Identificação Distribuída de Sistemas de Eventos Discretos com o Objetivo de Detecção de Falhas ⋆

José G. V. de Castro * **Gustavo S. Viana** * **Marcos V. Moreira** *

* *COPPE, Programa de Engenharia Elétrica, Universidade Federal do Rio de Janeiro, 21949-900, Rio de Janeiro, R.J, Brazil. (e-mail: gustavo.viana@poli.ufrj.br; moreira.mv@poli.ufrj.br)*

Abstract: The main drawback of identification of Discrete Event Systems with aim of fault detection is that if the system is composed of several subsystems with concurrent behavior, it may be necessary to record a huge amount of data to obtain the monolithic model with a view to guaranteeing that a large part of the possible fault-free system behavior has been observed. In these cases, we can use a distributed identification technique, where models for the fault-free behavior of the subsystems are obtained, and then these models are run in parallel for fault detection. In this work, we present a method for identifying concurrent subsystems based only on the observation of the input and output controller signals. A virtual plant, simulated using a 3D simulation software and controlled by a programmable logic controller, is used to illustrate the proposed method and to compare the efficiencies of the fault detection methods using the subsystem models and the monolithic model.

Resumo: A principal desvantagem da identificação de Sistemas de Eventos Discretos com o objetivo de detecção de falhas é que se o sistema for composto por vários subsistemas com comportamento concorrente, pode ser necessário registrar uma grande quantidade de dados para obter o modelo monolítico com o objetivo de garantir que uma grande parte do possível comportamento do sistema livre de falhas foi observada. Nesses casos, podemos utilizar uma técnica de identificação distribuída, em que são obtidos modelos para o comportamento livre de falhas dos subsistemas, e então esses modelos são executados em paralelo para detecção de falhas. Neste trabalho, apresentamos um método para identificação de subsistemas concorrentes baseado apenas na observação dos sinais de entrada e saída do controlador. Uma planta virtual, simulada usando um software de simulação 3D e controlada por um controlador lógico programável, é usado para ilustrar o método proposto e comparar as eficiências dos métodos de detecção de falhas usando os modelos de subsistema e o modelo monolítico.

Keywords: Subsystem identification; Fault detection; Discrete-event systems; Finite state automata, Black-box identification.

Palavras-chaves: Identificação de subsistemas; Detecção de falha; Sistemas a eventos discretos; Autômatos de estado finito, Identificação caixa-preta.

1. INTRODUÇÃO

Uma falha é uma perturbação que pode levar um sistema a se comportar de forma diferente do esperado, causando acidentes e danos aos equipamentos. O problema de diagnóstico de falhas de Sistemas de Eventos Discretos (SEDs) foi introduzido em Sampath et al. (1995), em que a noção de diagnosticabilidade de linguagem e um método para calcular um diagnosticador foram apresentados. Desde então, vários trabalhos foram propostos apresentando diferentes estratégias de diagnóstico de falhas e novos métodos de verificação de diagnosticabilidade (Debouk et al., 2000; Moreira et al., 2011; Zaytoon and Lafortune, 2013; Cabral and Moreira, 2019; Viana and Basilio, 2019; Viana et al., 2019). Em todos esses trabalhos, assume-se que um modelo de comportamento completo do sistema é conhecido, isto é, o modelo do sistema antes e depois da ocorrência da falha deve ser previamente conhecido ou construído, o que pode ser uma tarefa difícil, ou mesmo impossível para sistemas grandes e complexos.

A fim de contornar a dificuldade de obter o modelo da planta a partir do conhecimento prévio de seu funcionamento completo, técnicas de identificação têm sido propostas na literatura utilizando o formalismo de autômatos ou redes de Petri (Klein et al., 2005; Roth et al., 2010; Dotoli et al., 2011; Cabasino et al., 2015; Estrada-Vargas et al., 2015; Basile et al., 2016; Zhu et al., 2019; Moreira and Lesage, 2019a,b). Nos métodos de identificação de redes de Petri com o objetivo de detecção de falhas, supõe-se que a estrutura ou dinâmica da rede de Petri seja total ou parcialmente conhecida, o que torna este formalismo adequado para a modelagem desses sistemas. No entanto, quando nenhum conhecimento prévio da estrutura ou dinâmica do sistema é fornecido, então os autômatos tornam-se um formalismo adequado para identificação devido à sua estrutura mais básica (de Souza et al., 2020).

Em Klein et al. (2005), é proposto um modelo de autômato monolítico para detecção de falhas, capaz de representar o comportamento livre de falhas de um SED em malha fechada.

⋆ Este trabalho é financiado pela FAPERJ, CNPq, e Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES - Código de financiamento 001.

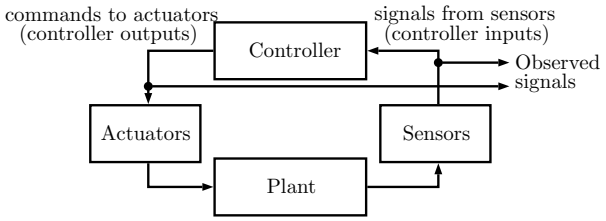


Figura 1. Sistema a eventos discretos em malha fechada.

Este modelo é não determinístico com saídas de estado, e é chamado NDAAO (*Non-Deterministic Autonomous Automaton with Output*). O NDAAO é obtido a partir de sequências observadas de sinais binários trocados entre a planta e o controlador (sinais dos sensores emitidos pela planta e comandos do atuador gerados pelo controlador), conforme mostrado na Figura 1. Em Klein et al. (2005), o modelo de sistema livre de falhas é usado no esquema de detecção de falhas comparando as mudanças de sinal observadas com as mudanças de sinal esperadas do modelo. Se houver uma diferença entre eles, a falha é detectada.

A principal desvantagem da estratégia proposta em Klein et al. (2005) é que se o sistema for composto por vários subsistemas com comportamento concorrente, pode ser necessário registrar uma grande quantidade de dados para obter o modelo monolítico, com o objetivo de garantir que uma grande parte do possível comportamento do sistema livre de falhas foi observada. Somente nesses casos, o número de falsos alarmes é reduzido, e o comportamento observado pode ser considerado suficientemente próximo do original.

Para contornar a necessidade de obtenção de um modelo monolítico, em Roth et al. (2010), é proposto um método de identificação distribuída, em que subsistemas são identificados resolvendo um problema de otimização. A função custo do problema de otimização está relacionada com medidas que podem ser utilizadas para indicar a concorrência entre subsistemas. A principal desvantagem do método proposto em Roth et al. (2010) é que a otimização depende muito da escolha inicial dos valores dos parâmetros de otimização. Em Saives et al. (2018), um método de identificação distribuída para fins de engenharia reversa é proposto, com o objetivo de obter redes de Petri interpretadas para os modelos de subsistemas. Fragmentos observáveis são usados para obter uma primeira divisão do sistema em subsistemas, e então esses fragmentos são usados como entrada de um algoritmo de agrupamento que fornece uma nova divisão do sistema, resolvendo um problema de otimização cujo objetivo é minimizar a complexidade estrutural dos modelos distribuídos identificados.

Neste trabalho, é apresentado um novo procedimento para dividir o sistema em subsistemas. O método é baseado apenas nos sinais observados de entrada e saída do controlador, em que cada subsistema é formado por atuadores que não são observados operando concorrentemente, e seus sensores associados definidos satisfazendo uma relação de causalidade. Após a identificação dos componentes dos subsistemas, o modelo NDAAO de cada subsistema é identificado e todos os modelos são executados em paralelo para detectar a falha do sistema. Se for observado um vetor formado por sinais do controlador de entrada e saída de um subsistema que não é viável no modelo correspondente, então a falha é detectada. Para ilustrar o método de identificação automática dos componentes do subsistema, é utilizada uma planta virtual, controlada por um controlador lógico programável (CLP). Também comparamos as eficiências dos métodos de detecção de falhas usando os modelos de subsistema e o modelo monolítico.

Este trabalho está organizado da seguinte forma. Na Seção 2, são apresentados alguns conceitos preliminares. Na Seção 3, formula-se o problema de identificação usando um modelo monolítico e apresenta-se o modelo NDAAO. Na Seção 4, são apresentados os algoritmos para identificar os componentes do subsistema, e na Seção 5, é mostrado um exemplo prático para ilustrar o método e também comparar a eficiência do esquema de detecção de falhas usando o modelo monolítico com a eficiência de usar os modelos de subsistema. Por fim, na Seção 6, são apresentadas as conclusões.

2. FUNDAMENTOS TEÓRICOS

Seja $G = (X, \Sigma \cup \{\varepsilon\}, f_{nd}, X_0, X_m)$ um autômato de estado finito não determinístico, em que X é o conjunto finito de estados, Σ é o conjunto finito de eventos, ε denota a sequência vazia de eventos, $f_{nd} : X \times \Sigma \cup \{\varepsilon\} \to 2^X$ é a função de transição não determinística, X_0 é o conjunto de estados iniciais e X_m é o conjunto de estados marcados.

Para definir a linguagem gerada por G, é necessário estender o domínio da função f_{nd} para $X \times \Sigma^*$, em que Σ^* denota o fechamento Kleene de Σ, obtendo a função de transição estendida f^e_{nd}. Seja $\varepsilon R(x)$ o alcance-ε de um estado x, isto é, o conjunto de estados alcançados de x seguindo as transições rotuladas com ε, incluindo o estado x (Cassandras and Lafortune, 2008). O alcance-ε pode ser estendido para um conjunto de estados $B \subseteq X$ como $\varepsilon R(B) = \cup_{x \in B} \varepsilon R(x)$. A função de transição não determinística estendida $f^e_{nd} : X \times \Sigma^* \to 2^X$, pode ser definida recursivamente como $f^e_{nd}(x, \varepsilon) = \varepsilon R(x)$, e $f^e_{nd}(x, s\sigma) = \varepsilon R[\{z : z \in f_{nd}(y, \sigma) \text{ para algum estado } y \in f^e_{nd}(x, s)\}]$. Assim, a linguagem gerada por G é definida como $\mathcal{L}(G) = \{s \in \Sigma^* : (\exists x \in X_0)[f^e_{nd}(x, s) \text{ está definido}]\}$.

Seja $\mathcal{G} = (V, E)$ um grafo, em que V é o conjunto de vértices e $E \subseteq V \times V$ é o conjunto de arestas. Se as relações entre todos os pares de vértices são simétricas, isto é, $(v_1, v_2) \in E$ implica que $(v_2, v_1) \in E$, $\forall v_1, v_2 \in V$, então o grafo é dito ser um grafo não direcionado. Neste trabalho, todos os grafos são não direcionados.

O complemento de um grafo $\mathcal{G}$, denotado por $\mathcal{G}^c = (V, E^c)$, é tal que $E^c = \{(v_i, v_j) \in V \times V : (v_i, v_j) \notin E\}$. Um grafo completo é um grafo não direcionado tal que cada par de vértices distintos é conectado por uma aresta (Bondy and Murty, 1976), *i.e.*, o complemento de um grafo completo é um grafo sem arestas.

Um subgrafo induzido $\mathcal{G}'$ de um grafo $\mathcal{G}$ é outro grafo, formado a partir de um subconjunto de vértices de $\mathcal{G}$ e suas arestas associadas, isto é, um subgrafo induzido subgrafo de $\mathcal{G} = (V, E)$ é um grafo $\mathcal{G}' = (V', E')$ tal que $V' \subset V$ e $E' = \{(v_i, v_j) \in E : v_i, v_j \in V'\}$.

Um clique C em um grafo $\mathcal{G}$ é um subconjunto de vértices $V' \subseteq V$, tal que o subgrafo de $\mathcal{G}$ induzido por V' é completo (Cormen and Leiserson, 2009). Com um leve abuso de notação, o termo clique também será usado para o subgrafo induzido por V'. O tamanho de um clique C é definido como o número de vértices de C. Um clique $C = (V', E')$ é maximal se não existir outra clique $\tilde{C} = (\tilde{V}, \tilde{E})$ tal que $V' \subset \tilde{V}$. Em Moon and Moser (1965), mostra-se que a complexidade computacional

para obter todos os cliques maximais de um grafo $\mathcal{G}$ é $O(3^{n/3})$, em que n é o número de vértices de $\mathcal{G}$.

O dual de um clique é chamado de conjunto independente. Um conjunto independente de um grafo não direcionado $\mathcal{G}$ é um subconjunto $V'' \subseteq V$, tal que para cada par de vértices $v_1, v_2 \in V''$, $(v_1, v_2) \notin E$, isto é, um conjunto de vértices V'' é um conjunto independente se, e somente se, V'' for um clique do complemento de $\mathcal{G}$, $\mathcal{G}^c$.

Um conjunto independente maximal V'' é um conjunto independente tal que não existe um conjunto independente diferente que contenha V''. A complexidade de calcular todos os conjuntos independentes maximais é a mesma que calcular todos os cliques maximais de um grafo (Cormen and Leiserson, 2009). É importante observar que, embora o número de cliques maximais cresça exponencialmente com o número de vértices de $\mathcal{G}$, encontrar um clique maximal que contenha um determinado vértice pode ser feito em $O(n)$. Um algoritmo para calcular conjuntos independentes maximais de um grafo é apresentado em Bondy and Murty (1976).

3. IDENTIFICAÇÃO MONOLÍTICA

Considere o sistema de malha fechada representado na Figura 1, e sejam n_I e n_O o número de entradas e saídas binárias do CLP, respectivamente. Então, o vetor formado pelos sinais de entrada e saída do CLP em um instante de tempo $t_i \in \mathbb{R}$ é denotado como $u_i = [I_1(i)\, I_2(i)\, \ldots\, : I_{n_I}(i)\, O_1(i)\, O_2(i)\, \ldots\, O_{n_O}(i)]$, tal que $I_\beta(i)$ e $O_\delta(i)$, para $\beta \in \{1,2,\ldots,n_I\}$ e $\delta \in \{1,2,\ldots,n_O\}$, são, respectivamente, a entrada e sinais de saída do controlador no instante de tempo t_i. O vetor $u_i \in \mathbb{Z}_2^n$, em que $n = n_I + n_O$ e $\mathbb{Z}_2 = \{0,1\}$, é chamado de vetor de I/O.

O objetivo da identificação do sistema é computar um modelo capaz de simular o comportamento observado do sistema, isto é, um modelo que represente as sequências de vetores de I/O que o sistema executa ao realizar suas tarefas. Para obter o comportamento livre de falhas do sistema, os caminhos formados pelos vetores de I/O observados devem ser registrados. Sejam $p_j = (u_{j,1}, u_{j,2}, \ldots, u_{j,l_j})$, $j = 1,\ldots,r$, caminhos associados a tarefas completas executadas pelo sistema. Em Klein et al. (2005), considera-se que todos os caminhos p_j, $j = 1,\ldots,r$, começam com o mesmo vetor I/O, isto é, $u_{j,1} = u_{z,1}$ para todos os $j,z \in \{1,2,\ldots,r\}$. Após observar os caminhos p_j, um modelo identificado pode ser calculado. Em Klein et al. (2005), um autômato com saídas autônomo e não determinístico (NDAAO) é usado para obter um modelo monolítico que simula o comportamento observado do sistema livre de falhas. O NDAAO é definido da seguinte forma.

Definição *1.* Um autômato com saídas autônomo e não determinístico (NDAAO) é uma quíntupla

$$NDAAO = (X, \Omega, f_{nd}, \lambda, x_0),$$

em que X é o conjunto finito de estados, $\Omega = \{\omega_1, \omega_2, ..., \omega_{|\Omega|}\}$ é o conjunto finito de símbolos de saída, $f_{nd} : X \to 2^X$ é a função de transição autônoma não determinística, $\lambda : X \to \Omega$ é a função de saída e x_0 é o estado inicial. □

O NDAAO possui um único estado inicial $x_0 \in X$ que está associado ao vetor de I/O inicial observado em todos os caminhos p_j. É importante notar que o NDAAO não possui um evento definido para rotular as transições, o que justifica o fato do modelo NDAAO apresentado na Definição 1 ser autônomo. Por ser um autômato não determinístico, um estado $x \in X$ pode ter vários estados alcançáveis dados por $f_{nd}(x)$, e qualquer transição entre estados pode ocorrer igualmente. A função de saída λ é responsável pela associação de cada estado $x \in X$ com um símbolo de saída $\omega \in \Omega$, em que Ω é composto por todos os vetores de I/O observados. Para satisfazer a relação de simulação, deve haver uma associação de cada $\omega \in \Omega$ com pelo menos um estado $x \in X$.

Cada estado do modelo NDAAO está associado a uma subsequência distinta de vetores de I/O de um dado comprimento k observados nos caminhos p_j, e é calculado de tal forma que todas as subsequências observadas de comprimento k estão associadas aos estados do modelo NDAAO. O parâmetro livre k leva a um compromisso entre o tamanho do modelo e sua precisão na descrição do comportamento observado (Klein et al., 2005).

Seja a função $Remove : \Omega^k \to \Omega^{k-1}$, que remove o primeiro vetor de uma sequência de vetores I/O de comprimento k, isto é, se $\sigma^k = (u_1, u_2, \ldots, u_k)$, então $Remover(\sigma^k) = (u_2, \ldots, u_k)$. Para calcular o modelo NDAAO é necessário primeiro gerar caminhos modificados, p_j^k, a partir de p_j, de acordo com o parâmetro livre k da seguinte forma:

$$p_j^k = (\sigma_{j,1}^k, \sigma_{j,2}^k, \ldots, \sigma_{j,l_j}^k),$$

em que $\sigma_{j,1}^k$ é formado pela concatenação de k vetores I/O todos iguais a $u_{j,1}$, e $\sigma_{j,i}^k = Remove(\sigma_{j,i-1}^k)u_{j,i}$, para $1 < i \leq l_j$. Após o cálculo dos caminhos modificados p_j^k, o modelo NDAAO pode ser calculado de forma que cada estado esteja associado a uma sequência distinta de vetores de I/O $\sigma_{j,i}^k$, e a transição entre dois estados existem no modelo NDAAO se, e somente se, as sequências associadas de vetores de I/O foram observadas em pelo menos um p_j^k. O valor de $\lambda(x)$, em que $x \in X$, é definido como o último vetor de I/O da sequência de vetores de I/O associados a x.

Para definir a linguagem formada por todas as sequências de vetores de I/O de comprimento até q do modelo NDAAO identificado, primeiro apresentamos as seguintes linguagens geradas a partir de um estado $x_z \in X$:

$$W_{x_z}^1 = \{\omega \in \Omega^1 | \omega = \lambda(x_z)\}$$

e

$$W_{x_z}^{q>1} = \{\omega \in \Omega^q | (\omega = (\lambda(x_z), \lambda(x_{z+1}), ..., \lambda(x_{z+q-1})) \wedge x_{\eta+1} \in f_{nd}(x_\eta), \forall z \leq \eta < z+q-1)\}.$$

Então,

$$L_{Iden}^q = \bigcup_{h=1}^{q} \bigcup_{x \in X} W_x^h.$$

A linguagem formada por todas as sequências observadas de vetores de I/O de comprimento q é dada por:

$$W_{Obs}^q = \bigcup_{j=1}^{r} \left(\bigcup_{i=1}^{l_j-q+1} (u_{j,i}, u_{j,i+1}, \ldots, u_{j,i+q-1}) \right).$$

Assim, a linguagem formada por todas as sequências de vetores I/O de comprimento um até um dado valor q é dada por:

$$L_{Obs}^q = \bigcup_{h=1}^{q} W_{Obs}^h.$$

Em Klein et al. (2005) e Roth et al. (2010) é provado que $L^q_{Obs} \subseteq L^q_{Iden}$, isto é, o modelo identificado simula a linguagem observada do sistema sem falhas.

4. MÉTODO PARA IDENTIFICAR AUTOMATICAMENTE SUBSISTEMAS CONCORRENTES

Nesta seção, apresentamos um método para dividir automaticamente o sistema em subsistemas $\rho \in \mathbb{N}$. A ideia principal é separar o sistema de tal forma que cada modelo de subsistema apresente baixo comportamento concorrente, reduzindo o número de dados observados necessários para identificação.

O primeiro passo para a obtenção dos modelos de subsistemas é identificar e separar os atuadores que operam concorrentemente. Assim, primeiro obtemos um grafo $\mathcal{G}$ cujo conjunto de vértices V é formado pelos atuadores do sistema, isto é, $V = \{O_1, O_2, \ldots, O_{n_O}\}$, e as arestas que os conectam são definidas com base na observação dos vetores de I/O do controlador. Se o sinal do controlador associado a um atuador O_δ for 1, e outro sinal do controlador associado a um atuador diferente O_γ também for 1, no mesmo vetor de I/O, então os atuadores O_δ e O_γ pode operar concorrentemente e uma aresta é criada conectando ambos os vértices do atuador no grafo. Depois disso, conjuntos independentes maximais IS_y são formados a partir de $\mathcal{G}$, de modo que o número de conjuntos independentes maximais é igual ao número de subsistemas ρ e $V = \cup_{y=1}^{\rho} IS_y$. Cada conjunto independente maximal IS_y, para $y = 1, \ldots, \rho$, é formado por todos os atuadores pertencentes ao mesmo modelo de subsistema, pois nenhum deles foi observado atuando ao mesmo tempo que outro atuador de IS_y, isto é, os atuadores de IS_y não funcionam simultaneamente.

No Algoritmo 1 apresentamos o método para calcular os conjuntos IS_y, para $y = 1, \ldots, \rho$. Nas Linhas 1 a 3, V é definido como o conjunto de atuadores, E é inicializado como conjunto vazio e M_o é definido como o conjunto de índices dos atuadores, respectivamente. Então, nas Linhas 4 a 12, as arestas do grafo $\mathcal{G}$ são definidas quando um atuador O_δ é observado trabalhando concorrentemente com outro atuador O_γ nos caminhos p_j. Finalmente, na Linha 13, um conjunto de conjuntos independentes maximais IS_y é formado a partir de $\mathcal{G}$ usando qualquer algoritmo proposto na literatura (Bondy and Murty, 1976).

Após calcular os conjuntos independentes maximais IS_y, os sensores são adicionados a IS_y para obter todos os dispositivos pertencentes ao mesmo subsistema se uma das seguintes condições for verdadeira: (i) a mudança no valor do sinal do sensor, ou o estado ativo do sensor, provoca uma mudança em pelo menos um dos atuadores pertencentes a IS_y, em pelo menos $P \in (0, 100]$ porcento dos casos em que o atuador altera seu valor; (ii) a mudança no valor do sensor sempre ocorre quando um dos atuadores em IS_y é acionado. O valor de P é um parâmetro livre que pode ser escolhido para estabelecer uma relação de causalidade entre sensores e atuadores. para verificar as Condições (i) e (ii), para obter os sensores associados aos subsistemas, todas as sequências observadas de vetores de I/O dos caminhos p_j, $j = 1, \ldots, r$, são usados.

No Algoritmo 2 obtemos os sensores que devem ser adicionados a IS_y, satisfazendo a Condição (i). Para isso, o contador $count_a$ é definido para calcular o número de vezes que um atuador O_δ de IS_y mudou seu valor, e o contador $count_\beta$ é definido para calcular o número de vezes que um sensor I_β mudou seu valor no mesmo ciclo de varredura que O_δ mudou seu valor, ou que I_β estava ativo antes de O_δ alterasse seu valor. Na Linha 15, se $count_\beta \geq P \times count_a$, então estabelecemos uma relação de causalidade entre o sensor I_β e a ativação do atuador O_δ, e I_β é adicionado ao conjunto de componentes do subsistema IS_y na Linha 16.

Algorithm 1: Computação dos conjuntos independentes maximais

Input: Caminhos p_j, para $j = 1, ..., r$
Output: Conjuntos independentes maximais IS_y, $y = 1, \ldots, \rho$

1 $V \leftarrow \{O_1, O_2, \ldots, O_{n_O}\}$
2 $E \leftarrow \emptyset$
3 $M_o \leftarrow \{1, \ldots, n_O\}$
4 **while** $M_o \neq \emptyset$ **do**
5 Defina δ como qualquer elemento de M_o
6 $M_o \leftarrow M_o \setminus \{\delta\}$
7 **for** $j = 1$ **to** r **do**
8 **for** $i = 1$ **to** l_j **do**
9 **for** $\gamma \in M_o$ **do**
10 Defina $O_\delta(i)$ e $O_\gamma(i)$ como o $(n_I + \delta)$-ésimo e $(n_I + \gamma)$-ésimo elementos do vetor $u_{j,i}$
11 **if** $O_\delta(i) = O_\gamma(i) = 1$ **then**
12 $E \leftarrow E \cup \{(O_\delta, O_\gamma)\}$
13 Encontre os conjuntos independentes maximais IS_y do grafo $\mathcal{G} = (V, E)$ tal que $V = \cup_{y=1}^{\rho} IS_y$

Algorithm 2: Relação de causalidade entre sensores e atuadores

Input: Caminhos p_j, para $j = 1, ..., r$, conjuntos IS_y, $y = 1, \ldots, \rho$, e P
Output: IS_y, $y = 1, \ldots, \rho$, atualizados com os sensores cujos valores podem ser associados a alterações nos atuadores

1 **for** $y = 1$ **to** ρ **do**
2 **for** $O_\delta \in IS_y$ **do**
3 $count_a \leftarrow 0$
4 **for** $\beta \in \{1, \ldots, n_I\}$ **do**
5 $count_\beta \leftarrow 0$
6 **for** $j = 1$ **to** r **do**
7 $[I_1(1) \ldots I_{n_I}(1)\, O_1(1) \ldots O_{n_O}(1)] \leftarrow u_{j,1}$
8 **for** $i = 2$ **to** l_j **do**
9 $[I_1(i) \ldots I_{n_I}(i)\, O_1(i) \ldots O_{n_O}(i)] \leftarrow u_{j,i}$
10 **if** $O_\delta(i-1) \neq O_\delta(i)$ **then**
11 $count_a \leftarrow count_a + 1$
12 **if** $\exists \beta \in \{1, 2, \ldots, n_I\}$ *tal que* $I_\beta(i-1) \neq I_\beta(i)$ *ou* $I_\beta(i-1) = 1$ **then**
13 $count_\beta \leftarrow count_\beta + 1$
14 **for** $\beta \in \{1, \ldots, n_I\}$ **do**
15 **if** $count_\beta \geq P \times count_a$ **then**
16 $IS_y \leftarrow IS_y \cup \{I_\beta\}$

No Algoritmo 3, encontramos os sensores que devem ser adicionados a IS_y satisfazendo a Condição (ii). Para isso, é calculado o número de vezes que um sensor I_δ muda de valor, e o número de vezes que um atuador O_β de IS_y esteve ativo ao mesmo tempo que o sensor mudou seu valor. Se os números de vezes forem iguais, então estabelecemos uma relação de causalidade entre o atuador O_β e o sensor I_δ, e I_δ é adicionado a IS_y.

Algorithm 3: Relação de causalidade entre atuadores e sensores

Input: Caminhos p_j, para $j = 1, ..., r$, conjuntos IS_y, $y = 1, \ldots, \rho$

Output: IS_y, $y = 1, \ldots, \rho$, atualizados com os sensores que são alterados quando há um atuador ativado de IS_y.

for $y = 1$ **to** ρ **do**
  for $O_\beta \in IS_y$ **do**
    $count_\beta \leftarrow 0$
  for $\delta \in \{1, 2, \ldots, n_I\}$ **do**
    $count_s \leftarrow 0$
    for $j = 1$ **to** r **do**
      $[I_1(1) \ldots I_{n_I}(1)\, O_1(1) \ldots O_{n_O}(1)] \leftarrow u_{j,1}$
      for $i = 2$ **to** l_j **do**
        $[I_1(i) \ldots I_{n_I}(i)\, O_1(i) \ldots O_{n_O}(i)] \leftarrow u_{j,i}$
        if $I_\delta(i-1) \neq I_\delta(i)$ **then**
          $count_s \leftarrow count_s + 1$
          if $\exists O_\beta \in IS_y$ *tal que* $O_\beta(i-1) = 1$ **then**
            $count_\beta \leftarrow count_\beta + 1$
    for $O_\beta \in IS_y$ **do**
      if $count_s = count_\beta$ **then**
        $IS_y \leftarrow IS_y \cup \{I_\delta\}$

Após obter os componentes do subsistema, os modelos NDAAO dos subsistemas podem ser calculados, conforme descrito em Klein et al. (2005), usando para cada subsistema apenas os elementos dos caminhos p_j dados por IS_y.

5. EXEMPLO PRÁTICO

Considere o sistema de separação de caixas representado na Figura 2, apresentado pela primeira vez em Moreira and Lesage (2019b). O sistema é composto por uma esteira de alimentação (FC, *feeder conveyor*), uma esteira de distribuição (DC, *distribution conveyor*), empurradores 1 e 2 (P1 e P2, *Pusher 1 e Pusher 2*) e sensores k_i, $i = 1, \ldots, 8$. Assim, o sistema completo possui 4 atuadores e 8 sensores, e o vetor I/O u é dado por:

$$u = [k_1\ k_2\ k_3\ k_4\ k_5\ k_6\ k_7\ k_8\ FC\ DC\ P1\ P2]^T.$$

O objetivo do sistema é separar caixas pequenas e altas, empurrando as caixas pequenas no primeiro escorregador e caixas altas no segundo. Apenas uma caixa pode estar na esteira de distribuição por vez. Assim, se houver uma caixa na esteira de distribuição e outra caixa chegar ao sensor k_1, então a esteira de alimentação é parada, e é religada somente após observar a borda de subida dos sensores k_6 ou k_8, indicando que os empurradores $P1$ ou $P2$, respectivamente, foram retraídos e a caixa já foi classificada. O sensor k_2 é usado para indicar se a caixa é alta, e os sensores k_3 e k_4 são usados para indicar que a

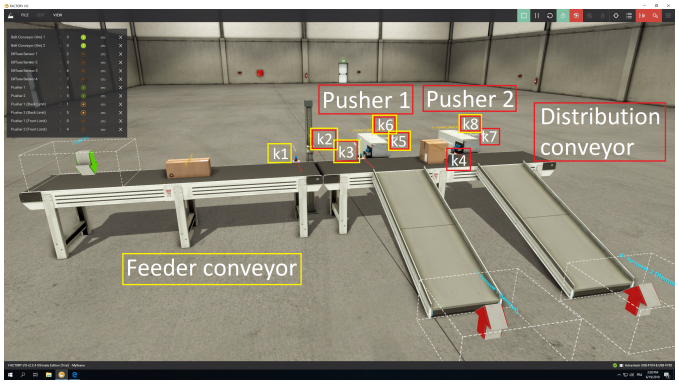


Figura 2. Sistema de separação de caixas.

caixa está na frente dos empurradores $P1$ e $P2$, respectivamente. Após observar a borda de descida do sensor k_3 (resp. k_4), a caixa fica na posição de ser empurrada por P_1 (resp. P_3), e a esteira de distribuição é parada. Os sensores k_5 e k_7 indicam que $P1$ e $P2$ estão completamente estendidos, respectivamente.

O sistema foi simulado usando o software de simulação 3D Factory IO e controlado por um CLP. Para separar o sistema em subsistemas, observamos 2577 vetores de I/O do comportamento livre de falhas do sistema. Usando esses dados, identificamos 13 caminhos cíclicos diferentes p_j. Usando o Algoritmo 1, obtemos o grafo $\mathcal{G}$, representado na Figura 3, a partir do qual identificamos dois conjuntos independentes maximais $IS_1 = \{FC\}$ e $IS_2 = \{DC, P1, P2\}$, isto é, a esteira de alimentação foi observada atuando junto com a esteira de distribuição e os empurradores $P1$ e $P2$, enquanto a esteira de distribuição e os empurradores não foram observados atuando ao mesmo tempo.

Utilizando o Algoritmo 2, definindo o parâmetro livre P como 40%, os sensores que estão associados a cada atuador são apresentados na Tabela 1. Assim, após executar o Algoritmo 2, o conjunto de componentes do primeiro subsistema é atualizado para $IS_1 = \{FC, k_1, k_6, k_8\}$ e o conjunto de componentes do segundo subsistema é atualizado para $IS_2 = \{DC, P1, P2, k_2, k_3, k_4, k_5, k_6, k_7, k_8\}$.

Seguindo os passos do Algoritmo 3, obtemos os atuadores que podem ser associados a cada sensor apresentado na Tabela 2. Assim, os conjuntos de componentes dos subsistemas são atualizados para $IS_1 = \{FC, k_1, k_2, k_3, k_5, k_6, k_8\}$ e $IS_2 = \{DC, P1, P2, k_2, k_3, k_4, k_5, k_6, k_7, k_8\}$. Observe que os sensores k_2, k_3, k_5, k_6 e k_8 são comuns a ambos os subsistemas.

É importante observar que, neste exemplo simulado, os conjuntos dos componentes do subsistema obtidos usando todos os dados coletados são iguais aos conjuntos de componentes do subsistema obtidos considerando apenas os primeiros 1061 vetores de I/O observados, ou seja, aproximadamente 41% do total de dados coletados. Assim, mostra-se que os conjuntos IS_1 e IS_2 podem ser computados usando apenas parte do total de dados coletados.

Após obter os componentes de cada subsistema, pode-se calcular o modelo NDAAO de cada subsistema. Para $k = 1$, o primeiro subsistema tem neste caso 13 estados e 23 transições, e o segundo subsistema tem 15 estados e 19 transições. O modelo monolítico também foi calculado para $k = 1$ e possui 34 estados e 50 transições. Para analisar o número de dados coletados necessários para a obtenção dos modelos, considere

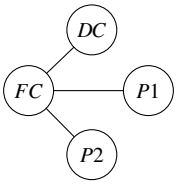


Figura 3. Grafo $\mathcal{G}$.

Tabela 1. Sensores associados com atuadores de acordo com o Algoritmo 2.

Actuator	Sensors
P_1	k_3, k_5, k_6, k_8
P_2	k_4, k_6, k_7, k_8
Feeder Conveyor	k_1, k_6, k_8
Distribution Conveyor	k_3, k_4, k_6, k_8

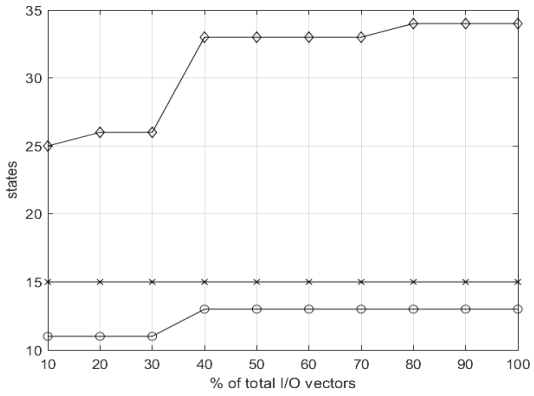
Tabela 2. Atuadores associados com sensores de acordo com o Algoritmo 3.

Sensor	Actuators
k_1	FC
k_2	FC, DC
k_3	FC, DC
k_4	DC
k_5	P_1, FC
k_6	
k_7	P_2
k_8	

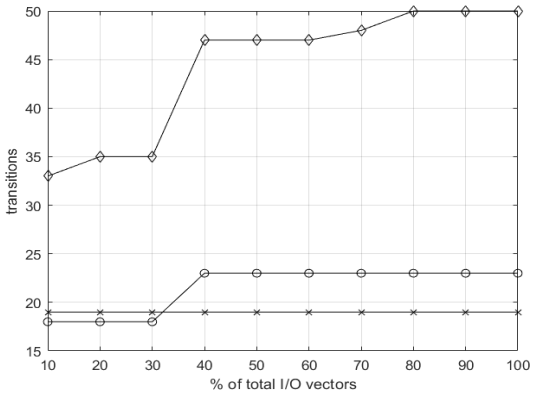
as curvas da Figura 4, que representam o número de estados (Figura 4(a)) e transições (Figura 4(b)) que são criadas nos modelos NDAAO após usar uma porcentagem dos dados coletados completos. Como pode ser visto na Figura 4, os estados e transições do modelo monolítico atingem seus valores máximos usando aproximadamente 80% do total de dados coletados, enquanto ambos os modelos de subsistema atingem seu número máximo de estados e transições com aproximadamente 40% do total de dados coletados. Isso mostra que o tamanho dos dados coletados necessários para obter os subsistemas é muito menor do que o tamanho dos dados necessários para obter o modelo monolítico.

Para comparar a eficiência da detecção de falhas usando os modelos de subsistema com a eficiência da detecção de falhas usando o modelo monolítico, simulamos 24 cenários de falhas, considerando falhas em todos os dispositivos do sistema, mas apenas uma falha por vez. No caso monolítico, verifica-se se o vetor de I/O observado u é igual a uma das saídas dos estados de $f_{nd}(x_c)$, em que x_c é o estado atual do modelo. Se houver um estado $x \in f_{nd}(x_c)$ tal que $\lambda(x) = u$, então o estado do modelo é atualizado para x. Caso contrário, a falha é detectada. A mesma estratégia de detecção de falhas é aplicada para os modelos de subsistema NDAAO. Neste caso, para cada subsistema, são consideradas apenas as entradas e saídas de u pertencentes a IS_y, para $y = 1, \ldots, \rho$. Se pelo menos um dos detectores de falha ρ identificar a ocorrência da falha, a falha será detectada.

Usando o modelo monolítico, 17 das 24 falhas foram detectadas, enquanto que usando os modelos de subsistema, 14 das falhas foram detectadas, isto é, apenas 3 falhas que são detectadas usando o modelo monolítico não podem ser detectadas usando os modelos de subsistema. Isso mostra que os modelos de subsistemas podem ser usados para detecção de falhas sem



(a)



(b)

Figura 4. Número de estados (a) e transições (b) dos modelos NDAAO para $k = 1$: monolítico ($\diamond$); subsistema 1 ($\circ$); e subsistema 2 ($\times$).

reduzir significativamente a eficiência do esquema de detecção de falhas em comparação com o caso monolítico.

6. CONCLUSÕES

Neste trabalho foi apresentado um método para identificar automaticamente os subsistemas concorrentes de um sistema, baseado apenas nas observações dos sinais binários do controlador. O método foi aplicado a uma planta virtual controlada por um CLP, e foi mostrado que os subsistemas podem ser computados usando muito menos observações do que o modelo monolítico. Além disso, mostra-se que a eficiência do esquema de detecção de falhas não é significativamente reduzida em relação à eficiência do esquema de detecção de falhas usando o modelo monolítico. Assim, há uma grande vantagem em utilizar a técnica de identificação distribuída proposta neste trabalho para detecção de falhas de sistemas com comportamento concorrente.

REFERÊNCIAS

Basile, F., Chiacchio, P., and Coppola, J. (2016). Faulty model identification in deterministic labeled time Petri nets. *International Workshop on Discrete Event Systems (WODES)*, 13, 486–492.

Bondy, J.A. and Murty, U.S.R. (1976). *Graph theory with applications*. Elsevier, New York, 1st edition.

Cabasino, M.P., Giua, A., Hadjicostis, C.N., and Seatzu, C. (2015). Fault model identification and synthesis in Petri nets. *Discrete Event Dynamic Systems*, 25(3), 419–440.

Cabral, F.G. and Moreira, M.V. (2019). Synchronous diagnosis of discrete-event systems. *IEEE Transactions on Automation Science and Engineering*, 17(2), 921–932.

Cassandras, C.G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Springer, New York, 2nd edition.

Cormen, T.H. and Leiserson, C.E. (2009). *Introduction to Algorithms*. The MIT Press, Cambridge, 3rd edition.

de Souza, R., Moreira, M., and Lesage, J. (2020). Fault detection of discrete-event systems based on an identified timed model. *Control Engineering Practice*, 105, 104638.

Debouk, R., Lafortune, S., and Teneketzis, D. (2000). Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete Event Dynamic Systems*, 10(1), 33–86.

Dotoli, M., Fanti, M.P., Mangini, A.M., and Ukovich, W. (2011). Identification of the unobservable behaviour of industrial automation systems by Petri nets. *Control Engineering Practice*, 19(9), 958–966.

Estrada-Vargas, A., López-Mellado, E., and Lesage, J. (2015). A black-box identification method for automated discrete-event systems. *IEEE Transactions on Automation Science and Engineering*, 14(3), 1321–1336.

Klein, S., Litz, L., and Lesage, J. (2005). Fault detection of discrete event systems using an identification approach. *IFAC Proceedings*, 38(1), 92–97.

Moon, J.W. and Moser, L. (1965). On cliques in graphs. *Israel Journal of Mathematics*, 3(1), 23–28.

Moreira, M., Jesus, T., and Basilio, J. (2011). Polynomial time verification of decentralized diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 56(7), 1679–1684.

Moreira, M. and Lesage, J. (2019a). Discrete event system identification with the aim of fault detection. *Discrete Event Dynamic Systems*, 29(2), 191–209.

Moreira, M. and Lesage, J. (2019b). Fault diagnosis based on identified discrete-event models. *Control Engineering Practice*, 91, 104101.

Roth, M., Lesage, J., and Litz, L. (2010). Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems. *Proceedings of the 2010 American Control Conference, (pp. 2601-2606), Baltimore, USA, June 30-July 2*.

Saives, J., Farault, G., and Lesage, J. (2018). Automated partitioning of concurrent discrete-event systems for distributed behavioral identification. *IEEE Transactions on Automation Science and Engineering*, 15(2), 832–841.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., and Teneketzis, D. (1995). Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9), 1555–1575.

Viana, G.S. and Basilio, J.C. (2019). Codiagnosability of discrete event systems revisited: A new necessary and sufficient condition and its applications. *Automatica*, 101, 354–364.

Viana, G.S., Moreira, M.V., and Basilio, J.C. (2019). Codiagnosability analysis of discrete-event systems modeled by weighted automata. *IEEE Transactions on Automatic Control*, 64(10), 4361–4368.

Zaytoon, J. and Lafortune, S. (2013). Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2), 308–320.

Zhu, G., Li, Z., Wu, N., and Al-Ahmari, A. (2019). Fault identification of discrete event systems modeled by Petri nets with unobservable transitions. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49, 333–345.