DOI: 10.48011/asba.v2i1.1177

Planejamento da produção em sistemas de manufatura modelados como Processo de Decisão Markoviano

Matheus C. dos Santos [*] **Lucas V. R. Alves** [**] **Patrícia N. Pena** [***]

[*] *Escola de Engenharia, Universidade Federal de Minas Gerais, MG, (e-mail: matheuscascalho@ufmg.br).*
[**] *Colégio Técnico, Universidade Federal de Minas Gerais, MG (e-mail: lucasvra@ufmg.br)*
[***] *Departamento de Engenharia Eletrônica, Universidade Federal de Minas Gerais, MG (e-mail: ppena@ufmg.br)*

Abstract: Task planning is an important area of study as it is directly linked to increased productivity in industrial systems, through metrics such as makespan and throughput. This article aims to present a method for creating an event selection policy that maximizes the accumulated parallelism in processes modeled by automata. This policy seeks to produce sequences of reduced makespan, for the case of finite sequences of events, or of high throughput, for the case of continuous production. In this approach, we model the problem as a Markov Decision Process, proposing a modification in the Value Iteration Algorithm.

Resumo: O planejamento de tarefas é uma importante área de estudo pois está diretamente ligada ao aumento de produtividade em sistemas industriais, por meio de métricas como makespan e throughput. Esse artigo tem por objetivo apresentar um método de criação de política de seleção de eventos que maximize o paralelismo acumulado em processos modelados por autômatos. Essa política busca produzir sequências de *makespan* reduzido, para o caso de sequências finitas de eventos, ou de alto *throughput*, para o caso de produção contínua. Nessa abordagem modelamos o problema como um Processo de Decisão Markoviano, propondo uma modificação no Algoritmo de Iteração de Valor

Keywords: Discrete Event Systems, Supervisory Control Theory, Production Planning, Task Schedulling, Markov Decision Process

Palavras-chaves: Sistemas a Eventos Discretos, Teoria de Controle Supervisório, Planejamento da Produção, Escalonamento de Tarefas, Processo de Decisão Markoviano

1. INTRODUÇÃO

Produtividade é a relação entre o volume de bens produzidos e as horas de trabalho necessárias para fabricar esses bens. Essa medida é de grande importância para a sociedade, uma vez que gera impacto sobre os custos de produção, lucro de empresas e preço ao consumidor final. O aumento de produtividade é, portanto, um objetivo a ser alcançado por empresas em tempos de prosperidade e de crise. No primeiro caso, empresas desejam atingir esse objetivo a fim de manter sua competitividade no mercado e no segundo, o mesmo é desejado a fim de sustentar economicamente a instituição.

Se o tempo necessário para realizar um determinado processo fabril for reduzido, os ativos de uma empresa podem ser utilizados para expandir a produção no intervalo de tempo economizado. Por essa razão o tempo é um recurso valioso e a escolha de boas sequências operacionais, como aquelas que minimizam o *makespan* (tempo de produção) ou maximizam o *throughput* (taxa de produção), é de grande importância em manufaturas. Dito isto, o planejamento eficiente da produção e técnicas de escalonamento de tarefas são a chave para responder à demanda por eficiência produtiva.

A maioria dos atuais sistemas industriais podem, em certo nível de abstração, ser modelados como Sistemas a Eventos Discretos (SED), onde estados de um sistema mudam conforme ocorrem eventos. Essa abordagem é interessante, uma vez que permite aplicar um controle de alto nível aos mais complexos sistemas industriais. Nesse trabalho modelamos Sistemas a Eventos Discretos utilizando autômatos e linguagens, permitindo a distinção de sistemas controlados (plantas) e controladores (supervisores).

O escalonamento de tarefas de um sistema diz respeito à alocação de recursos ao longo do tempo em processos de produção, utilizando algum critério de otimização. Esse problema pode ser dividido em duas classes: problemas de escalonamento determinístico (também conhecido como modelo de escalonamento preditivo) e problemas de escalonamento estocástico. Um escalonamento é dito determinístico quando o sistema é tão previsível que um modelo pode ser utilizado para gerar o comportamento exato do

sistema. Já um escalonamento é dito estocástico quando seus eventos não podem ser previstos com exatidão, sendo necessário o uso de técnicas estatísticas em sua criação.

Ao longo dos anos, diversas abordagens surgiram na literatura para tratar do problema de escalonamento de tarefas, como programação matemática (Schrijver, 1986), redes de Petri (López-Mellado et al., 2005; Luo et al., 2014; Wang et al., 2019; Mejía et al., 2017), autômatos temporizados (Abdeddaïm et al., 2006), verificação formal (Herzig et al., 2014; Malik and Pena, 2018), entre outros.

No contexto de teoria de controle supervisório (TCS), existem diversas abordagens para problema de escalonamento (Kobetski and Fabian, 2006; Pinha et al., 2011; Su et al., 2012), normalmente focadas em minimização de *makespan*, o tempo total para a produção de um lote de determinado produto. Uma importante vantagem ao representar SEDs utilizando a linguagem de autômatos é o acesso à Teoria de Controle Supervisório, um *framework* onde plantas e especificações podem ser utilizadas para gerar supervisores ótimos, no sentido de serem minimamente restritivos.

Dado seu caráter combinatório, o número de estados dos autômatos que representam o comportamento em malha fechada de sistemas industriais de médio e grande porte é elevado. Isso confere uma alta complexidade ao processo de criação de sequências de eventos ótimas, tornando o o cálculo impraticável em problemas reais, de forma que, usualmente o problema é tratado por meio de heurísticas. Uma opção de resolução de problemas como esse é a criação de políticas que relacionam cada estado a um evento de forma a atender um determinado critério.

Processos de Decisão Markovianos (MDP - *Markov Decision Process*) são processos estocásticos cujo estado do processo no futuro depende unicamente do estado atual do processo e da decisão escolhida no presente. Esse modelo é útil para gerar políticas de estados de sistemas, sendo aplicável à pesquisa em controle de espaçonaves (Tipaldi and Glielmo, 2017), gestão de problemas ambientais (Bassey and Chigbu, 2012), redes de transporte (Guettiche and Kheddouci, 2019), entre outros.

A política de eventos em um MDP é criada com base em probabilidade de ocorrência de eventos e recompensas associadas ao estado de destino, sendo a recompensa uma característica intrínseca ao estado. No contexto de sistemas industriais, o paralelismo (Alves et al., 2015b) é uma propriedade dos estados e o paralelismo acumulado pode ser usado como métrica do desempenho do sistema. Se para a produção de uma mesma quantidade de produtos é necessária uma mesma quantidade de tarefas realizadas pelas máquinas, usualmente a sequência que possuir mais tarefas sendo executadas em paralelo terá melhores resultados de makespan. O objetivo deste trabalho é modelar um sistema industrial em malha fechada, controlado pela Teoria de Controle Supervisório, como um processo de decisão markoviano e obter uma política de eventos que maximize o paralelismo acumulado, sendo esta uma política sub-ótima de minimização de makespan.

Na seção 2 apresentamos alguns conceitos preliminares, as principais definições e o modelo de paralelismo. Na seção 3, o algoritmo proposto para a criação de uma política de eventos é apresentado. Na seção 4 apresentamos o estudo de caso utilizando o Sistema Flexível de Manufatura. Conclusões e comentários finais são expostos na seção 5.

2. PRELIMINARES

Seja Σ um conjunto finito não vazio de eventos, chamado de alfabeto. O comportamento de um SED é modelado como cadeias sobre Σ. O Fechamento Kleene Σ^* é o conjunto de todas as cadeias que podem ser construídas utilizando os símbolos do alfabeto Σ, incluindo a cadeia vazia ϵ (Wonham, 2014). Um subconjunto $L \subseteq \Sigma^*$ é chamado linguagem. A concatenação de cadeias $s, u \in \Sigma^*$ é representada como su.

Uma cadeia $s \in \Sigma^*$ é chamada *prefixo* de $t \in \Sigma^*$, escrito $s \leq t$, se existe $u \in \Sigma^*$ tal que $su = t$. O prefixo fechamento $\overline{L}$ de uma linguagem $L \subseteq \Sigma^*$ é o conjunto de todos os prefixos de cadeias em L, isto é, $\overline{L} = \{\, s \in \Sigma^* \mid s \leq t \text{ para algum } t \in L \,\}$.

***Definição** 1.* Um autômato finito determinístico é uma 5-tupla $G = (Q, \Sigma, \delta, q_0, Q_m)$ onde Q é um conjunto finito de estados, Σ é um conjunto finito de eventos (alfabeto), $\delta : Q \times \Sigma \to Q$ é a função de transição, $q_0 \in Q$ é o estado inicial e $Q_m \subseteq Q$ é o conjunto de estados marcados.

A função de transição pode ser estendida para reconhecer cadeias sobre Σ^* como $\delta(q, \sigma s) = q'$ se $\delta(q, \sigma) = x$ e $\delta(x, s) = q'$.

As linguagens gerada e marcada são, respectivamente, $\mathcal{L}(G) = \{s \in \Sigma^* | \delta(q_0, s) = q' \wedge q' \in Q\}$ e $\mathcal{L}_m(G) = \{s \in \Sigma^* | \delta(q_0, s) = q' \wedge q' \in Q_m\}$.

A função de eventos ativos, definida por $\Gamma : Q \to 2^{\Sigma}$, é, dado um estado q, o conjunto de eventos $\sigma \in \Sigma$ para os quais $\delta(q, \sigma)$ é definido.

***Definição** 2.* Seja $G_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, Q_{m1})$ e $G_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, Q_{m2})$. O produto síncrono de G_1 e G_2 é:

$$G_{12} = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{12}, (q_{01}, q_{02}), Q_{m1} \times Q_{m2})$$

onde

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

e $\Gamma_{1||2}(q_1, q_2) = [\Gamma_1(q_1) \cap \Gamma_2(q_2)] \cup [\Gamma_1(q_1) \backslash \Sigma_2] \cup [\Gamma_2(q_2) \backslash \Sigma_1]$.

A Teoria de Controle Supervisório é um método formal, baseado na teoria de linguagens e autômatos, para o cálculo sistemático de supervisores. O sistema a ser controlado é chamado *planta*, o controlador é chamado de *supervisor* e o problema de controle é encontrar um supervisor que garanta especificações de desempenho e segurança de maneira minimamente restritiva.

A planta é modelada por um autômato $G = (Q, \Sigma, \delta, q_0, Q_m)$ e $\Sigma = \Sigma_c \cup \Sigma_{nc}$ onde Σ_c é o conjunto de eventos controláveis, que podem ser desabilitados por um agente externo, e Σ_{nc} é o conjunto de eventos não controláveis, que não podem ser desabilitados por um agente externo, como o supervisor. A planta representa o modelo lógico de um SED, o comportamento do sistema sem nenhuma ação de controle. O papel de um supervisor S é regular o comportamento da planta e alcançar um comportamento seguro

K desabilitando eventos controláveis, impedindo assim a ocorrência de sequências incorretas de eventos, atendendo as demandas de alocação correta de recursos.

As linguagens gerada e marcada de uma planta G sobre a ação de um supervisor S são, respectivamente, $\mathcal{L}(S/G)$ e $\mathcal{L}_m(S/G) \subseteq \mathcal{L}(S/G)$. Um supervisor S é dito não bloqueante quando $\overline{\mathcal{L}_m(S/G)} = \mathcal{L}(S/G)$.

A partir daqui, chama-se supervisor um autômato S tal que $S = S||G$, ou seja, a dinâmica do supervisor engloba a dinâmica da planta sob controle.

2.1 Modelagem do Paralelismo

Uma vez que a transição entre estados de um autômato se dá pela ocorrência de eventos que representam o início ou o fim de tarefas, podemos modelar tarefas como uma propriedade intrínseca de estados do autômato. Nesse contexto, um determinado estado pode ter zero ou mais tarefas sendo executadas. Usualmente, quando modelamos cada componente do sistema, associamos 0 tarefas aos estados onde a máquina está ociosa e 1 tarefa aos estados onde máquina está trabalhando. Em casos especiais, quando a máquina possui um paralelismo intrínseco, o número de tarefas associado a cada estado pode ser qualquer valor inteiro não negativo (Alves et al., 2015a).

Dado que queremos relacionar o conjunto de estados do autômato com o número de tarefas em cada estado, podemos representar essa relação por uma função chamada função de tarefas ativas.

***Definição** 3.* Seja $G = (Q, \_\_, \_\_, \_\_, \_\_)$ um autômato. a função de tarefas ativas associada à G, definida como $f_{ta} : Q \to \mathbb{Z}^*$, é tal que, para cada estado $q \in Q$, associa um número inteiro não negativo de tarefas.

Usualmente é mais simples estabelecer o número de tarefas nos autômatos das plantas, ao invés do autômato do supervisor, tornando então necessário a definição de uma nova função de tarefas ativas para a composição de dois autômatos.

***Definição** 4.* Sejam $G_1 = (Q_1, \_\_, \_\_, \_\_, \_\_)$ e $G_2 = (Q_2, \_\_, \_\_, \_\_, \_\_)$ autômatos e sejam f_{ta_1} e f_{ta_2} as funções de tarefas ativas associadas, respectivamente, à G_1 e G_2. A função de tarefas ativas do autômato $G_{1||2} = (Q_1 \times Q_2, \_\_, \_\_, \_\_, \_\_)$ é definida como:

$$f_{ta_{1||2}}((q_1, q_2)) = f_{ta_1}(q_1) + f_{ta_2}(q_2)$$

onde $q_1 \in Q_1$ e $q_2 \in Q_2$.

Especificações não executam tarefas, visto que são apenas restrições de comportamento, e dessa forma, para todas as especificações, a função de tarefas ativas é sempre zero para todos os estados.

A fim de ilustrar as principais ideias desse artigo, utilizaremos o exemplo da pequena fábrica (Wonham, 2014) ao longo das demais definições.

***Exemplo** 1.* A pequena fábrica é composta por duas máquinas e um *buffer* unitário, assim como mostrado na Figura 1. Para se produzir um produto, é necessário que primeiramente a matéria prima passe pela máquina M_1. Após processada, ela é colocada sobre o *buffer* unitário B até que a máquina M_2 esteja disponível para o processamento. Finalmente, a máquina M_2 processa a matéria prima, finalizando assim o produto.

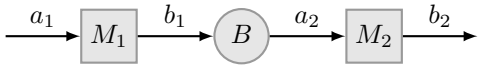

Figura 1. Diagrama da pequena fábrica

A Figura 2 apresenta o supervisor monolítico da pequena fábrica, cada estado é representado por três letras, referindo-se respectivamente ao estado da máquina 1, estado da máquina 2 e ao estado do *buffer*. As máquinas podem estar nos estados I (ociosa) ou W (trabalhando) e o *buffer* pode estar nos estados E (vazio) ou F (cheio). Cada máquina está executando uma tarefa quando se encontra no estado W, ou seja, o estado WWE possui duas tarefas ativas e o estado IIF possui zero tarefas ativas, por exemplo.

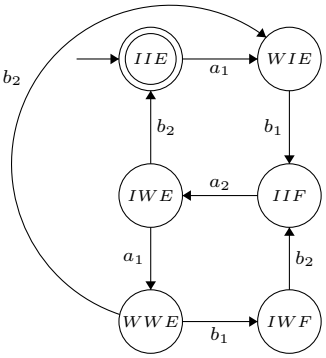

Figura 2. Supervisor monolítico da pequena fábrica

Para se avaliar o paralelismo de uma sequência finita de eventos, é necessária a definição da função acumulativa de tarefas ativas:

***Definição** 5.* Seja $S = (Q, \Sigma, \delta, \_\_, \_\_)$ um supervisor. A função acumulativa de tarefas ativas, $F_{ta} : Q \times \Sigma^* \to \mathbb{Z}^*$, de S é definida como:

$$\begin{cases} F_{ta}(q, \epsilon) & = f_{ta}(q) \\ F_{ta}(q, \sigma s) & = f_{ta}(q) + F_{ta}(\delta(q, \sigma), s) \end{cases}$$

onde $\sigma s \in \Sigma^*$.

Quanto maior o valor da função acumulativa de tarefas ativas determinada sequência apresenta, maior o seu paralelismo. A função acumulativa de tarefas ativas somente deve ser usada para a comparação de sequências de mesmo tamanho.

***Exemplo** 2.* Considerando o supervisor da pequena fábrica (Figura 2), podemos comparar o paralelismo de duas sequências finitas de eventos, a sequência $s_1 = a_1b_1a_2b_2a_1b_1a_2b_2$ e a sequência $s_2 = a_1b_1a_2a_1b_1b_2a_2b_2$ que produzem dois produtos. A função de tarefas ativas é avaliada como:

$$F_{ta}(IIE, s_1) = 4$$
$$F_{ta}(IIE, s_2) = 6$$

Estes cálculos podem ser verificados percorrendo o autômato do supervisor e somando o número de letras W em cada estado visitado. Como $F_{ta}(IIE, s_2)$ é maior que

$F_{ta}(IIE, s_1)$, podemos concluir que s_2 tem maior paralelismo acumulado que s_1.

2.2 Processo de Decisão Markoviano

Como apresentado na seção anterior, diferentes sequências de eventos, embora tenham o mesmo número de eventos e produzam os mesmos produtos, podem possuir desempenhos distintos. Em problemas reais, decidir entre qual o melhor evento a se realizar em determinado estado não é uma tarefa trivial, uma vez que cada decisão imediata tem impacto nas decisões futuras do sistema.

Para tomar decisões em grandes sistemas, onde há uma grande quantidade de estados e eventos, podemos tratar esses sistemas como Processos de Decisão Markoviano. MDPs são processos onde a transição entre estados tem natureza probabilística, é possível observar o estado atual do sistema e interferir no mesmo, ocasionalmente, por meio de ações (Pellegrini and Wainer, 2007). O Exemplo 3 ilustra o impacto de uma decisão imediata sobre o futuro do sistema.

***Exemplo** 3.* Um coelho deve decidir se movimentar para a direita ou para a esquerda. Caso ele se movimente para a direita, ele comerá um brócolis e caso ele se movimente para a esquerda, irá comer uma cenoura, como ilustrado na Figura 3. Uma vez que sua refeição favorita é uma cenoura, ele decide se movimentar para a esquerda.

Figura 3. A seta preta indica a movimentação do coelho.

Em uma outra situação existe um tigre ao lado da cenoura. Se o coelho decidir de movimentar para a esquerda, poderá ser devorado pelo tigre. Portanto, para se proteger, ele decide se movimentar para a direita, comer o brócolis e fugir.

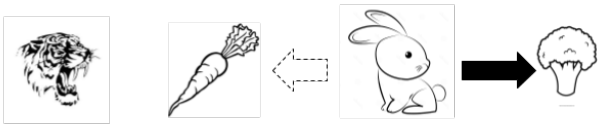

Figura 4. Novo ambiente em que há um tigre ao lado da cenoura

A Figura 5 apresenta um diagrama que resume esse exemplo, atribuindo valores quantitativos para a recompensa de cada configuração do ambiente (estado). A letra abaixo das setas representa cada evento (movimentação do coelho e do tigre).

Modelando o coelho como um agente tomador de decisões, as configurações do ambiente como estados e a movimentação do coelho como ações. Para cada ação e estado de destino existe uma recompensa, no caso o bem estar do coelho. Embora a recompensa imediata de uma transição de estados seja maior que outra, a recompensa da transição futura também irá influenciar na decisão atual. De acordo

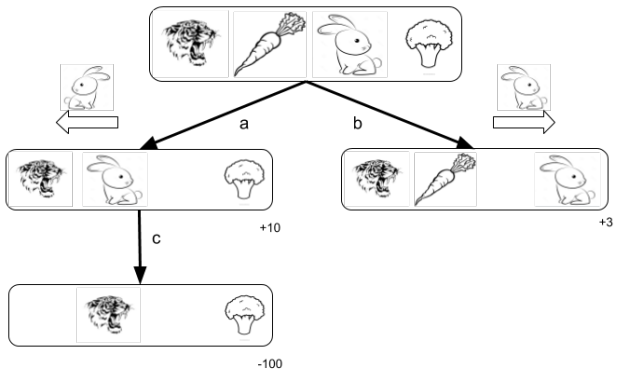

Figura 5. Diagrama das decisões do coelho. O número abaixo de cada estado representa a recompensa daquele estado.

com a Figura 5, ao seguir a ação a (mover para a esquerda), o coelho terá a recompensa $+10$. Em seguida, a única ação possível é a c, levando para um estado onde a recompensa é -100, resultando em uma recompensa final de -90. Optando pelo evento b, a recompensa final será $+3$. Como $+3 \geq -90$, a melhor decisão que o coelho pode tomar é se movimentar para a direita (ação b).

***Definição** 6.* Um processo de decisão markoviano é uma 4-tupla $M = (X, A, P, R)$ onde X é um conjunto de estados, A é um conjunto de ações, P é a função que dá a probabilidade de o sistema passar para um estado $x' \in X$, dado que o sistema está em um estado $x \in X$ e o agente decidiu executar uma ação $a \in A$ (denotada por $P(x'|x, a)$) e R a função que dá a recompensa da transição (x, a, x').

Um importante elemento do processo de decisão markoviano é a definição de uma política que relaciona cada estado a uma ação ótima, no sentido de maximizar a recompensa do sistema. Dado o estado atual do sistema, o tomador de decisão verifica a política (π) para definir qual a melhor ação a ser executada.

***Definição** 7.* A função valor de uma política π para um MDP é uma função $V^\pi : X \to \mathbb{R}$, tal que $V^\pi(x)$ dá o valor esperado da recompensa para esta política, de acordo com o critério de otimalidade. A função valor ótimo para uma política ($V^*(x)$) é calculada por:

$$V^*(x) = \max_a \Sigma_{x'} P(x'|x, a)(R(x, a, x') + \gamma V^*(x'))$$

Existem diversas formas de se encontrar a melhor política de estados e eventos, das quais podemos citar o método de iteração de políticas, programação linear, programação dinâmica, entre outros (Pellegrini and Wainer, 2007).

Dentre os diversos algoritmos que geram políticas para MDP, escolhemos o Algoritmo de Iteração de Valor, um algoritmo que usa programação dinâmica e determina a função valor ótima $V^*(x)$ de cada estado $x \in X$, onde X é o espaço de estados do autômato. Esse algoritmo calcula a função valor para cada estado em n iterações, onde o cálculo de $V(x)$ em cada iteração leva em consideração o cálculo de $V(x)$ da iteração anterior, aplicado um valor de desconto γ. Esse desconto é aplicado a fim de garantir a convergência do método.

Como critério de parada para o algoritmo, usualmente, analisa-se a diferença entre $V_k(x)$ e $V_{k-1}(x)$ em cada

iteração. Neste trabalho foi utilizado como critério de parada $\forall x \in X, |V_k(x) - V_{k-1}(x)| \leq \phi$, onde ϕ é um valor pequeno (Puterman, 2014).

3. RESULTADOS PRINCIPAIS

Um sistema em malha fechada ($S = (Q, \Sigma, \delta, q_0, Q_m)$) pode ser tratado como um processo de decisão markoviano $M = (X, A, P, R)$, onde os estados desse processo X correspondem aos estados do sistema Q, as ações do processo (A) equivalem aos eventos do sistema (Σ) e a função de recompensa do processo (R) equivale à função de tarefas ativas (f_{ta}) do sistema. A função de probabilidade do processo de decisão markoviano é dada por:

$$P(q'|q,\sigma) = \begin{cases} 1 & \text{se } \sigma \in \Sigma_c \wedge \delta(q,\sigma) = q' \\ \dfrac{1}{|\Gamma(q) \cap \Sigma_{nc}|} & \text{se } \sigma \in (\Gamma(q) \cap \Sigma_{nc}) \wedge \exists \sigma' \in \Sigma_{nc} : \delta(q,\sigma') = q' \\ 0 & \text{caso contrário} \end{cases}$$

Ou seja, a probabilidade de um evento controlável acontecer, quando desejado, é 1, contudo, quando deseja-se que um evento não controlável ocorra, consideramos que qualquer um dos eventos não controláveis habilitados no estado pode ocorrer com igual probabilidade, dessa forma, mesmo quando a política define que determinado evento não controlável deve ocorrer, não há garantia de que outro evento não controlável não ocorra antes dele.

Dado que, devido às infactibilidades temporais e restrições à ocorrência de eventos (quantidade de ocorrência de eventos limitada pelo tamanho do lote), nem sempre um evento habilitado em determinado estado pode ocorrer de fato, propomos uma modificação no algoritmo de iteração de valor, tal que, ao invés de mapear para cada estado x a ação que maximiza a função de valor $V(x)$, relacionamos ao estado uma lista de eventos ordenada pela função valor. Dessa forma, caso o o evento que maximiza $V(x)$ esteja infactível, o planejador pode tentar executar o segundo evento da lista e assim por diante, até que algum evento possa ser executado.

O Algoritmo 1 recebe como entradas um processo de decisão markoviano M e um valor ϕ, que é utilizado como critério de parada (A diferença entre $V_k(x)$ e $V_{k-1}(x)$ deve ser menor que ϕ).

Implementadas essas adaptações, a complexidade do algoritmo será de $\mathcal{O}(z|A||X|^2 + |X||A|\log(|A|))$, uma vez que, em cada iteração (sendo z iterações) é necessário percorrer todos os estados existentes (estados de origem da transição), para cada estado analisar todos os eventos possíveis ($|A|$), e para cada evento, calcular a probabilidade de transição para todos os outros estados possíveis (estados destino) ($|X|$) resultando na complexidade ($\mathcal{O}(z|A||X|^2)$). Ao fim do algoritmo, é necessário, para cada estado de origem, ordenar os eventos, passando também pelos estados de destino, considerando que a ordenação tem complexidade $\mathcal{O}(|A|\log(|A|))$, esse último passo tem complexidade $\mathcal{O}(|X||A|\log(|A|))$. Representando a complexidade com as estruturas equivalentes do sistema em malha fechada temos $\mathcal{O}(z|\Sigma||Q|^2 + |Q||\Sigma|\log(|\Sigma|))$.

Algoritmo 1: Iteração de Valor Modificado (VIM)

```
   Input: M = (X, A, P, R), φ
   Output: π
 1 atribua V_0[x] arbitrário para cada x ∈ X ;
 2 k ← 0
 3 repeat
 4 |   k ← k + 1 ;
 5 |   foreach x ∈ X do
 6 |   |   V_k[x] ← max_a Σ_x' P(x'|x,a)(R(x,a,x') + γV_{k-1}[x'])
 7 |   end
 8 until |V_k(x) − V_{k-1}(x)| ≤ φ, ∀x ∈ X;
 9 foreach x ∈ X do
10 |   π[x] ← A ordenado por
   |     Σ_x' P(x'|x,a)(R(x,a,x') + γV_k[x']) ;
11 end
```

Essa complexidade considera o pior caso do algoritmo, em que todos estados podem transitar para todos os outros, porém, usualmente, casos reais não chegam ao pior caso. Outro ponto que merece destaque nessa técnica é o fato de que o cálculo da política se dá apenas uma vez, podendo ser gravada para posterior consulta a fim de ser aplicada no sistema.

Exemplo 4. Utilizando como valor de desconto $\gamma = 0,7$ e aplicando o Algoritmo 1 ao supervisor da pequena fábrica apresentado na Figura 2, os resultados da função $V(x)$ são apresentados na Tabela 1.

Tabela 1. Resultados da função $V(x)$ para cada iteração

	V_0	V_1	V_2	V_3	...	V_{109}	V_{110}
IIE	0	1	1,0	1,490	...	2,863	2,863
WIE	0	0	0,7	1,680	...	2,662	2,662
IIF	0	1	2,4	2,889	...	3,803	3,803
IWE	0	2	2,7	2,700	...	4,004	4,004
WWE	0	1	1,0	1,490	...	2,863	2,863
IWF	0	0	0,7	1,680	...	2,662	2,662

Após o cálculo de $V^*(x)$ para todos os estados do supervisor é possível criar a política de eventos. A linha 10 do Algoritmo 1 ordena os melhores eventos de cada estado com base na probabilidade de ocorrência do evento, a recompensa do estado destino e o valor de $V^*(x')$, onde x' é o estado destino. Sendo $I(x,a) = \Sigma_{x'} P(x'|x,a)(R(x,a,x') + \gamma V(x'))$ a função pela qual ordenamos os as ações (a) para cada estado (x), considerando o estado IWE temos:

$$I(IWE, b_2) = 1 * (0 + 0,7 * 2,862) = 1,8781$$
$$I(IWE, a_1) = 1 * (2 + 0,7 * 2,862) = 3,8781$$

Como $I(IWE, a_1) > I(IWE, b_2)$, o evento a_1 tem prioridade sobre b_2. A Tabela 2 apresenta a política completa para a pequena fábrica.

4. ESTUDO DE CASO

Como exemplo de aplicação do algoritmo, será utilizado o Sistema Flexível de Manufatura (SFM) (de Queiroz et al., 2005), cujo diagrama está apresentado na Figura 6 e os autômatos de suas máquinas na Figura 7, onde cada estado está representado como (q, t), sendo q o nome do estado

Tabela 2. Política para o problema da pequena fábrica

Estado	Eventos
IIE	a_1
WIE	b_1
IIF	a_2
IWE	a_1, b_2
WWE	b_1, b_2
IWF	b_2

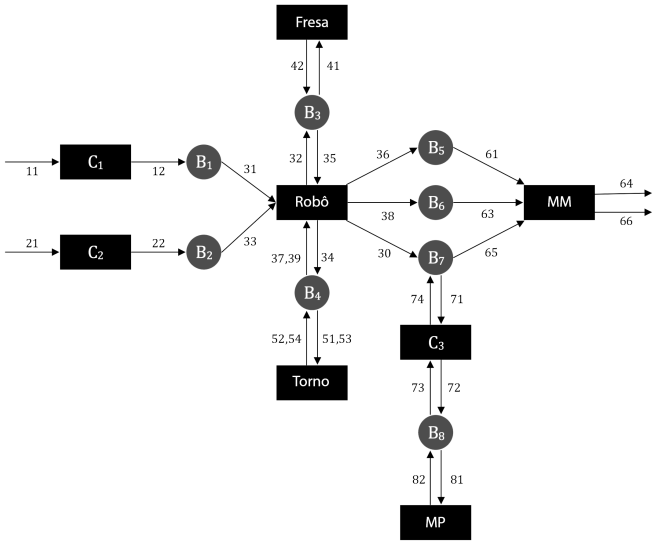

Figura 6. Diagrama do Sistema Flexível de Manufatura

e t o número de tarefas ativas no estado ($t = f_{ta}(q)$). O Sistema Flexível de Manufatura é uma planta composta por oito dispositivos, três esteiras (C_1, C_2 e C_3), uma fresadora, um torno, um robô, uma máquina de pintura (MP) e uma máquina de montagem (MM). O SFM produz dois tipos de produto a partir de blocos e tarugos, sendo eles uma base com pino cônico, chamado Produto A, e uma base com um pino cilíndrico pintado, chamado Produto B.

Na Tabela 3 estão apresentados os intervalos de tempo entre a execução de um evento controlável e a ocorrência de seu respectivo evento não controlável de resposta. Além do especificado na tabela, existe uma especificidade do SFM relativa aos eventos controláveis 63 e 65 que somente podem ocorrer, no mínimo, 15 unidades de tempo após o evento 61.

A produção da base de ambos os produtos (A e B) é realizada pela execução dos seguintes pares de eventos (com os eventos controláveis na ordem apresentada):

$$b = \{(11, 12), (31, 32), (41, 42), (35, 36), (61)\}.$$

Para produzir o pino do produto A, os pares de eventos executados são:

$$p_a = \{(21, 22), (33, 34), (51, 52), (37, 38), (63, 64)\}.$$

Finalmente, para produzir um pino do produto B, é necessário executar os pares:

$$p_b = \{(21, 22), (33, 34), (53, 54), (39, 30), (71, 72), (81, 82), (73, 74), (65, 66)\}.$$

Tabela 3. Intervalo de tempo entre pares de eventos

Eventos Controláveis	Eventos Não Controláveis	Intervalo de Tempo [$u.t.$]
11	12	25
21	22	25
31	32	21
33	34	19
35	36	16
37	38	24
39	30	20
41	42	30
51	52	38
53	54	32
63	64	26
65	66	26
71	72	25
73	74	25
81	82	24

Considerando o supervisor monolítico para o SFM, obtido pela Teoria de Controle Supervisório convencional, composto por 45.504 estados e 200.124 transições, a produção de um produto A e um produto B é considerado um lote de tamanho 1. Desta forma, um lote de tamanho N consiste na produção de N produtos A e N produtos B, tal que o número de eventos necessários para a execução de um lote qualquer é dado por 44 N.

Utilizando a função de tarefas ativas do supervisor monolítico, o algoritmo de iteração de valor modificado (VIM) foi executado, e a política resultante foi obtida, utilizando os tempos da Tabela 3. O *makespan* e paralelismo acumulado das sequências obtidas utilizando a política gerada pelo VIM foram comparados com os resultados dos algoritmos de Maximização de Paralelismo com Restrições Temporais (PMT - *Parallelism Maximization with Time Restrictions*) (Alves et al., 2019) e o algoritmo de Minimização Heurís-

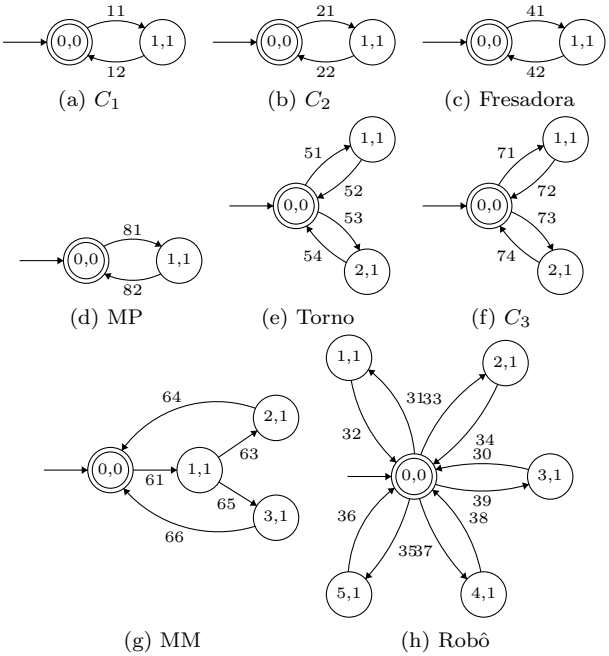

Figura 7. Autômatos das máquinas utilizadas na SFM

tica de Makespan (HMM - *Heuristic Makespan Minimization*) (Alves et al., 2019).

A tabela 4 foi construída a partir de dados de um simulador, cujo cálculo de *makespan* é descrito em (Cassandras and Lafortune, 2008). Podemos observar que o algoritmo VIM tem *makespan* ligeiramente maior que os demais algoritmos, diferindo do algoritmo PMT em 22 u.t. e do HMM em 34 u.t. para um lote de 100 produtos. A Tabela 5 e a Figura 4 apresentam o grande diferencial do algoritmo VIM, que é a sua velocidade de execução para criação de sequência de eventos. Isso se deve ao fato de que, uma vez que a política já esteja pronta, basta que o escalonador de tarefas consulte a política e defina qual o melhor evento para determinado estado. O tempo de execução associado ao VIM é o tempo de execução do escalonador para a criação da sequência. A Tabela 6 apresenta os resultados de paralelismo acumulado entre os algoritmos, sendo que o VIM possui um resultado inferior ao PMT e superior ao HMM.

O VIM foi executado em um computador com CPU *Intel Pentium G2020* 2.90 GHz e 8 GB de memória RAM. O tempo médio necessário para calcular a política foi de aproximadamente 49 segundos e o algoritmo convergiu em 124 iterações, com um desconto $\gamma = 0,7$. Valores menores de γ podem ser sintonizados e permitem uma convergência mais rápida do algoritmo, porém podem reduzir a qualidade do resultado.

Tabela 4. Makespan obtido por cada algoritmo

	Makespan [u.t.]		
Tamanho do Lote	VIM	PMT	HMM
1	**272**	239	238
5	**900**	878	866
10	**1.685**	1.663	1.651
15	**2.470**	2.448	2.436
50	**7.965**	7.943	7.931
100	**15.815**	15.793	15.781
500	**78.615**	78.593	78.581
750	**117.865**	117.843	117.831
1000	**157.115**	157.093	157.081

Tabela 5. Tempo de execução de cada algoritmo.

	Tempo de execução [segundos]		
Tamanho do Lote	VIM	PMT	HMM
1	**0,002**	0,014	0,019
5	**0,006**	0,020	0,122
10	**0,007**	0,082	0,299
15	**0,009**	0,183	0,517
50	**0,032**	1,888	3,836
100	**0,075**	4,753	8,716
500	**0,484**	38,450	46,090
750	**0,721**	58,867	66,733
1000	**0,881**	78,215	89,960

Tabela 6. Paralelismo acumulado obtido por cada algoritmo.

	Paralelismo Acumulado		
Tamanho do Lote	VIM	PMT	HMM
1	**96**	95	93
5	**706**	713	635
10	**1.471**	1.488	1.315
15	**2.236**	2.263	1.995
50	**7.591**	7.688	6.755
100	**15.241**	15.438	13.555
500	**76.441**	77.438	67.955
750	**114.691**	116.188	101.955
1000	**152.941**	154.938	135.955

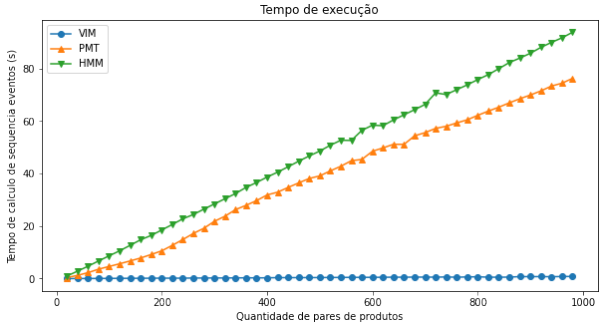

Figura 8. Relação entre o tempo de execução de cada algoritmo e a quantidade de produtos por lote.

5. CONCLUSÃO

Esse artigo apresentou uma nova abordagem para solução de um problema de planejamento da produção baseado na maximização de paralelismo em sistemas a eventos discretos. Nessa abordagem, tratamos o problema como um processo de decisão markoviano propondo uma alteração no algoritmo de iteração de valor. Dessa forma foi possível criar uma política de eventos robusta que permita o cálculo de uma sequência de eventos de forma muito mais rápida que os outros algoritmos utilizados como comparação, independentemente da quantidade de produtos a serem produzidos. Apontamos como possibilidade de trabalho posterior a coleta de dados estatísticos de ocorrência de eventos não controláveis, permitindo assim a adoção de melhores valores probabilísticos, bem como analisar o impacto dos mesmos na elaboração da política pelo algoritmo VIM.

REFERÊNCIAS

Abdeddaïm, Y., Asarin, E., Maler, O., et al. (2006). Scheduling with timed automata. Theoretical Computer Science, 354(2), 272–300.

Alves, L., Bravo, H., Pena, P., and Takahashi, R. (2015a). Escalonamento da produçao baseado no critério de máximo paralelismo em sistemas a eventos discretos. XII Simpósio Brasileiro de Automaçao Inteligente (SBAI), 594–599.

Alves, L.V., Pena, P.N., and Takahashi, R.H. (2019). Planning on discrete event systems using parallelism maximization. arXiv preprint arXiv:1912.12985.

Alves, L., Bravo, H., Pena, P., and Takahashi, R. (2015b). Escalonamento da produção baseado no critério de máximo paralelismo em sistemas a eventos discretos. In XII Simpósio Brasileiro de Automação Inteligente (SBAI), 594–599.

Bassey, K. and Chigbu, P. (2012). On optimal control theory in marine oil spill management: A markovian decision approach. European Journal of Operational Research, 217(2), 470–478.

Cassandras, C. and Lafortune, S. (2008). Introduction to Discrete Event Systems. Springer, 2 edition.

de Queiroz, M.H., Cury, J.E.R., and Wonham, W.M. (2005). Multitasking supervisory control of discrete-event systems. Discrete Event Dynamic Systems, 15(4), 375–395.

Guettiche, M. and Kheddouci, H. (2019). Critical links detection in stochastic networks: application to the transport networks. International Journal of Intelligent Computing and Cybernetics.

Herzig, A., de Menezes, M.V., De Barros, L.N., and Wassermann, R. (2014). On the revision of planning tasks. In ECAI, 435–440.

Kobetski, A. and Fabian, M. (2006). Scheduling of discrete event systems using mixed integer linear programming. In Discrete Event Systems, 2006 8th International Workshop on, 76–81. doi:10.1109/WODES.2006.1678411.

Luo, J., Xing, K., Zhou, M., Li, X., and Wang, X. (2014). Deadlock-free scheduling of automated manufacturing systems using petri nets and hybrid heuristic search. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 45(3), 530–541.

López-Mellado, E., Villanueva-Paredes, N., and Almeyda-Canepa, H. (2005). Modelling of batch production systems using petri nets with dynamic tokens. Mathematics and Computers in Simulation, 67(6), 541 – 558.

Malik, R. and Pena, P.N. (2018). Optimal task scheduling in a flexible manufacturing system using model checking. IFAC-PapersOnLine, 51(7), 230–235.

Mejía, G., Caballero-Villalobos, J.P., and Montoya, C. (2017). Petri nets and deadlock-free scheduling of open shop manufacturing systems. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 48(6), 1017–1028.

Pellegrini, J. and Wainer, J. (2007). Processos de decisão de markov: um tutorial. Revista de Informática Teórica e Aplicada, 14(2), 133–179.

Pinha, D., de Queiroz, M., and Cury, J. (2011). Optimal scheduling of a repair shipyard based on supervisory control theory. In Automation Science and Engineering (CASE), 2011 IEEE Conference on, 39–44. doi:10.1109/CASE.2011.6042515.

Puterman, M.L. (2014). Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons.

Schrijver, A. (1986). Theory of Linear and Integer Programming. John Wiley & Sons, Inc., New York, NY, USA.

Su, R., van Schuppen, J., and Rooda, J. (2012). The synthesis of time optimal supervisors by using heaps-of-pieces. Automatic Control, IEEE Transactions on, 57(1), 105–118. doi:10.1109/TAC.2011.2157391.

Tipaldi, M. and Glielmo, L. (2017). A survey on model-based mission planning and execution for autonomous spacecraft. IEEE Systems Journal, 12(4), 3893–3905.

Wang, X., Xing, K., Feng, Y., and Wu, Y. (2019). Scheduling of flexible manufacturing systems subject to no-wait constraints via petri nets and heuristic search. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 1–12. doi:10.1109/TSMC.2019.2958494.

Wonham, W.M. (2014). Supervisory Control of Discrete-Event Systems. Systems Control Group, Department of Electrical & Computer Engineering, University of Toronto, Toronto, Canada.