4 é agendada a quinta operação da tarefa 1. Observa-se que no agendamento das operações para o exemplo em 4.19 ocorre uma violação da ordem de precedência, ou seja, é agendada a quarta operação da tarefa 2 antes do agendamento da primeira operação. No caso de estudo, significa que se agenda primeiro a retirada da peça 2 do sistema sem que ela tenha entrado no sistema pela operação 2.

Essa codificação pode violar a ordem de precedência das tarefas, uma vez que essa codificação pode escalonar máquinas em ordem diferente de produção, como ocorre para a máquina 2 no exemplo apresentado em 4.19. Neste caso, nota-se que é agendada a primeira operação da tarefa 1 da máquina 1 e depois é agendada a primeira operação da tarefa 2 na máquina 1. Porém na máquina 2, como a máquina está disponível, é agenda-se a operação 2 da tarefa 2, infligindo a ordem de precedência das operações.

Como forma de contornar o problema de violação da ordem de precedência, adaptou-se a codificação. A parte inteira varia de 1 até o número de tarefas, onde cada parte inteira repete-se de acordo com o número de operações na respectiva tarefa, sendo ímpares para peças do tipo 1 e pares para peças do tipo 2. A parte fracionária aleatória determinará a ordem de agendamento das operações da seguinte forma: a parte fracionária será ordenada de maneira crescente, permutando consequentemente as operações das tarefas. Um exemplo dessa adaptação da representação, utilizando o sistema em questão, é apresentado em 4.20.

$$[1.905 \quad 1.9796 \quad 1.035 \quad 1.031 \quad 1.381 \quad 1.6797 \quad 2.255 \quad 2.2543 \quad 2.8308 \quad 2.054] \qquad (4.20)$$

Então, para decodificar a sequência de genes, primeiro considera-se apenas a parte fracionária, na qual ela é colocada em ordem crescente. O exemplo de cromossomo em 4.20 então, torna-se na sequência em 4.21. Assim, depois da ordenação empregando somente a parte fracionária, considera-se apenas a sequência dos símbolos de números naturais, a parte inteira, e se realiza a tradução de maneira semelhante com a representação baseada em operação: a primeira repetição do símbolo representa a primeira operação da tarefa identificada pelo símbolo que representa a subsequência invariante correspondente que equivale a uma subsequência de eventos. A segunda repetição do símbolo, a segunda operação e assim por adiante. No caso do exemplo de cromossomo 4.20, a sua decodificação em uma subsequência de operações e sequência de eventos são apresentadas em 4.22 e 4.23.

$$[1.031 \quad 1.035 \quad 2.054 \quad 2.254 \quad 2.255 \quad 1.381 \quad 1.679 \quad 2.830 \quad 1.905 \quad 1.979] \tag{4.21}$$

$$[\varepsilon_{11} \quad \varepsilon_2 \quad \varepsilon_{21} \quad \varepsilon_3 \quad \varepsilon_2 \quad \varepsilon_7 \quad \varepsilon_2 \quad \varepsilon_8 \quad \varepsilon_4 \quad \varepsilon_5] \tag{4.22}$$

$$Seq.even. = "E_{11}E_1E_2E_{12}E_9E_{10}E_{13}E_3E_4E_{14}E_{15}E_{16}E_9E_{10}E_{17}E_{18}E_9E_{10}E_7E_8E_{19}E_{20}E_5E_6" \tag{4.23}$$

Para realizar a decodificação do cromossomo em uma sequência de eventos no algoritmo desenvolvido, implementou-se um *script* (nomeado de *decoding2*) que é similar ao desenvolvido para a representação baseada em operações e apresentado no Apêndice A. Então, para a verificação da legalidade do cromossomo pela TCS, utilizando a sequência de eventos, foi desenvolvido outro *script* (de maneira similar aos desenvolvidos para as outras representações apresentado no Apêndice B) chamado de *controlesupervisorio2*. A seguir, no Algoritmo 4.2, apresenta-se o pseudocódigo para o algoritmo utilizando a representação em chaves aleatórias.

Algoritmo 4.2 – Pseudocódigo do algoritmo desenvolvido para a representação em chaves aleatórias.

% Geração da população inicial totalmente legal

1. **Enquanto** (cromossomo atual < tamanho da população)
2. Geração do cromossomo totalmente aleatório;
3. Ordenação do cromossomo e decodificação através do *script decoding2*;
4. Verificação da legalidade do cromossomo através do *script* controlesupervisorio2;
5. **Se** o cromossomo é legal
6. Avança para o próximo cromossomo;

7. **Senão**
8. Volta para o passo 2;
9. **Fim se**
10. **Fim enquanto**

% Início da evolução

11. **Enquanto** (geração atual < número de geração)
12. Avaliação da população total e ordenação da população em ordem crescente;
13. Seleção dos pais para o *crossover* levando em consideração a qualidade de cada indivíduo;
14. *Crossover* dos pais para a geração de herdeiros;
15. Decodificação dos herdeiros através do *script decoding2*;
16. Verificação da legalidade dos herdeiros através do *script* controlesupevisorio2;
17. **Se** os herdeiros são legais
18. Com uma probabilidade p_m, um dos herdeiros é submetido ao operador de mutação;
19. Decodificação dos herdeiros através do *script decoding2*;
20. Verificação dos herdeiros através do *script* controlesupervisorio2;
21. **Se** os herdeiros são legais
22. Inserção dos herdeiros no lugar dos dois piores indivíduos da população;
23. **Senão**
24. Volta para o passo 12;
25. **Fim se**
26. **Senão**
27. Volta para o passo 12;
28. **Fim se**
29. **Fim enquanto**
30. Apresentação dos resultados

%Fim do Algoritmo

De acordo com o pseudocódigo proposto, na fase da geração inicial do cromossomo (passo 2), ocorre a construção do cromossomo. Os valores são gerados de maneira aleatória e então, ordenados também de maneira aleatória, juntamente com as tarefas, permutando-as. En-

Então, utilizando o *script decoding2* traduz-se esses cromossomos em sequências de eventos (passo 3). Na sequência, o *script controlesupervisório2* avalia e verifica se o cromossomo atende às especificações que foram modeladas pela TCS (passo 4). Se o cromossomo verificado for legal, avança para a geração do próximo cromossomo. Caso isto não ocorra, gera-se um novo cromossomo, que novamente será decodificado e avaliado. Essa construção dos cromossomos é feita dessa maneira para garantir que todos os cromossomos da população inicial sejam legais e factíveis. Garantindo a legalidade dos cromossomos, tem-se uma maior chance de que os cromossomos encontrados na solução final do algoritmo sejam legais ou factíveis, visto que a porcentagem de cromossomos que atendem a TCS é muito pequena dentre todos os cromossomos que podem ser permutados.

Após a construção dos cromossomos da população inicial, dá-se início ao processo de evolução. Primeiramente, esses cromossomos (ou indivíduos) são avaliados e classificados em ordem decrescente (passo 12) para então serem selecionados. No processo de seleção (passo 13), de maneira similar ao algoritmo de representação baseada em operação, utilizou-se o operador de seleção por classificação, sendo que os melhores cromossomos (indivíduos) possuem mais chances de serem selecionados para o processo de cruzamento. Os indivíduos selecionados são chamados de pais, que neste caso, são dois.

Posteriormente a seleção dos dois pais, aplica-se o operador de *crossover* de dois pontos de corte (passo 14), da mesma maneira que ocorre na representação baseada em operação, onde geram-se dois herdeiros provenientes dos dois pais. Em seguida, esses dois herdeiros são avaliados e verificados pela TCS, utilizando os *scripts decoding2* (passo 15) e *controlesupervisorio2* (passo 16). Caso os dois herdeiros não sejam legais, dois novos pais são selecionados e um novo processo de *crossover* é realizado. Caso os dois herdeiros sejam legais, avança-se para o processo de mutação.

O processo de mutação (passo 18) ocorre com uma probabilidade pré-determinada. Neste algoritmo foram utilizados os valores de 1% e 25% de chance de ocorrer uma mutação com um dos herdeiros do processo de *crossover*. Para esse processo, utilizou-se um operador de mutação *swap* que corresponde ao mesmo operador utilizado no algoritmo da representação baseada em operação, dado que, como já citado, esse operador diminui as chances de gerar um cromossomo que seja ilegal.

Por fim, os novos herdeiros gerados dos processos de *crossover* e mutação, são novamente decodificados em sequência de eventos e verificados pela modelagem feita pela TCS u-

tilizando os *scripts decoding2* (passo 19) e controlesupervisorio2 (passo 20). Caso os herdeiros sejam factíveis e legais eles são inseridos na população tomando lugar dos dois piores indivíduos da população (passo 22) e avança-se para a próxima geração. Caso não sejam legais, volta-se novamente ao processo da seleção, para um novo *crossover* e processo de mutação até encontrar indivíduos que sejam factíveis e legais.

Devido à natureza aleatória da construção da população inicial do algoritmo na representação em chaves aleatórias, optou-se por escolher um lote pequeno de peças em vista do grande esforço computacional necessário, uma vez que ao tentar executar os outros lotes, o algoritmo não conseguiu alcançar resultados. Devido a essa limitação, optou-se então, um lote de produção de 3 peças, sendo estas 1 do tipo 1 e 2 do tipo 2.

Os parâmetros que devem ser escolhidos, de maneira similar ao algoritmo da representação baseada em operação, para o AG que interferem em sua performance foram os seguintes. Para o tamanho da população foram utilizados os valores de 10 e 20 indivíduos, para o número de gerações foram determinados os utilizados 100 e 500 gerações e por fim, para a probabilidade de mutação, como já citado, foram escolhidos os valores de 1% e 25% de chance de ocorrência em um dos herdeiros.

O algoritmo foi executado 10 vezes para cada configuração de parâmetros. Na Tabela 4.25, apresentam-se os resultados da implementação e execução para esse lote. Como no algoritmo o anterior, também foram obtidos o tempo de execução total em segundos e a quantidade de iterações de cada *script* implementado (*decoding2* e *controlesupervisorio2*). Esses valores são importantes para a análise de performance do algoritmo e se encontram na Tabela 4.26.

Tabela 4.25 - Resultados da implementação e execução.

Tamanho da população	**10**				**20**			
Número de gerações	**100**		**500**		**100**		**500**	
Probabilidade de mutação (%)	**1**	**25**	**1**	**25**	**1**	**25**	**1**	**25**
Média	496	496	492	492	494	494	492	492
Melhor resultado	492	492	492	492	492	492	492	492
Desvio padrão	6,4	6,4	0	0	3,6	3,6	0	0
Processamento sequencial	512							

Tabela 4.26 - Dados de performance utilizando a representação em chaves aleatórias.

Tamanho da população	**10**				**20**			
Número de geração	**100**		**500**		**100**		**500**	
Probabilidade de mutação (%)	**1**	**25**	**1**	**25**	**1**	**25**	**1**	**25**
Média de tempo (segundos)	1735	1828	1953	1783	2151	2069	2251	2147
Média de iterações: *decoding2*	19940	19483	27531	21720	50060	39297	47757	46711
Média de iterações: controlesupervisorio2	19941	19484	27532	21721	50061	39298	47758	46712

As conclusões obtidas através da análise das Tabelas 4.25 e 4.26 são apresentadas a seguir:

- Primeiramente, conclui-se que a adaptação dessa representação se torna ineficiente enquanto aumenta-se o tamanho do cromossomo através do incremento de peças no lote. Isso ocorre por essa representação se basear no sorteio de posições das tarefas e operações e, na medida em que cresce o número de peças, as possibilidades de construção de cromossomos crescem exponencialmente, enquanto as soluções factíveis e legais são poucas frente a todos os possíveis cromossomos;
- Como previsto, o incremento do número de gerações no algoritmo promove melhores soluções para o problema de escalonamento, como consta na Tabela 4.25, onde a média das soluções encontradas nas execuções com maiores gerações é superior as execuções com menores gerações;
- Apesar de uma probabilidade maior de mutação proporcionar ao algoritmo uma maior diversidade, não ocorram diferenças nas execuções com valores diferentes. Uma possível explicação para esse comportamento é devido ao tamanho de lote ser pequeno;
- No quesito de performance do algoritmo, a probabilidade de mutação não possui influência, porém o número de gerações e o tamanho de população afetam de maneira direta a performance;
- Assim como a representação baseada em operação, o tempo de processamento elevado e a quantidade de iterações dos *scripts* de decodificação em sequência de eventos (*decoding2*) e do teste de legalidade através da TCS (*controlesupervisorio2*) reflete no alto esforço computacional necessário para a obtenção de soluções;
- Tal como a representação baseada em operação, uma possível explicação para o alto esforço computacional é devido ao problema de explosão de estados presente na TCS

e principalmente, ao fato da geração da população inicial ser aleatória. No caso da representação em chaves aleatórias, se torna claro que a geração da população inicial é a principal influência na performance do algoritmo;

- Apesar do alto esforço computacional, foi possível obter resultados satisfatórios com o algoritmo. Isto é, nota-se que as médias das soluções encontradas são superiores em relação ao processamento sequencial (uma peça por vez), validando assim a metodologia.

- **Caso 3: Representação baseada em Regra de Prioridade**

Na representação baseada em regra de prioridade, um cromossomo codifica uma sequência de regras de despacho ou regras de prioridade de agendamento para a atribuição de uma tarefa. O cromossomo nessa representação também possui tamanho determinado pela expressão 4.6, porém cada gene representa uma regra de prioridade para a escolha da operação a ser agendada. Uma regra de prioridade indica apenas qual operação deve ter preferência em ser agendada. Entretanto, isto ocorre apenas se for possível o agendamento e se não violar nenhuma restrição, seja de precedência ou do próprio sistema. Nessa representação, um gene identifica, através de um símbolo, a regra de prioridade que determina a operação que será agendada. Essa operação é associada a uma subsequência invariante que é associada a uma subsequência de eventos. A operação determinada pela regra de prioridade e que leva o sistema a violar regras de precedência e restrições de segurança leva, consequentemente, o sistema a comportamentos indesejados. Neste caso, a operação é descartada no agendamento e a próxima operação a atender a regra determinada é escolhida.

Para o sistema proposto foram escolhidas duas regras, apresentadas na Tabela 4.27. Assim, os genes do cromossomo serão representados por 1 ou 2, indicando, respectivamente, a regra a ser selecionada. Por exemplo, empregando essa representação, para a produção de duas peças, sendo uma de cada tipo, a expressão 4.24 apresenta o gene.

Tabela 4.27- Tabelas das regras de prioridade aplicadas ao problema.

Regra	Descrição
1	Selecione a operação com o menor tempo de processamento
2	Seleciona a operação com o maior tempo de processamento

[2 1 2 1 1 1 2 1 1 2] (4.24)

A representação baseada em regra de prioridade é classificada como uma abordagem indireta e, por este motivo, é necessária a tradução do cromossomo em uma sequência de operações para então traduzir essa sequência de operações em uma sequência de eventos. No caso, o gene do cromossomo corresponde aos símbolos 1 e 2 que identificam as regras de prioridades propostas. Nas regras de prioridade propostas, é necessário levar em consideração o tempo das operações e a sequência de máquinas para a produção das peças. Nas Tabelas 4.28 e 4.29 são apresentadas as sequências de máquinas e as durações das operações medidas em unidades de tempo (u.t.) das produções das peças tipo 1 e tipo 2, respectivamente. As operações, para serem diferenciadas por tipo de peça, foram acrescidas do símbolo *a* para peças do tipo 1 e *b* para peças do tipo 2.

Tabela 4.28- Sequência de máquinas e durações para a produção da peça tipo 1.

Operação	**1a**	**2a**	**3a**	**4a**	**5a**	**6a**
Sequência de máquinas	M_1	M_2	M_3	M_2	M_4	M_1
Duração das operações (u.t.)	30	20	60	20	49	29

Tabela 4.29- Sequência de máquinas e durações para a produção da peça tipo 2.

Operação	**1b**	**2b**	**3b**	**4b**
Máquinas	M_1	M_2	M_3	M_1
Duração das operações (u.t.)	34	20	70	29

Utilizando esse sistema de representação e o cromossomo de exemplo da expressão 4.24, a decodificação do mesmo em uma sequência de eventos é feita como segue. Para o primeiro gene do exemplo, identifica-se que a regra de prioridade a ser obedecida (através da Tabela 4.27) é a regra de selecionar a operação com o menor tempo (regra 1). Assim, selecionam-se as operações cujo agendamento é possível. Obedecendo as regras de precedência, nota-se que as duas operações possíveis são as operações 1a e 1b. Então, é selecionada a operação 1a, pois possui o menor tempo de duração das operações. Uma vez identificada a operação, é selecionada a máquina correspondente a essa operação, sendo assim a máquina M_1. Nesse momento, são selecionadas todas as operações que podem ocorrer na máquina M_1 obedecendo as regras de precedência, tendo assim as operações 1a e 1b e utilizando a regra de prioridade determinada pelo cromossomo, seleciona-se a operação 1a.

Desse modo, a regra selecionada consiste em selecionar a operação com menor tempo de processamento e se essa operação obedecer às regras de procedência e sendo possível pela TCS,

agenda-se assim a operação. Neste ponto, ocorre a tradução das operações em subsequência de eventos de maneira similar à que ocorre nas outras duas representações e, também de maneira similar, é realizada a comprovação da sequência pela TCS. Novamente, foram desenvolvidos dois *scripts*: um que traduz a sequência de operação em uma sequência de eventos chamado de *decoding3* e outro que realiza a verificação da legalidade pela TCS que foi chamado de *controlesupervisorio3*. O *script decoding3*, assim como a representação em chaves aleatórias, é similar ao *script decoding* e é apresentado no Apêndice A. O *script controlesupervisorio3* também é similar ao *script controlesupervisorio* e é apresentado no Apêndice B. Para o processo de tradução do cromossomo para sequência de operações foi desenvolvido um *script* chamado de *translate2*, o qual segue o procedimento de decodificação do cromossomo apresentado no Procedimento 1. Nesse *script*, a decodificação do cromossomo é realizada de modo que o cromossomo seja factível e possível segundo os modelos da TCS.

No Algoritmo 4.3 é apresentado o pseudocódigo do *script translate2*, que realiza a decodificação do cromossomo em sequência de operações que são legais e factíveis segundo a TCS. Esse *script* utiliza os dois *scripts decoding3* e *controlesupervisorio3*.

Algoritmo 4.3 – Pseudocódigo do *script translate2*.

%Início do *script*

1. **Para** o primeiro gene até o último
2. S recebe todas as operações possíveis de agendamento;
3. Auxiliar=0;
4. **Enquanto** (Auxiliar não for 1)
5. Encontrar a operação (Φ) com o menor tempo de processamento;
6. **Se** o conjunto S não tem tamanho zero
7. Seleção da máquina m onde a operação (Φ) é realizada;
8. Construção do conjunto de concorrência (C_t) que concorre pela máquina m;
9. Auxiliar2=0;
10. **Enquanto** (Auxiliar2 não for 1)
11. **Se** o conjunto de concorrência tem tamanho zero
12. Elimina-se a operação selecionada do conjunto S e volta ao passo 3;

13. **Senão**
14. **Se** o gene atual é igual 1
15. Seleção da operação com menor tempo de processamento;
16. Inserção na sequência de operação;
17. Decodificação através do *script decoding3*;
18. Verificação da legalidade da sequência através do *script controlesupervisorio3*; **Se** a sequência é legal
19. Auxiliar=1;
20. Auxiliar2=1;
21. **Senão**
22. Eliminação da operação da sequência e do conjunto de concorrência e retorna ao passo 15;
23. **Fim se**
24. **Fim se**
25. **Se** o gene atual é igual a 2
26. Seleciona a operação com o maior tempo de processamento;
27. Inserção na sequência de operação;
28. Decodificação através do *script decoding3*;
29. Verificação da legalidade da sequência através do *script controlesupervisorio3*;
30. **Se** a sequência é legal
31. Auxiliar=1;
32. Auxiliar2=1;
33. **Senão**
34. Elimina-se a operação da sequência e do conjunto de concorrência (C_t) e volta ao passo 27;
35. **Fim se**
36. **Fim se**
37. **Fim se**
38. **Fim enquanto**
39. **Senão**
40. Auxiliar=1;
41. **Fim se**

42. **Fim enquanto**
43. **Fim para**
44. Retorna a sequência de operações

%Fim do *script*

Na sequência, utilizando os *scripts translate2*, *decoding3* e *controlesupervisorio3*, implementa-se o AG proposto, apresentado no Algoritmo 4.4.

Algoritmo 4.4 – Pseudocódigo proposto para o algoritmo utilizando a representação baseada em regra de prioridade.

%Geração da população inicial

1. Geração da população inicial em cada gene é sorteado entre o valor 1 ou 2;
2. **Enquanto** (geração atual < número de geração)
3. Tradução em sequência de operações pelo *script translate2* para a avaliação dos cromossomos;
4. Seleção dos pais para o *crossover* levando em consideração a qualidade de cada indivíduo;
5. *Crossover* para a geração de herdeiros;
6. Com uma probabilidade p_m, um dos herdeiros é submetido ao operador de mutação;
7. Tradução pelo *script translate2* e avaliação dos herdeiros;
8. **Se** os dois herdeiros são melhores que os dois piores indivíduos da população atual
9. Herdeiros são inseridos na população no lugar dos dois piores indivíduos da população;
10. **Fim se**
11. Avança para a próxima geração;
12. **Fim enquanto**
13. Apresentação de resultados;

%Fim do Algoritmo

Seguindo o pseudocódigo apresentado no Algoritmo 4.4, a fase de geração da população inicial (passo 1) é a fase em que os indivíduos são criados, onde cada indivíduo corresponde a um cromossomo. Durante a construção do cromossomo, para cada gene é sorteado o valor 1 ou o valor 2.

Esse cromossomo posteriormente será traduzido utilizando o *script translate2* para ser avaliado. De maneira similar ao algoritmo que utiliza a representação baseada em operação, após o processo de geração da população inicial, esses cromossomos são avaliados (passo 2) e classificados em ordem crescente para dois deles serem selecionados. Utilizou-se também no processo de seleção (passo 4), o operador de seleção por classificação, onde os melhores possuem maiores chances de serem selecionados para o processo de *crossover*.

Após a seleção dos pais, realiza-se o processo de *crossover* (passo 5). Para este processo, similarmente as outras duas representações, selecionou-se o operador de *crossover* de dois pontos de corte, gerando através de dois pais, dois herdeiros.

Os herdeiros então, com uma probabilidade p_m, são submetidos ao processo de mutação (passo 6), que, assim como as outras duas representações, utiliza-se o operador *swap* que troca a posição de dois genes de lugar. Para esse algoritmo, a probabilidade de mutação que é proposta é de 1% e 25%. Por fim, após todos esses processos, os herdeiros são novamente avaliados e se eles forem melhores que os dois piores indivíduos da população, os herdeiros são inseridos em seus lugares.

Foram escolhidos, assim como no caso da representação baseada em operação, dois lotes: um lote de 10 peças (5 peças do tipo 1 e 5 peças do tipo 2) e um lote de 3 peças (1 peça tipo 1 e 2 peças tipo 2). Para o tamanho da população foram determinados os valores de 10 e 20 indivíduos. Para o número de gerações foram determinados os valores de 100 e 500 gerações e por fim, para a probabilidade de mutação, como já citado, foram escolhidos os valores de 1% e 25% de chance de ocorrência em um dos herdeiros.

O algoritmo foi executado 10 vezes para cada configuração de parâmetros. Na Tabela 4.30 apresenta-se os resultados da implementação para a execução do lote de 10 peças e na Tabela 4.31 os resultados para o lote de 3 peças.

Tabela 4.30- Resultados da implementação do algoritmo utilizando a representação baseada em regra de prioridade com um lote de 10 peças.

Tamanho da população	**10**				**20**			
Número de gerações	**50**		**100**		**50**		**100**	
Probabilidade de mutação (%)	**1**	**25**	**1**	**25**	**1**	**25**	**1**	**25**
Média	1704	1704	1702	1702	1702	1702	1700	1700
Melhor resultado	1700	1700	1700	1700	1700	1700	1700	1700
Desvio padrão	6,4	6,4	3,6	3,6	3,6	3,6	0	0
Processamento sequencial	1800							

Tabela 4.31- Resultados da implementação do algoritmo utilizando a representação baseada em regra de prioridade com um lote de 3 peças.

Tamanho da população	**10**				**20**			
Número de gerações	**50**		**100**		**50**		**100**	
Probabilidade de mutação (%)	**1**	**25**	**1**	**25**	**1**	**25**	**1**	**25**
Média	492	492	492	492	492	492	492	492
Melhor resultado	492	492	492	492	492	492	492	492
Desvio padrão	0	0	0	0	0	0	0	0
Processamento sequencial	512							

Como nos casos anteriores, também foram obtidos o tempo de execução total em segundos do algoritmo e a quantidade de iterações de cada *script* implementado (*decoding3*, *translate2* e *controlesupervisorio3*). Esses valores são importantes para a análise de performance do algoritmo e se encontram na Tabela 4.32 os resultados da implementação com 10 peças e na Tabela 4.33 os resultados da implementação com 3 peças.

Tabela 4.32- Dados de performance de execução do algoritmo de implementação com a representação baseada em regra de prioridade com 10 peças.

Tamanho da população	**10**				**20**			
Número de gerações	**50**		**100**		**50**		**100**	
Probabilidade de mutação (%)	**1**	**25**	**1**	**25**	**1**	**25**	**1**	**25**
Média de tempo (segundos)	3074	3088	5410	5777	5010	5212	9377	9445
Média de iterações: *translate2*	599	599	1199	1199	1099	1099	2199	2199
Média de iterações: *decoding3*	209600	217167	435690	411026	391223	396943	784398	782492
Média de iterações: controlesupervisorio3	209599	217166	435689	411025	391222	396942	784397	782491

Tabela 4.33- Dados de performance de execução do algoritmo utilizando a representação baseada em regra de prioridade com um lote de 3 peças.

Tamanho da população	**10**				**20**			
Número de gerações	**50**		**100**		**50**		**100**	
Probabilidade de mutação (%)	**1**	**25**	**1**	**25**	**1**	**25**	**1**	**25**
Média de tempo (segundos)	483	463	581	560	541	593	737	739
Média de iterações: *translate2*	599	599	1199	1199	1099	1099	2199	2199
Média de iterações: *decoding3*	11636	11621	23227	23323	21278	21315	42594	42060
Média de iterações: controlesupervisorio3	11635	11620	23226	23322	21277	21314	42593	42059

A partir da análise das Tabelas 4.30, 4.31, 4.32 e 4.33 é possível concluir que:

- No quesito de geração da população inicial, esse algoritmo difere dos outros implementados, uma vez que, a geração é aleatória e o cromossomo é constituído por regras e ele sempre será considerado possível ou legal. Por este motivo, nesse algoritmo a geração da população inicial não interfere em sua performance;
- O incremento no número de gerações e no tamanho da população consegue encontrar resultados melhores, como pode se notar nas Tabela 4.32 e 4.33, devido a esses parâmetros aumentarem a intensidade e a diversidade de busca do algoritmo;
- O parâmetro de probabilidade de mutação (p_m) não parece interferir na eficiência do algoritmo, porém, sabe-se que pode promover uma diversidade. Conclui-se que, nesse caso, como o lote de peças é pequeno, o parâmetro p_m possui pouca influência;
- Apesar de p_m possuir uma pequena influência na performance do algoritmo, o número de gerações e tamanho da população impacta amplamente, como pode se notar naa Tabela 4.30 e 4.31;
- Tal como os outros algoritmos, esse também necessita de um alto esforço computacional. Entretanto, esse esforço é reflexo da necessidade de três *scripts*: um *script* (*translate2*) que realiza a tradução do cromossomo que é constituído por regras de prioridade em uma sequência de operações, para aí sim ser decodificado utilizando outro *script* (*decoding3*) para uma sequência de eventos e finalmente ser avaliado por outro *script* (*controlesupervisorio3*);

- Dois dentre os três *scripts* citados acima são os principais responsáveis pelo alto esforço computacional:
    - *Script controlesupervisorio3*: visto que ele é responsável pela verificação através da TCS, ocorre o problema de explosão de estados já apresentado, apesar da modelagem utilizando a CSML;
    - *Script translate2*: devido à natureza do procedimento para a tradução do cromossomo em uma sequência de operações. Seguindo a regra de prioridade que é imposta pelo gene, muitas vezes a operação resultante dessa regra não pode ser agendada e ela é eliminada, sendo novamente selecionada a próxima operação que atende a regra, podendo levar a um grande esforço pela busca da operação que atende as restrições;
- Apesar do alto esforço computacional, consegue-se obter resultados satisfatórios com o algoritmo. Isto é, nota-se que as médias das soluções encontradas são superiores em relação ao processamento sequencial (uma peça por vez), validando assim a metodologia.

A seguir, é realizado um breve paralelo entre as três representações implementadas.

4.2.4 Análise Comparativa entre as Representações Implementadas

Nesta seção é realizado um breve comparativo entre as soluções encontradas, bem como as performances dos algoritmos.

- **Representação baseada em operação:**
    - Por ser uma representação classificada como direta, tem uma maior facilidade de visualizar no cromossomo a sequência de operações que determinam o agendamento;
    - A implementação do *script* que ocorre na etapa da geração da população inicial foi considerada mais complexa que as outras duas representações implementadas, visto que ela é a mais trabalhosa;

- Dado que os cromossomos possíveis e legais são considerados poucos em vista de todos os possíveis cromossomos existentes no espaço de busca, buscou-se garantir que toda a população inicial fosse constituída de cromossomos legais. Dessa forma, a geração da população inicial para essa representação não pode ser considerada aleatória, já que se utilizou um processo de indução na geração dos genes no cromossomo;
- Das três representações implementadas, foi a que teve uma melhor performance em relação ao tempo de execução. Entretanto, essa condição não pode ser garantida, uma vez que se os lotes aumentam, e o consequente tamanho do cromossomo, o tempo de execução tende a crescer de maneira exponencial;
- Por fim, também se notou que o alto tempo de execução exigido é proveniente da etapa de geração da população inicial, enquanto nas etapas evolutivas, o tempo de execução é considerada pequeno.

- **Representação em chaves aleatórias:**
  - Apesar de ser uma representação direta, a visualização da sequência de operações no cromossomo não é tão fácil, como ocorre na representação baseada em operação;
  - A implementação do algoritmo na etapa de geração da população inicial, especialmente na geração do indivíduo (cromossomo) foi considerada simples, porém a geração de um cromossomo que seja legal é considerada difícil;
  - Assim como na representação baseada em operação, pelos mesmos motivos buscou-se garantir que a população inicial fosse totalmente legal. Diferentemente da representação baseada em operação, a geração da população nessa representação é aleatória;
  - A representação em chaves aleatórias apresentou ótimos resultados para o sistema proposto para a validação da metodologia quando foram utilizados lotes pequenos. No caso, o tempo de execução foi menor em comparação com as outras representações utilizando lotes menores. A implementação dessa adaptação da representação se mostra ineficiente quando os lotes aumentam, aumentando o tamanho do cromossomo e as possibilidades de geração dos cromossomos se tornam exponenciais;

- Assim como a representação baseada em operação, o alto tempo de execução é proveniente da etapa de geração da população inicial. O tempo das etapas evolutivas é considerado pequeno;
- Utilizando essa representação, a autora considera que a implementação do algoritmo foi a mais simples, porém é o mais ineficiente entre os três que foram implementados.

- **Representação baseada em regra de prioridade:**
  - Por se tratar de uma representação classificada como indireta e o cromossomo representar uma sequência de regras de prioridade, não há maneiras de conseguir visualizar a sequência de operações determinada pelo cromossomo;
  - A geração da população inicial foi considerada a mais simples e fácil de implementar;
  - A geração da população inicial por ser realizada através de sorteios de regras é considerada aleatória;
  - Essa representação, ao contrário das outras duas implementadas, necessita de uma etapa de tradução do cromossomo em uma sequência de operações. A implementação dessa etapa em um *script* foi considerada complexa, uma vez que em sua implementação utilizou-se diversos aspectos do problema, como os tempos de duração das operações, as máquinas envolvidas, as regras de prioridade, conversão em sequência de eventos e verificação pela TCS;
  - Nas etapas de avaliação da população e também na verificação dos herdeiros, foi necessário a utilização de um *script*, aumentando o tempo de execução do algoritmo. A autora considera que o alto tempo de execução do algoritmo é reflexo do *script* para a tradução do cromossomo em sequência de operações;
  - Diferentemente das outras duas representações, no algoritmo utilizando a representação baseada em regra de prioridade, o alto tempo de execução do algoritmo não é proveniente da etapa de geração da população inicial, mas das etapas evolutivas do algoritmo;

- O algoritmo dessa representação foi considerado o melhor, entre as três representações, no que se refere a encontrar uma boa solução, apesar do alto tempo de execução. Isto ocorreu devido ao fato do tempo de execução não crescer de maneira exponencial com o aumento do tamanho do cromossomo que é consequência do aumento do tamanho de lotes. Nessa representação, o tempo de execução possui uma maior dependência de dois parâmetros: o tamanho da população e o número de gerações no processo de evolução.

Este comparativo é feito como forma de avaliação das implementações realizadas e principalmente, como avaliação das representações para o problema proposto. Um breve resumo das comparações é apresentado na Tabela 4.34. Nota-se que todos os algoritmos possuem um tempo de execução alto, o que, para a aplicabilidade prática, é um fator crítico para a implementação em indústrias.

Tabela 4.34 - Tabela comparativa entre as implementações.

Dados avaliadores	**Caso 1**	**Caso 2**	**Caso 3**
Performance	Melhor	Ruim	Razoável
Dificuldade de implementação	Muito complexo	Simples	Complexo
Dificuldade de codificação do cromossomo	Simples	Simples	Complexo
Tempo de execução versus tamanho do lote	Melhor para lotes maiores	Melhor para lotes pequenos e pior para lotes maiores	Razoável para todos os tamanhos de lote
Etapa com maior custo computacional	Etapa de geração da população inicial	Etapa de geração da população inicial	Etapa evolutiva

4.3 Considerações finais

Neste capítulo foi apresentada a metodologia que foi proposta para problemas de escalonamento de tarefas utilizando a metaheurística do Algoritmo Genético (AG) combinado com a Teoria do Controle Supervisório (TCS) como forma de descrição das restrições do sistema. Essa metodologia é constituída de 3 etapas: modelagem do sistema pela CSML e síntese dos seus supervisores, aplicação do algoritmo genético ao problema de escalonamento utilizando a TCS e por fim, a implementação de estruturas de controle e otimização em equipamento industrial.

Como forma de validação, a metodologia é aplicada a um sistema teste, o qual é encontrado no Laboratório de Robótica da Unioeste.

Utilizando a metodologia, na primeira etapa, foi modelado o sistema através da abordagem CSML e realizada a síntese de seus supervisores.

Após a modelagem e síntese dos supervisores, foi implementado o AG utilizando três diferentes representações para os cromossomos: representação baseada em operação, representação em chaves aleatórias e representação baseada em regra de prioridade. Foram apresentados os resultados das implementações discutindo quais, para o sistema proposto, foram melhores e quais foram estas justificativas.

Capítulo 5

Conclusão

O trabalho teve como objetivo elaborar uma metodologia para a solução do problema de escalonamento de tarefas utilizando a metaheurística de algoritmos genéticos (AG) onde a TCS fornece o conjunto de sequência de operações que são possíveis ou legais. Para a modelagem do sistema foi empregado a teoria das máquinas de estados finitos com parâmetros, que permite incluir a representação de tempo necessária ao problema de otimização do AG.

A metodologia proposta é dividida em três etapas: a primeira constituiu-se da modelagem da planta e síntese dos supervisores; a segunda etapa, da aplicação do algoritmo genético ao problema de escalonamento em plantas que estão representadas por SEDs utilizando a TCS. Então, como forma de validação da metodologia, ela foi aplicada a um estudo de caso que é um sistema real que se encontra no Laboratório de Robótica da Unioeste, onde a função objetivo era encontrar o tempo mínimo de produção

Empregando a metodologia, na primeira etapa foi realizada a modelagem do sistema. Para a modelagem da planta, utilizou-se a teoria das maquinas de estados finitos com parâmetros para apresentar as componentes de tempo de realização das tarefas. As restrições do sistema foram modeladas através de especificações do sistema. A implementação do algoritmo genético para o problema de escalonamento utilizando a TCS, foi feita para três representações para a codificação do cromossomo: representação baseada em operação, representação em chaves aleatórias e representação baseada em regra de prioridade.

Das implementações, pode-se concluir que:

- O número de soluções existentes no espaço de busca cresce exponencialmente com o tamanho do cromossomo, porém, as soluções legais do sistema são muito poucas. Por isso, foi determinado que a população inicial seria constituída apenas de soluções legais. Por esse motivo, para as duas representações diretas (operação e chaves aleatórias), a geração da população inicial apresentou um alto esforço computacional;

- Ao contrário das representações diretas, a representação baseada em regra de prioridades apresentou um esforço computacional maior para o processo evolutivo, ou seja, as fases de avaliação, seleção, *crossover*, mutação e inserção na população. Isto ocorreu devido ao fato que essa representação necessita de uma etapa de tradução do cromossomo em uma sequência de operações; A implementação do algoritmo utilizando a representação em chaves aleatórias foi considerada a mais simples. Entretanto, essa representação foi a menos eficiente e teve a pior performance, uma vez que se torna inviável com o crescimento de lotes de peças;
- A representação baseada em operação foi a que apresentou uma performance melhor na questão do tempo de execução para os lotes propostos, porém, foi considerada pela autora, o algoritmo mais complexo para a implementação;
- Considerou-se a melhor representação para encontrar uma boa solução, a representação baseada em regra de prioridade. A justificativa se encontra na performance do algoritmo em relação ao tempo, pois não cresce de maneira exponencial quando se incrementa o tamanho do cromossomo.

Por fim, tais resultados encontrados corroboram a hipótese neste trabalho, que a utilização da TCS dentro do problema de escalonamento consegue evitar a descrição analítica das restrições e que a metodologia empregada consegue resolver problemas de escalonamentos em SFMs cujo comportamento dinâmico dirigido a eventos de uma planta modelo de acordo com o CSML.

A partir dos resultados e contribuições alcançadas, novos trabalhos podem ser vislumbrados, dentre os quais citam-se:

- A implementação em equipamento industrial das estruturas de controle supervisório através de controladores lógicos programáveis (CLPs) integrado a uma estrutura de supervisão, aquisição e otimização do sistema através de sistemas Scada;
- A aplicação de outras formas de representação do cromossomo na metodologia e uma comparação de seus resultados
- O desenvolvimento de formas de melhorar a performance do algoritmo, incluindo a transcrição dos algoritmos para linguagens computacionais mais eficientes;
- A implementação de outras metaheurísticas para o problema de escalonamento de tarefas utilizando a TCS para a descrição das restrições do problema.

Referências Bibliográficas

Alves, L. V. R. (2016). *Planejamento da Produção em Sistemas a Eventos Discretos – Análise Lógica e Temporal,* Dissertação de Mestrado, Universidade Federal de Minas Gerais, Belo Horizonte, Brasil.

Alur, R. Dill, D. (1990). Automata for modeling real-time systems, *Proceedings of 17th International Colloquium on Automata, Languages and Programmig*, Springer, Berlin, Heidelberg, pp. 322-335

Becerra, R. L., & Coello, C. A. C. (2005). A Cultural Algorithm for Solving the Job Shop Scheduling Problem, *Knowledge Incorporation in Evolutionary Computation*, Springer, Berlin, Heidebelger, pp. 37:55.

Braga, R. P. F. (2006). *Contribuições ao Controle Supervisório de Sistemas Modelados por Redes de Petri,* Dissertação de Mestrado, Universidade Federal de Santa Catarina, Florianópolis, Brasil.

Brandin, B. A. Wonham, W. M. (1994). Supervisory Control of Timed Discrete-Event System, *IEEE Transactions on Automatic Control*, **39**(2): 329-342.

Brave, Y. Heymann, M. (1998). Formulation and control of real time discrete event process, *Proceedings of the 27th IEEE Conference on Decision and Control*, Austin, Estados Unidos da América, pp.1131-1132.

Browne, J. (1984). Classification of Flexible Manufacturing Systems, *The FMS Magazine*, **2**(2): 114-117.

Cassandras, C. G. Lafortune, S. (2008). *Introduction to Discrete Event Systems*, Springer Science Bussness Media. Estados Unidos da América.

Chan. F.T.S. Chan, H. K. (2004). A comprehensive survey and future trend of simulation study on FMS scheduling, *Journal of Intelligent Manufaturing*. **15**(1): 87-102.

Chen, Y. L. Lin, F. (2000). Modeling of Discrete Event Systems Using Finite State Machines with Parameters, *International Conference on Control Applications*, Anchorage, Estados Unidos da América, pp. 941-946.

Constain, N. B. P. (2014). *Integração de Sistemas SCADA com Implementação de Controle Supervisório em CLP para Sistemas de Manufatura*, Dissertação de mestrado. Universidade Federal de Santa Catarina, Florianópolis, Brasil.

Costa, T. A. (2012). *Metodologia CSO – Controle Supervisório e Otimização Aplicada a Resolução de Problemas de Sequenciamento de tarefas em Sistemas Flexíveis de Manufatura*, Tese de Doutorado. Universidade Federal de Santa Catarina. Florianópolis, Brasil.

Croce, F. D. Tadei, R. Vota, G. (1995). A Genetic Algorithm for the job-shop Problem, *Computers & Operations Research* **22**(1):15-24.

Cury, J. E. R. (2001). Teoria do Controle Supervisório de Sistemas a Eventos Discretos. *V Simpósio Brasileiro de Automação Inteligente* (Minicurso), Canela, Brasil.

Das, G. Gupta, M. Nagar, S.K. (2014) Discrete Event Systems Modelling and Supervisory Control. *Advance in Eletronic and Electring Engineering* **4**(6): 561-570.

Fabian, M. Hellgren, A. PLC-based implementation of supervisory control for discrete event systems, *Proceedings of the 37th IEEE Conference on Decision and Control*, Tampa, Estados Unidos da América, pp. 3305-3310.

Falkenauer, E. Bouffouix, S. (1991). A Genetic Algorithm for Job-Shop, *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, Sacramento, Estados Unidos da América, pp. 824-829.

Gao, H. Kwong, S. Fan, B. Wang, R. (2014). A Hybrid Particle-Swarm Tabu Search Algorithm for Solving Job Shop Scheduling Problems, *IEEE Transactions in Industrial Informatics* **10**(4): 2044-2054.

Glynn, P. W. (1989). A GSMP formalism for discrete event systems, *Proceedings of the IEEE* **77**(1): 14-23.

Groover, M. P. (2011). *Automação industrial e sistemas de manufatura*, Pearson Education do Brasil.

Hill, R.C. Tilbury, D. M. (2006). Modular Supervisory Control of Discrete Event Systems with Abstraction and Incremental Construction, *International Workshop on Discrete Event Systems*, Ann Arbor, Estados Unidos da América, pp. 399-406.

Ho, N. B. Tay, J. C. Loi, E.M-K. (2007). An effective architecture for learning and evolving flexible job-shop schedules, *European Journal of Operational Research* **179**(2):316-333.

Hoitomt, D. J. Luh, P. B. Pattipatti, K. R. (1993). A practical approach to job-shop scheduling problems, *IEEE transactions on Robotic and Automation* **9**(1): 1-13.

Holland, J. H. (1975). *Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence*, MI: University of Michigan Press, Estados Unidos da América.

Holsapple, C. W., Jacob, V. S., Pakath, R., & Zaveri, J. S. (1993). A Genetic-based Hybrid Scheduler for Generating Static Schedules in Flexible Manufacturing Contexts, *IEEE Transactions on Systems, Man, and Cybernetics* **23**(4): 953-972.

Lathi, B. P. (2007). *Sinais e Sistemas Lineares*, Bookman, Porto Alegre, Brasil.

Lee, D. Y. DiCesare, F. (1994). Scheduling Flexible Manufacturing Systems Using Petri Nets Heuristic Search, *IEEE Transactions Robotic and Automation* **10**(2): 123-132.

Lopes, Y. K. (2012). *Integração dos Níveis MES, SCADA e Controle da Planta de Manufatura com Base na Teoria de Linguagens e Autômatos*, Dissertação de mestrado, Universidade Estadual de Santa Catarina, Florianópolis, Brasil.

Maia, C. A. Mendes, R. S. Lüders, R. Hardouim, L. (2005). Estratégias de Controle por Modelo de Referência de Sistemas à Eventos Discretos Max-Plus Lineares, *SBA: Controle & Automação Sociedade Brasileira de Automática* **16**(3): 263-278.

Mitchell, M. (1998). *An introduction to genetic algorithms*, MIT press, Estados Unidos da América.

Moller, P. (1986). Introducing real time in the algebraic theory of finite automata, *Relatório Técnico*, International Institute for Applied Systems Analysis. Luxemburgo, Austria.

Nakano, R., & Yamada, T. (1991). Conventional genetic algorithm for job shop problems, *ICGA* **91**(1): 474-479.

Nauom, L. B. Boel, R. Bongarts, L. Schutter, B. Peng, Y. Valckenaer, P. Vandewalle, J. Wertz, V. (1995). Methodologies for Discrete Event Dynamic Systems: A survey, *Relatório*

Técnico, Department of Electrical Engineering, Katholieke Universiteit te Leuven, Leuven, Bélgica. Pena, P. N. (2007). *Verificação de Conflito na Supervisão de Sistemas Concorrentes Usando Abstrações*, Tese de doutorado, Universidade Federal de Santa Catarina, Florianópolis, Brasil.

Pena, P. N. Costa, T. A. Silva, R. S. Takahashi, R. H. C. (2015). Control of Flexible Manufacturing System under model uncertainty using Supervisory Control Theory and evolutionary computation schedule synthesis, *Journal Information Sciences: an International Journal* **329**(1): 491-502.

Pena, P. N. Cury, J. E. R. Cunha, A. E. C. Lafortune, S. (2010). Metodologia e Ferramenta de Apoio ao Teste de não-conflito no Controle Modular de Sistemas a Eventos Discretos, *SBA Controle & Automação* **21**(1): 58-68.

Penha, D. Queiroz, M. Cury, J. (2011). Optimal Scheduling of a Repair Shipyard Based on Supervisory Control Theory. *IEEE Conference on Automation Science and Engineering*, Trieste, Itália, pp. 39-44.

Pinha, D. C. (2004). *Escalonamento ótimo baseado na teoria de controle supervisório aplicado a um estaleiro de reparo naval*, Dissertação de mestrado, Universidade Federal de Santa Catarina, Florianópolis, Brasil.

Queiroz, M. H. (2000). *Controle Supervisório Modular de Sistemas de Grande Porte*, Dissertação de mestrado, Universidade Federal de Santa Catarina, Florianópolis, Brasil.

Queiroz, M. H. Cury, J. E. R. (2002). Controle Supervisório Modular de Sistemas de Manufatura, *SBA Controle & Automação Sociedade Brasileira de Automática* **13**(2): 123-133.

Queiroz, M. H. Cury, J. E. R. (2002). Synthesis and implementation of local modular supervisory control for a manufacturing cell, *Sixth International Workshop on Discrete Event Systems*, Zaragoza, Espanha, pp. 377-382.

Ramadge, P. J. G. Wonham, W. M. (1987). Supervisory Control of a class of discrete event process, *SIAM Journal of Control and Optmization* **25**(1): 206-230.

Ramadge, P. J. G. Wonham, W. M. (1989).The Control of Discrete Event System, *Proceedings of the IEEE* **77**(1):81-98.

Ramasesh, R. (1990). Dynamic job-shop scheduling: a survey of simulation research, *Omega* **18**(1): 43-57.

Sakurada, D. Miyake, I. (2009) Aplicação de simuladores de eventos discretos no processo de modelagem de sistemas de operações de serviços, *Gestão & Produção* **16**(1):25-43.

Santos, G. A pirâmide da Automação Industrial. Automação Industrial, 2012. Acesso em: 8 de maio de 2018. Disponivel em: <https://www.automacaoindustrial.info/a-piramide-da-automacao-industrial/>.

Schmidt, K. Moor, T. Perk, S. (2008). Nonblocking hierarchical control of decentralized discrete event systems, *IEEE Transactions on Automatic Control* **53**(10):2252-2265

Shi, L. Pan, Y. (2005). An Efficient Search Method for Job-Shop Scheduling Problems. *IEEE transactions on automation science and engineering* **2**(1): 73-77.

Silva, R. Oliveira, A. Pena, P. Takahashi, R. (2011). Algoritmo Clonal para Job shop scheduling com controle supervisório, *X SBAI – Simpósio Brasileiro de Automação Inteligente*, São João del Rei, pp. 1376:1381.

Silva, Y. G. Queiroz, M. H. Cury, J. E. R. (2010). Síntese e Implementação de Controle Supervisório Modular Local para um Sistema de AGV, *XVIII Congresso Brasileiro de Automática*, Bonito, Brasil, pp. 2808:2815.

Trivelato, G. C. (2003) Técnicas de Modelagem e Simulação de Sistemas Dinâmicos, Relatório Técnico, INPE, São Jose dos Campos.

Vieira, A. D.; Santos, E. A. P.; Queiroz, M. H.; Leal, A. B.; Neto, A. D. P.; Cury, J. E. R. (2017). A Method for Implementation of Supervisory Control of Discrete Event Systems, *IEEE Transactions on Control Systems Technology*, **25**(1): 175-191.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and computing*, **4**(1): 65:85.

Wong-Toi, H. Hoffman, G. (1991). The control of dense real-time discrete event systems. *In Proceedings of 30th IEEE Conference on Decision and Control*, Brighton, Inglaterra, pp 1527-1528.

Apêndice A

Código de implementação do *script de-coding*

O algoritmo abaixo realiza a decodificação dos genes de um cromossomo codificado através da representação baseada em operação para uma sequência de eventos da Teoria de Controle Supervisório (TCS).

```
function [seqeventos]=decoding(cromossomo, npecas1, npecas2)

tamanhocromossomo=length(cromossomo);
tabeladeeventospeca1 = [string("withmat1"), string("getmat1"),
string("endm1"),     string("withoutmat1");
                        string(""),          string("Turn"),
string("endturn"),   string("");
                        string(""),          string("Turnon1"),
string("endt11"),    string("");
                        string(""),          string("Turn"),
string("endturn"),   string("");
                        string(""),          string("Turnon"),
string("endt2"),     string("");
                        string(""),          string("getp1"),
string("endp1"),     string("")];

tabeladeeventospeca2 = [string("withmat2"), string("getmat2"),
string("endm2"),     string("withoutmat2");
                        string(""),          string("Turn"),
string("endturn"),   string("");
                        string(""),          string("Turnon2"),
string("endt12"),    string("");
                        string(""),          string("getp2"),
string("endp2"),     string("")];


caso1=[string("Turnon1"), string("Turnon"), string("endt11"),
string("endt2")];
  caso2=[string("Turnon2"), string("Turnon"), string("endt12"),
string("endt2")];
```

```
  seqeventos1="";

if(npecas1>npecas2)
  npecas=2*npecas1;
else
  npecas=2*npecas1;
end

contador2=zeros(1,npecas);
aux1=cromossomo(1);
contador2(aux1)=1;

if(mod(aux1,2)==0)
  seqeventos2=tabeladeeventospeca2(1,:);
else
  seqeventos2=tabeladeeventospeca1(1,:);
end

if(tamanhocromossomo>1)
  for i=2:tamanhocromossomo
    aux2=cromossomo(i);
    aux3=cromossomo(i-1);
    if(aux2~=0)
      contador2(aux2)=contador2(aux2)+1;
      if(mod(aux2,2)==0)
        if(aux2~=aux3)
          if(contador2(aux2)==2)
            if(mod(aux3,2)==0)
              if(contador2(aux3)==2)
                seqeventos2=seqeventos2;
              else
                seqeventos2=[seqeventos2 tabeladeeventospeca2(conta-
dor2(aux2),:)];
              end
            else
              if(contador2(aux3)==2 || contador2(aux3)==4)
                seqeventos2=seqeventos2;
              else
                seqeventos2=[seqeventos2 tabeladeeventospeca2(conta-
dor2(aux2),:)];
              end
            end
          else
            if((contador2(aux2)==5 && contador2(aux3)==3)||(con-
tador2(aux2)==3 && contador2(aux3)==5))
              if(mod(aux2,2)==0)
                seqeventos2=[seqeventos2(1:(end-2)) caso2];
              else
                if(mod(aux3,2)==0)
                  seqeventos2=[seqeventos2(1:(end-2)) caso2];
                else
                  seqeventos2=[seqeventos2(1:(end-2)) caso1];
                end
              end
            else
              seqeventos2=[ seqeventos2 tabeladeeventospeca2(conta-
dor2(aux2),:)];
            end
          end
        else
```

```
          seqeventos2=[seqeventos2 tabeladeeventospeca2(conta-
dor2(aux2),:)];
        end
      else
        if(aux2~=aux3)
          if(contador2(aux2)==2 || contador2(aux2)==4)
            if(mod(aux3,2)==0)
              if(contador2(aux3)==2)
                seqeventos2=seqeventos2;
              else
                seqeventos2=[seqeventos2 tabeladeeventospeca1(conta-
dor2(aux2),:)];
              end
            else
              if(contador2(aux3)==2 || contador2(aux3)==4)
                seqeventos2=seqeventos2;
              else
                seqeventos2=[seqeventos2 tabeladeeventospeca1(conta-
dor2(aux3),:)];
              end
            end
          else
            if((contador2(aux2)==5 && contador2(aux3)==3)|| con-
tador2(aux3)==5 && contador2(aux2)==3)
              if(mod(aux2,2)==0)
                seqeventos2=[seqeventos2(1:(end-2)) caso2];
              else
                if(mod(aux3,2)==0)
                  seqeventos2=[seqeventos2(1:(end-2)) caso2];
                else
                  seqeventos2=[seqeventos2(1:(end-2)) caso1];
                end
              end
            else
              seqeventos2=[seqeventos2 tabeladeeventospeca1(conta-
dor2(aux2),:)];
            end
          end
        else
          seqeventos2=[seqeventos2 tabeladeeventospeca1(conta-
dor2(aux2),:)];
        end
      end
    end
  end
end
aux2=length(seqeventos2);

for i=1:aux2
  aux3=seqeventos2(i);
  if(aux3~="")
    seqeventos1=[seqeventos1 seqeventos2(i)];
  end
end
  seqeventos=seqeventos1(2:end);

end
```

Apêndice B

Código de Implementação do *script* controlesupervisorio

O algoritmo abaixo realiza a comprovação da legalidade pela Teoria da Controle Supervisório da sequência de eventos decodificada do cromossomo pelo *script decoding*. Ele retorna apenas se a sequência de eventos é legal ou não.

```
function [legalidade]=controlesupervisorio3(seq)
tamanhocontrolador=length(seq);
legal=zeros(1,tamanhocontrolador);

%Setar os estados iniciais dos supervisores
estadoatualsp1=0;
estadoatualsp2=0;
estadoatualsp3=0;
estadoatualsp4=0;
estadoatualsp5=0;
estadoatualsp6=0;
estadoatualsp7=0;
estadoatualsp8=0;
estadoatualsp9=0;
estadoatualsp10=0;
estadoatualsp11=0;
estadoatualsp12=0;
estadoatualsp13=0;
estadoatualsp14=0;
estadoatualsp15=0;
estadoatualsp16=0;
estadoatualsp17=0;

%Estados marcados
estadomarcadosp1_1=0;
estadomarcadosp1_2=10;
estadomarcadosp2_1=0;
estadomarcadosp2_2=10;
estadomarcadosp3_1=0;
estadomarcadosp3_2=10;
estadomarcadosp4_1=0;
estadomarcadosp4_2=15;
estadomarcadosp5_1=0;
```

```
estadomarcadosp5_2=10;
estadomarcadosp6=0;
estadomarcadosp7=0;
estadomarcadosp8=0;
estadomarcadosp9_1=0;
estadomarcadosp9_2=46;
estadomarcadosp9_3=76;
estadomarcadosp9_4=106;
estadomarcadosp10=0;
estadomarcadosp11_1=0;
estadomarcadosp11_2=46;
estadomarcadosp11_3=76;
estadomarcadosp11_4=106;
estadomarcadosp12=0;
estadomarcadosp13=0;
estadomarcadosp14_1=0;
estadomarcadosp14_2=30;
estadomarcadosp15_1=0;
estadomarcadosp15_2=12;
estadomarcadosp15_3=24;
estadomarcadosp15_4=36;
estadomarcadosp16_1=0;
estadomarcadosp16_2=30;
estadomarcadosp17_1=0;
estadomarcadosp17_2=20;

for i=1:tamanhocontrolador
 evento=seq(i);
 %Inicialização do teste
  xsupervisor1=0;
  xsupervisor2=0;
  xsupervisor3=0;
  xsupervisor4=0;
  xsupervisor5=0;
  xsupervisor6=0;
  xsupervisor7=0;
  xsupervisor8=0;
  xsupervisor9=0;
  xsupervisor10=0;
  xsupervisor11=0;
  xsupervisor12=0;
  xsupervisor13=0;
  xsupervisor14=0;
  xsupervisor15=0;
  xsupervisor16=0;
  xsupervisor17=0;

  %Testar Supervisor 1
     if(estadoatualsp1==0 && evento=="getp2") %1
         estadoatualsp1=1;
         xsupervisor1=1;
     end
     if(estadoatualsp1==0 && evento=="getp1") %2
         estadoatualsp1=2;
         xsupervisor1=1;
     end
     if(estadoatualsp1==0 && evento=="getmat2") %3
         estadoatualsp1=4;
         xsupervisor1=1;
     end
```

```
if(estadoatualsp1==0 && evento=="withmat1") %4
    estadoatualsp1=15;
    xsupervisor1=1;
end
if(estadoatualsp1==1 && evento=="endp2") %5
    estadoatualsp1=0;
    xsupervisor1=1;
end
if(estadoatualsp1==1 && evento=="withmat1") %6
    estadoatualsp1=16;
    xsupervisor1=1;
end
if(estadoatualsp1==2 && evento=="endp1") %7
    estadoatualsp1=0;
    xsupervisor1=1;
end
if(estadoatualsp1==2 && evento=="withmat1") %8
    estadoatualsp1=17;
    xsupervisor1=1;
end
if(estadoatualsp1==3 && evento=="endm1") %9
    estadoatualsp1=0;
    xsupervisor1=1;
end
if(estadoatualsp1==3 && evento=="withmat1") %10
    estadoatualsp1=18;
    xsupervisor1=1;
end
if(estadoatualsp1==4 && evento=="endm2") %11
    estadoatualsp1=0;
    xsupervisor1=1;
end
if(estadoatualsp1==4 && evento=="withmat1") %12
    estadoatualsp1=19;
    xsupervisor1=1;
end
if(estadoatualsp1==5 && evento=="withoutmat1") %13
    estadoatualsp1=0;
    xsupervisor1=1;
end
if(estadoatualsp1==5 && evento=="getp2") %14
    estadoatualsp1=6;
    xsupervisor1=1;
end
if(estadoatualsp1==5 && evento=="getp1") %15
    estadoatualsp1=7;
    xsupervisor1=1;
end
if(estadoatualsp1==5 && evento=="getmat2") %16
    estadoatualsp1=9;
    xsupervisor1=1;
end
if(estadoatualsp1==6 && evento=="withoutmat1") %17
    estadoatualsp1=1;
    xsupervisor1=1;
end
if(estadoatualsp1==6 && evento=="endp2") %18
    estadoatualsp1=5;
    xsupervisor1=1;
end
if(estadoatualsp1==7 && evento=="withoutmat1") %19
```

```
    estadoatualsp1=2;
    xsupervisor1=1;
end
if(estadoatualsp1==7 && evento=="endp1") %20
    estadoatualsp1=5;
    xsupervisor1=1;
end
if(estadoatualsp1==8 && evento=="withoutmat1") %21
    estadoatualsp1=3;
    xsupervisor1=1;
end
if(estadoatualsp1==8 && evento=="endm1") %22
    estadoatualsp1=5;
    xsupervisor1=1;
end
if(estadoatualsp1==9 && evento=="withoutmat1") %23
    estadoatualsp1=4;
    xsupervisor1=1;
end
if(estadoatualsp1==9 && evento=="endm2") %24
    estadoatualsp1=5;
    xsupervisor1=1;
end
if(estadoatualsp1==10 && evento=="getmat1") %25
    estadoatualsp1=3;
    xsupervisor1=1;
end
if(estadoatualsp1==10 && evento=="getp2") %26
    estadoatualsp1=11;
    xsupervisor1=1;
end
if(estadoatualsp1==10 && evento=="getp1") %27
    estadoatualsp1=12;
    xsupervisor1=1;
end
if(estadoatualsp1==10 && evento=="getmat2") %28
    estadoatualsp1=14;
    xsupervisor1=1;
end
if(estadoatualsp1==10 && evento=="withmat1") %29
    estadoatualsp1=15;
    xsupervisor1=1;
end
if(estadoatualsp1==11 && evento=="endp2") %30
    estadoatualsp1=10;
    xsupervisor1=1;
end
if(estadoatualsp1==11 && evento=="withmat1") %31
    estadoatualsp1=16;
    xsupervisor1=1;
end
if(estadoatualsp1==11 && evento=="endp1") %32
    estadoatualsp1=10;
    xsupervisor1=1;
end
if(estadoatualsp1==12 && evento=="withmat1") %33
    estadoatualsp1=17;
    xsupervisor1=1;
end
if(estadoatualsp1==13 && evento=="endm1") %34
    estadoatualsp1=10;
```

```
    xsupervisor1=1;
end
if(estadoatualsp1==13 && evento=="withmat1") %35
    estadoatualsp1=18;
    xsupervisor1=1;
end
if(estadoatualsp1==14 && evento=="endm2") %36
    estadoatualsp1=10;
    xsupervisor1=1;
end
if(estadoatualsp1==14 && evento=="withmat1") %37
    estadoatualsp1=19;
    xsupervisor1=1;
end
if(estadoatualsp1==15 && evento=="getmat1") %38
    estadoatualsp1=8;
    xsupervisor1=1;
end
if(estadoatualsp1==15 && evento=="withoutmat1") %39
    estadoatualsp1=10;
    xsupervisor1=1;
end
if(estadoatualsp1==15 && evento=="getp2") %40
    estadoatualsp1=16;
    xsupervisor1=1;
end
if(estadoatualsp1==15 && evento=="getp1") %41
    estadoatualsp1=17;
    xsupervisor1=1;
end
if(estadoatualsp1==15 && evento=="getmat2") %42
    estadoatualsp1=19;
    xsupervisor1=1;
end
if(estadoatualsp1==16 && evento=="withoutmat1") %43
    estadoatualsp1=11;
    xsupervisor1=1;
end
if(estadoatualsp1==16 && evento=="endp2") %44
    estadoatualsp1=15;
    xsupervisor1=1;
end
if(estadoatualsp1==17 && evento=="withoutmat1") %45
    estadoatualsp1=12;
    xsupervisor1=1;
end
if(estadoatualsp1==17 && evento=="endp1") %46
    estadoatualsp1=15;
    xsupervisor1=1;
end
if(estadoatualsp1==18 && evento=="withoutmat1") %47
    estadoatualsp1=13;
    xsupervisor1=1;
end
if(estadoatualsp1==18 && evento=="endm1") %48
    estadoatualsp1=15;
    xsupervisor1=1;
end
if(estadoatualsp1==19 && evento=="withoutmat1") %49
    estadoatualsp1=14;
    xsupervisor1=1;
```

```
    end
    if(estadoatualsp1==19 && evento=="endm2") %50
        estadoatualsp1=15;
        xsupervisor1=1;
    end
    if(estadoatualsp1==estadomarcadosp1_1 && evento~="getmat1")
        xsupervisor1=1;
    end
```

(Para todos os 17 supervisores é realizada essa operação)

```
     if(xsupervisor1==1 && xsupervisor2==1 && xsupervisor3==1 && xsupervi-
sor4==1 && xsupervisor5==1 && xsupervisor6==1 && xsupervisor7==1 && xsuper-
visor8==1 && xsupervisor9==1 && xsupervisor10==1 && xsupervisor11==1 &&
xsupervisor12==1 && xsupervisor13==1 && xsupervisor14==1 && xsupervi-
sor15==1 && xsupervisor16==1 && xsupervisor17==1)
        legal(i)=1;
    else
        legal(i)=0;
    end
end
aux1=0;
for i=1:tamanhocontrolador
    if(legal(i)==0)
        aux1=aux1+1;
    else
        aux1=aux1;
    end

end

if aux1>0
    legalidade=0;
else
    legalidade=1;
end

end
```