Une Approche par Décomposition de Modèles pour l'Analyse de la Diagnosticabilité des SEDs par Model-Checking

Abderraouf Boussif[1,2] and Mohamed Ghazel[1,2]

[1] Univ. Lille Nord de France, F-59000 Lille, France
[2] IFSTTAR, Cosys/Estas, F-59650 Villenveuve d'Ascq, France
abderraouf.boussif@ifsttar.fr, mohamed.ghazel@ifsttar.fr

Résumé

Cet article s'intéresse à la vérification de la diagnosticabilité des systèmes à évènements discrets (SEDs) dans le contexte de model-checking. Le formalisme de modélisation adopté est les systèmes de transitions labellisés (LTSs). Dans un premier lieu, nous proposons un algorithme permettant la transformation d'un LTS à base d'évènements en un LTS à base d'états, dont le but est d'étendre l'application de l'analyse de la diagnosticabilité par Model-Checking au contexte de diagnostic à base d'évènements. Pour remédier au problème de l'explosion combinatoire de l'espace d'états, nous proposons une technique de décomposition de modèles, qui permet d'extraire uniquement la partie nécessaire pour l'analyse de la diagnosticabilité. L'idée est basée sur la séparation entre les comportements normal et fautif du système. Dans le cas où le système est diagnosticable, nous fournissons une spécification dans la logique temporelle RT-CTL pour analyser la $\mathcal{K}_{min}$-diagnosticabilité en un coup, sans faire appel à un processus incrémental. L'évaluation de nos contributions est faite à travers une étude expérimentale effectuée sur un benchmark relatif à un système de contrôle/commande ferroviaire.

Table des matières

1 Introduction

Bien que récente comparée à la notion de diagnostic, la diagnosticabilité a été largement étudiée par les deux communautés du diagnostic : la communauté DX (pour *Data eXtraction*), issue de l'intelligence artificielle et la communauté FDI (pour *Fault Detection and Isolation*) issue de l'automatique [18].

Dans cet article, nous nous intéressons au problème de la diagnosticabilité des fautes permanentes dans le contexte des Systèmes à Évènements Discrets (SEDs) [3], c.-à-d. l'analyse de la capacité de la fonction de diagnostic à déterminer, *avec certitude*, la présence d'un type de fautes (prédéfini) à partir d'une séquence (finie) d'évènements observés [15]. De nombreux travaux ont déjà été consacrés à ce sujet. La définition originale, dans le contexte du diagnostic à base d'évènements (*Event-Based Diagnosis*), a été introduite dans le travail pionnier de Sampath et al.[15], une approche systématique pour la vérification, basée sur la construction d'un diagnostiqueur a été établie. Des techniques plus efficaces, basées sur la construction de modèles intermédiaires (*twin plant, verifier*) ont été également proposées par la suite [9, 16].

Dans le contexte du diagnostic à base d'états (*State-Based Diagnosis*), i.e. l'occurrence d'une faute est définie par l'atteignabilité d'un état fautif, les auteurs de [17], ont proposé une approche pour l'analyse de la diagnosticabilité basée également sur la construction d'un diagnostiqueur. Le model-checking [6], en tant qu'un ensemble de techniques efficaces pour la vérification formelle, entièrement automatique, a été déployé pour l'analyse de la diagnosticabilité. Dans [10], des spécifications en logique temporelle (LTL) ont été utilisées pour décrire la violation de la diagnosticabilité, un algorithme polynomial à base du (LTL) model-checking pour la vérification de la diagnosticabilité et la synthèse de diagnostiqueur a été fourni. Une définition formelle de la diagnosticabilité dans le contexte du model-checking a été développée dans [5] où la vérification de la diagnosticabilité est réduite à un problème d'atteignabilité dans un modèle intermédiaire (*twin plant*). Dans ce travail la diagnosticabilité est exprimée comme une spécification CTL. Nous avons proposé une extension à ce travail dans [2]. Le but était de se conformer avec la définition orignale de la diagnosticabilité [15].

Un concept pratique de la diagnosticabilité, appelé $\mathcal{K}$-diagnosticabilité, a été initié dans [7], il consiste à quantifier, en terme de nombre d'évènements ou d'états, le délai nécessaire pour rendre un verdict sur la diagnosticabilité du système. Une méthode incrémentale et à la volée qui permet une détermination efficace de ce délai a été proposée dans [12].

Le travail ici présenté s'inscrit dans la continuité des travaux précités relatifs à l'analyse de la diagnosticabilité dans le contexte du model-checking. En effet, il représente une extension de notre précédent travail [2], où des reformulations de la diagnosticabilité et de la $\mathcal{K}$-diagnosticabilité ont été présentées. Notre contribution dans cet article est triple :

1. Nous présentons un algorithme permettant de transformer un modèle de SEDs (modélisé comme système de transitions labellisé) à base d'événements, i.e. l'occurrence d'une faute est modélisée par l'occurrence d'un évènement fautif, vers un modèle à base d'états. Cette transformation permet d'élargir l'applicabilité de l'analyse de la diagnosticabilité avec le model-checking vers les systèmes modélisés à base d'évènements, ce qui permet, par la suite, d'effectuer des comparaisons avec les techniques existantes dans le contexte du diagnostic à base d'évènements.
2. Afin de combattre l'explosion de l'espace d'états du modèle intermédiaire (*twin plant*), nous proposons une approche permettant une réduction de la partie construite et explorée de l'espace d'états. Elle consiste à décomposer le modèle en deux sous-modèles (un sous-modèle contenant les comportements normaux et l'autre ne contient que les exécutions fautives). La composition stricte de ces deux modèles permet d'extraire uniquement la

partie nécessaire pour l'analyse de la diagnosticabilité.

3. Finalement, nous proposons une formulation de la $\mathcal{K}_{min}$-diagnosticabilité dans le contexte du model-checking et nous fournissons une spécification de la logique temporelle RT-CTL (CTL temps réel) qui permet de l'exprimer. A notre connaissance, c'est la première fois où le problème de la $\mathcal{K}_{min}$-diagnosticabilité est étudié dans le contexte de la vérification par model-checking.

Le reste de l'article est organisé de la façon suivante : dans la section 2, nous introduisons le formalisme adopté pour la modélisation des SEDs, ainsi que la définition de la diagnosticabilité et la technique classique pour l'analyser. La section 3 présente la reformulation de la question de diagnosticabilité dans le contexte du model-checking. Nous détaillons, par la suite, les différentes contributions du papier dans la section 4. Une étude expérimentale évaluant les contributions est présentée dans la section 5. Ensuite, nous présentons une discussion par rapport aux travaux connexes dans la section 6, suivie d'une conclusion et des perspectives dans la section 7.

2 Préliminaires

Dans cette section, nous introduisons les systèmes de transitions labellisés (LTSs) comme formalisme adopté pour la modélisation des SEDs [3] ainsi que les différentes notations et opérations utilisées le long de ce papier. Nous rappelons, par la suite, la définition de la diagnosticabilité ainsi que les différentes techniques pour l'analyser.

2.1 Systèmes de transitions labellisés

Nous nous intéressons dans ce papier aux SEDs modélisés par des systèmes de transitions labellisés, tel que défini ci-après.

Définition 1. *Un LTS est un quadruplet $\langle Q, \Sigma, \rightarrow, q_0 \rangle$, avec :*
- *Q : un ensemble fini d'états ;*
- *Σ : un alphabet fini d'évènements (des actions) ;*
- *$\rightarrow \subseteq Q \times \Sigma \times Q$: la relation de transition ;*
- *$q_0 \in Q$: l'état initial.*

2.1.1 Notations

Soit $G = (Q, \Sigma, \rightarrow, q_0)$ un LTS, pour $q, q' \in Q$ et $\sigma \in \Sigma$, on note $s \xrightarrow{\sigma} s' \triangleq (q, \sigma, q') \in \rightarrow$; cette notation est étendue à une séquence d'évènements s de la manière suivante : si $s = \sigma_1, \ldots, \sigma_n$, $q \xrightarrow{s} q' \triangleq \exists\, q_1, \ldots, q_{n-1} : q \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} q_2, \ldots, q_{n-1} \xrightarrow{\sigma_n} q'$.

On pose également $\triangle_Q(q, s) \triangleq \{q' \in Q \mid q \xrightarrow{s} q'\}$ l'ensemble des états que le système peut avoir atteint en tirant la séquence d'évènements s à partir de q.

Le *langage généré* par le système G est $\mathcal{L}(G)$; étant donné une séquence $s \in \mathcal{L}(G)$, l'ensemble des suffixes de s dans $\mathcal{L}(G)$ est noté : $\mathcal{L}(G)/s \triangleq \{t \in \Sigma^* \mid s.t \in \mathcal{L}(G)\}$.

On suppose que le système est partiellement observable, i.e. l'ensemble des évènements Σ est partitionné en deux sous-alphabets disjoints Σ_o et Σ_u où Σ_o est l'ensemble des évènements observables et Σ_u l'ensemble des évènements inobservables ($\Sigma = \Sigma_o \biguplus \Sigma_u$). Pour une étude du diagnostic des fautes, on considère que les fautes sont également des évènements inobservables et on les regroupe dans l'ensemble Σ_f ($\Sigma_f \subseteq \Sigma_u$). L'ensemble des évènements fautifs peut être partitionné en plusieurs sous-ensembles représentant les différents types de fautes. Nous avons

ainsi $\Sigma_f = \Sigma_{f_1} \biguplus \Sigma_{f_2} \biguplus \ldots \biguplus \Sigma_{f_n}$ avec $\Sigma_{f_i}\,(i = 1, \ldots, n)$ est la classe de fautes de types F_i. Pour une séquence d'évènements s, on note $\Sigma_{f_i} \in s$ s'il existe au moins un évènement fautif $f_i \in s$.

On définit la *fonction de projection* sur l'ensemble des évènements observables par $P_{\Sigma_o} : \Sigma^* \to \Sigma_o^*$, i.e. P_{Σ_o} supprime tout simplement tous les évènements inobservables d'une séquence d'évènements donnée. D'une manière générale, $P_{\Sigma_o}(\sigma) = \sigma$ si $\sigma \in \Sigma_o$ et $P_{\Sigma_o}(\sigma) = \epsilon$ si $\sigma \in \Sigma_u$; $P_{\Sigma_o}(s.\sigma) = P_{\Sigma_o}(s)P_{\Sigma_o}(\sigma)$ avec $s \in \Sigma^*$ et $\sigma \in \Sigma$. Par abus de notation, on étend la fonction de projection aux langages par $\forall L \subseteq \Sigma^*$, $P_{\Sigma_o}(L) = \{P_{\Sigma_o}(s) \mid s \in L\}$. Le *langage des traces* de G, défini par $Trace(G) = P_{\Sigma_o}(\mathcal{L}(G))$, représente donc le comportement observable du système G.

Inversement, on définit la *fonction de projection inverse* $P^{-1}_{\Sigma_o}$ telle que $P^{-1}_{\Sigma_o}(y) = \{s \in L \mid P_{\Sigma_o}(s) = y\}$. La notation est également valable pour les langages.

2.1.2 Opérations sur les LTSs

Nous rappelons ici trois opérations essentielles pour notre étude. Le Produit synchrone, la Σ_o-clôture, et la déterminisation.

Définition 2. *(le produit synchrone) Soit $G^i = (Q^i, \Sigma^i, \to_i, q_0^i)$, $i = 1, 2$, deux LTSs. Le produit synchrone entre G^1 et G^2 est le LTS $G^1 \times G^2 = (Q^1 \times Q^2, \Sigma^1 \cap \Sigma^2, \to_{G_1 \times G_2}, (q_0^1, q_0^2))$, où $(q^1, q^2) \xrightarrow{\sigma}_{G1 \times G_2} (q'^1, q'^2)$ existe si et seulement si $q^1 \xrightarrow{\sigma} q'^1$ et $q^2 \xrightarrow{\sigma} q'^2$ existent aussi.*

Le langage du produit satisfait $\mathcal{L}(G^1 \times G^2) = \mathcal{L}(G^1) \cap \mathcal{L}(G^2)$. Il est évident que le LTS produit $(G^1 \times G^2)$ n'est pas forcement vivant, il faut donc utiliser les opérateurs d'accessibilité et de coaccessibilité [3] pour générer la partie vivante du LTS. Contrairement à l'opération de produit synchrone, qui ne tolère que l'évolution sur les évènements communs, la composition parallèle, quant à elle, permet une évolution libre des LTSs sur les évènements non partagés, elle est noté $(G^1 \| G^2)$.

Nous définissons maintenant une opération essentielle pour la vérification de la diagnosticabilité qui est la Σ_o-*clôture*. Elle consiste à s'abstraire des évènements inobservables tout en respectant la sémantique de l'observation (les traces).

Définition 3. *(Σ_o-clôture) pour un LTS $G = (Q, \Sigma, \to, q_0)$, la Σ_o-clôture de G, notée $OBS_{\Sigma_o}(G)$ est un LTS $(Q', \Sigma_o, \to_o, q_0')$ où pour chaque $q, q' \in Q' \subseteq Q$, $\sigma \in \Sigma_o$, $q \xrightarrow{\sigma}_o q'$ s'il existe $s \in \Sigma_u^*$ t.q. $q \xrightarrow{s.\sigma} q'$.*

Notons que la Σ_o-clôture préserve les traces, i.e. $\mathcal{L}(OBS_{\Sigma_o}(G)) = Trace(G)$. Le LTS $OBS_{\Sigma_o}(G)$ est non-déterministe.

La dernière opération, que nous présentons ici, est la déterminisation par rapport à l'ensemble des évènements observables. Elle consiste à générer un LTS déterministe et sans évènements inobservables.

Définition 4. *Soit un LTS $G = (Q, \Sigma, \to, q_0)$, Le LTS déterministe de G relativement à l'ensemble des évènements observables Σ_o est un LTS noté $\mathcal{D}_{\Sigma_o}(G) = (\mathcal{X}, \Sigma_o, \to_d, \mathcal{X}_0)$, où $\mathcal{X} = 2^Q$ un ensemble d'états de G, $\mathcal{X}_0 = q_0$, et $\to_d = \{(X, \sigma, \triangle_Q(X; \Sigma_u^*.\sigma) | X \in \mathcal{X}, \sigma \in \Sigma_o\}$.*

Pour cette définition, le macro-état X' cible d'une transition $X \xrightarrow{\sigma}_d X'$ est composé d'un ensemble d'états q_i' de G qui sont accessibles à travers des séquences de transitions $q \xrightarrow{s_i.\sigma} q_i'$ qui se terminent par l'évènement observable σ (avec $s_i \in \Sigma_u^*$). On note aussi que la procédure de déterministaion préserve les observations, i.e. $\mathcal{L}(\mathcal{D}_{\Sigma_o}(G)) = Trace(G)$.

2.2 L'analyse de la diagnosticabilité

Dans le présent article, nous considérons un seul mode de défaillance, ce qui revient à se donner une seule classe de faute, notée Σ_f. La généralisation à n modes de défaillances est directe. Nous supposons donné également un LTS G satisfaisant les deux hypothèses suivantes

(H_1) *Le langage de G est vivant, i.e. il existe une transition à partir de tout état q de Q ;*

(H_2) *Le modèle ne contient pas de cycle formé exclusivement d'évènements inobservables.*

2.2.1 Définition de la diagnosticabilité

La diagnosticabilité est une propriété importante pour la tâche de surveillance. On dit qu'un SED est diagnosticable pour l'ensemble des partitions de fautes et par rapport à l'ensemble des évènements observables s'il est possible de détecter toute occurrence de fautes dans un délai fini. La définition formelle de la diagnosticabilité à base de langage a été initialement introduite dans [15] comme suit :

Définition 5. *Un langage L préfixe clos et vivant est dit diagnosticable par rapport à une fonction de projection P_{Σ_o} et un ensemble de fautes Σ_f si et seulement si :*

$$(\exists n_i \in \mathbb{N}\, [\forall s \in \Psi(\Sigma_f)]\, (\forall t \in L/s)\, [|t| \geq n_i \Rightarrow D]$$

telle que la condition de diagnosticabilité D est : $\omega \in P_{\Sigma_o}^{-1}[P_{\Sigma_o}(s.t)] \Rightarrow \Sigma_f \in \omega$.

où $\Psi(\Sigma_f)$ est l'ensemble des séquences finies d'évènements qui se terminent par un évènement fautif (de Σ_f). Cette définition signifie qu'un langage L est diagnosticable *si et seulement si*, pour toute séquence s se terminant par un évènement fautif et toute continuation t de s suffisamment longue ($|t| \geq n_i$), alors toute autre séquence d'évènements ω ayant la même projection observable que $s.t$ ($P_{\Sigma_o}(s.t) = P_{\Sigma_o}(\omega)$), contient nécessairement un évènement fautif de Σ_f.

L'approche classique pour l'analyse de la diagnosticabilité, est basée sur la construction d'un automate déterministe, augmenté par des étiquettes, appelé *diagnostiqueur*[15].

Définition 6. *(le diagnostiqueur) soit $G = (Q, \Sigma, \rightarrow, q_0)$ un LTS à diagnostiquer. Le diagnostiqueur correspondant à G est le LTS déterminisé $G_d = \mathcal{D}_{\Sigma_o}(G) = (\mathcal{X}_d, \Sigma_o, \rightarrow_d, x_0)$, augmenté d'une fonction de décision : $\mathcal{D}iag : \mathcal{X}_d \rightarrow 2^{\triangle}$, avec $\triangle = \{F, N\}$, (pour une seule classe de fautes) sachant que F et N indiquent respectivement si un état "Fautif" ou "Normal".*

Chaque macro-état du diagnostiqueur est de la forme $x = \{(q_1, \ell_1), \ldots, (q_n, \ell_n)\}$, avec $q_i \in Q$ et $\ell \in \{F, N\}$. Si $\forall i = 1, \ldots, n$, $\ell_i = N$ (resp. $\ell_i = F$), le macro-état x est dit N-certain (resp. F-certain), sinon il est F-incertain.

La condition nécessaire et suffisante pour la diagnosticabilité d'un LTS G avec cette approche dite "approche diagnostiqueur" a été établie dans [15] et correspond à l'absence de **cycle indéterminé**, défini comme un cycle dans le diagnostiqueur, constitué seulement de macro-états F-incertain et qui correspond à deux cycles dans le LTS G : le 1^{er} cycle ne contient aucun évènement fautif et n'est pas accessible à partir d'une séquence contenant un évènement fautif, alors que le $2^{ème}$ l'est.

2.2.2 La $\mathcal{K}$-diagnosticabilité

Comme nous l'avons mentionné (Définition 5), la diagnosticabilité d'un système implique la diagnosticabilité de chaque faute dans un *délai fini* (en terme de nombre d'évènements).

Cette définition exige l'existence d'une borne supérieure pour ce délai sans spécifier cette borne en terme de nombre fixe d'évènements. Un concept pratique de la diagnosticabilité, appelé la $\mathcal{K}$-diagnosticabilité, considère une quantification de ce délai de diagnosticabilité (un entier $\mathcal{K}$). Ci-après, nous rappelons la définition de la $\mathcal{K}$-diagnosticabilité introduite dans [7].

Définition 7. *Un LTS G est $\mathcal{K}$-diagnosticable, pour un $\mathcal{K} \in \mathbb{N}$, par rapport à une fonction de projection P_{Σ_o} et à un ensemble de fautes Σ_f, si et seulement s'il n'existe pas deux séquences d'évènements s_1 et s_2, qui satisfont les conditions suivantes :*

- $P_{\Sigma_o}(s_1) = P_{\Sigma_o}(s_2)$;
- $\forall f \in \Sigma_f,\ f \notin s_1$;
- $\exists$ *(au moins un évènement)* $f \in \Sigma_f$ *t.q.* $f \in s_2$;
- s_2 *contient, au moins,* $\mathcal{K}+1$ *évènements après l'occurrence de l'évènement fautif* $f \in \Sigma_f$;

Cela signifie que pour toute paire de séquences d'évènements partageant la même observation, avec une séquence qui contient un évènement fautif et pas l'autre, le système est considéré comme $\mathcal{K}$-diagnosticable si et seulement si les deux séquences n'ont pas plus de $\mathcal{K}$ évènements partagés, i.e. ayant la même observation, après l'occurrence de l'évènement fautif.

Généralement, l'analyse de la $\mathcal{K}$-diagnosticabilité fait référence à l'analyse de deux questions :

Question 1. Étant donné un système à diagnostiquer et un entier $\mathcal{K}$, vérifier si le système est $\mathcal{K}$-diagnosticable.

Question 2. Étant donné un système diagnosticable, trouver la valeur minimale $\mathcal{K}$ qui garantit la diagnosticabilité du système (le problème de la $\mathcal{K}_{min}$-diagnosticabilité).

3 Vérification de la diagnosticabilité par model-checking

Le Model-Checking [6] regroupe un ensemble de techniques de vérification formelle entièrement automatique. Il est, à nos jours, largement utilisé durant la spécification, la conception, la vérification et la validation des systèmes dynamiques. Il permet de vérifier si un modèle de système (sous forme d'un système de transition - une structure de Kripke) satisfait une propriété, exprimée en une logique temporelle (LTL, CTL, etc.) en utilisant un ensemble d'algorithmes efficaces basés sur une exploration exhaustive de toutes les exécutions possibles du modèle. Dans le cas où la propriété est violée par une exécution du modèle, un contre-exemple retraçant cette exécution est généré, ce qui permet, par la suite, de guider le débogage.

Dans cette section, nous allons voir la reformulation de la diagnosticabilité comme un problème de model-checking.

3.1 La reformulation de la diagnosticabilité

La diagnosticabilité est à l'origine, une propriété comme toute autre propriété formelle que l'on souhaite vérifier sur un modèle du système. En général, la vérification de la diagnosticabilité avec le Model-Checking se fait à partir d'un modèle de départ à base d'états, i.e. c'est un diagnostic à base d'états (*state-based diagnosis*) où l'ensemble d'états Q du LTS est partitionné en deux ensembles disjoints, un ensemble d'états normaux, noté Q_N, et un ensemble d'états fautifs, noté Q_F (avec $Q{=}Q_N \uplus Q_F$). L'occurrence d'une faute, ou l'exécution d'un comportement fautif, se traduit par l'atteignabilité d'un état ou un ensemble d'états fautifs. Dans cette section, on désigne par G_s un LTS à base d'états. L'analyse de la diagnosticabilité dans le contexte de model-checking est basée sur la notion de *condition de diagnostic* [5], définie comme suit :

Définition 8. *([5]) Une Condition de diagnostic est une paire d'ensembles d'états disjoints et non vides c_1,c_2 de l'ensemble d'états d'un système ($c_1 \cap c_2 = \emptyset$), notée $c_1 \perp c_2$.*

Cette définition permet d'exprimer la détection d'une faute en considérant la condition de diagnostic $Q_N \perp Q_F$ ou la distinction (l'identification) entre deux fautes F_a et F_b avec la condition de diagnostic $Q_{F_a} \perp Q_{F_b}$.

Dans le but de se conformer avec la définition originale de la diagnosticabilité [15], nous avons introduit dans [2] la notion des *paires critiques infinies* comme une extension à la définition des *paires critiques* introduite dans [5].

Définition 9. *Une paire critique infinie dans un LTS G_s, par rapport à une condition de diagnostic $c_1 \perp c_2$, est une paire de séquences d'évènements infinies s_1 et s_2 ayant une observation identique ($P_{\Sigma_o}(s_1) = P_{\Sigma_o}(s_2)$), telle que :* $\exists\, t \in \mathbb{N} \mid \forall\, i \geq t\, :\, c_1(q^i_{s_1}) \perp c_2(q^i_{s_2})$

avec $c_i(q)$ indique que l'état q appartient à l'ensemble d'états c_i, et $q^i_{s_j}$ désigne l'état q du système atteignable, à partir de l'état initial, par l'occurrence de l'$i^{ème}$ évènement dans la séquence d'évènements s_j.

Une paire critique infinie désigne deux séquences d'évènements infinies s_1 et s_2 partageant la même séquence d'évènements observables qui, après l'occurrence d'un nombre (fini) d'évènements, atteignent une condition de diagnostic, i.e. s_1 atteint un état dans c_1 et s_2 atteint un état dans c_2 et elles demeurent dans cette condition ($c_1 \perp c_2$) indéfiniment. A partir de cette définition, on peut déduire que la condition nécessaire et suffisante pour la diagnosticabilité d'un système, dans le sens de [15], est l'absence de paires critiques infinies.

Définition 10. *Une condition de diagnostic $c_1 \perp c_2$, dans un LTS G_s, est diagnosticable ssi G_s ne contient aucune paire critique infinie par rapport à $c_1 \perp c_2$.*

Cette nouvelle définition est équivalente à la définition originale de diagnosticabilité et il est montré qu'un cycle indéterminé peut être représenté par une paire critique infinie [2].

La propriété de $\mathcal{K}$-diagnosticabilité peut être également reformulée comme suit :

Définition 11. *Un LTS G_s est $\mathcal{K}$-diagnosticable, pour un entier $\mathcal{K} \in \mathbb{N}$, par rapport à une condition de diagnostic $c_1 \perp c_2$ et une fonction de projection P_{Σ_o} si et seulement s'il n'existe pas une paire de séquences s_1, s_2 partageant la même observation ($P_{\Sigma_o}(s_1) = P_{\Sigma_o}(s_2)$) telle que :*

$$\exists\, t \in \mathbb{N} \mid c_1(q^{t+i}_{s_1}) \perp c_2(q^{t+i}_{s_2}),\ pour\ 0 \leq i \leq \mathcal{K}$$

En d'autres termes, on dit qu'un LTS est $\mathcal{K}$-diagnosticable, s'il n'existe pas de paires de séquences d'évènements ayant la même projection observable qui mènent le système dans une situation d'ambiguïté, i.e. incertaine, tout en y restant pour plus de $\mathcal{K}$ états successifs .

A l'aide de cette reformulation, l'analyse de la diagnosticabilité peut être réduite à un problème de vérification d'une propriété de sûreté (absence des paires critiques infinies) sur un modèle intermédiaire (un twin plant [9]) avec les techniques de Model-Checking.

3.2 La diagnosticabilité comme un problème de Model-Checking

Pour la vérification de la diagnosticabilité par model-checking, deux tâches doivent être établies :

1. Reformuler le modèle intermédiaire (twin plant) comme une structure de Kripke au regard de la propriété de diagnosticabilité.
2. Exprimer la propriété de diagnosticabilité avec une logique temporelle (LTL, CTL, etc.)

Pour traduire un twin plant en une structure de Kripke, il suffit d'incorporer les états (des deux copies du modèle) ainsi que les évènements observables du twin plant dans l'espace d'états de la structure. Formellement, chaque état de la structure de Kripke est défini par un vecteur de variables (v_1, v_2, v_3) où v_1, v_2 et v_3 représentent respectivement l'état de la $1^{ère}$ copie du modèle, l'état de la $2^{ème}$ copie du modèle, et l'évènement observable qui correspond à leur transition. Finalement, la fonction d'étiquetage associe à chaque état de la structure une proposition parmi $\{N, F\} \times \{N, F\}$, avec N et F indiquent respectivement que l'état est normal ou fautif. Plus de détails sur cette reformulation peuvent être trouvés dans [5].

On définit informellement une propriété de sûreté comme celle stipulant que "*quelque chose de mauvais n'arrive jamais*". Il est à noter que pour tel type de propriété si qu'une fois violée par un modèle de système, un contre-exemple est généré par le model-checker, ce qui permet de retracer l'exécution qui viole la propriété. La diagnosticabilité, quant à elle, peut être vue comme une propriété de sûreté, en considérant que l'atteignabilité d'une paire critique infinie est la situation indésirable. Cette situation indésirable peut être exprimée avec la formule CTL suivante ϕ : EG $(c_1(q) \wedge (c_2(q'))$. dit autrement : " *il existe une exécution dans le twin plant qui mène globalement à une condition de diagnostic*".

La diagnosticabilité peut être, à son tour, exprimée par la formule CTL suivante : $\neg$ EF EG $(c_1(q) \wedge (c_2(q'))$. En d'autres termes : "*il n'existe pas d'exécution pour laquelle, dans le futur, la situation indésirable ϕ arrive*".

Il est maintenant possible de formuler le problème du Model-Checking lié à la vérification de la diagnosticabilité comme suit :

$$K_{(P\times P)} \models \neg\, EF\, EG\, (c_1(q) \wedge (c_2(q'))$$

avec $K_{(P\times P)}$ est la structure de Kripke qui représente le twin plant du modèle de système P. Si la propriété est satisfaite, alors le système est diagnosticable, sinon un contre-exemple est fourni.

4 Contribution à l'analyse de la diagnosticabilité avec Model-checking

Dans cette section, nous introduisons d'abord l'algorithme qui permet la transformation d'un LTS à base d'évènements vers un LTS à base d'états. Nous détaillons par la suite l'approche de décomposition de modèles pour la réduction de l'espace d'états du twin plant généré et qui servira à l'analyse de la diagnosticabilité. Finalement, nous reformulons la question de la $\mathcal{K}_{min}$-diagnosticabilité comme un problème de model-checking.

4.1 La transformation d'un LTS à base d'évènements vers un LTS à base d'états

Dans les travaux de référence, traitant l'analyse de la diagnosticabilité, le modèle à diagnostiquer est un modèle à base d'évènements. Dans le but d'étendre l'analyse de la diagnosticabilité, dans le contexte de model-checking, vers ce type de modèles nous proposons un algorithme (voir algorithme 1) qui permet de transformer un LTS à base d'évènements en un LTS à base d'états.

L'algorithme est un DFS (Depth First Search) qui prend en entrée un LTS à base d'évènements G et génère en sortie un LTS à base d'états G_s. L'idée de base est de dupliquer chaque état q_i atteignable par au moins une séquence d'évènements fautive et une autre

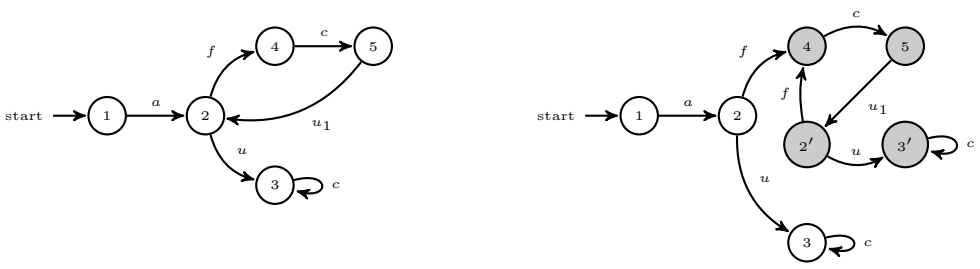
normale, en deux états q_i et q_i_F, avec q_i qui est atteignable par la séquence normale et q_i_F qui est atteignable par la séquence fautive. Le LTS à base d'états généré contient au maximum le double du nombre d'états du LTS à base d'évènements, i.e. le cas où tous les états seront dupliqués.

Pour utiliser l'algorithme, nous associons à chaque état trois étiquettes : q.IsVisited (si l'état est déjà visité ou non), q.IsDuplicated (si l'état est déjà dupliqué ou non) et q.Tag (si l'état est fautif (F) ou non (N)). Nous définissons aussi les deux ensembles **Succ**(q) et **Pred**(q) comme suit :

1. **Succ**(q) $= \{(\sigma_i, q_i) \in \Sigma \times Q \mid q \xrightarrow{\sigma_i} q_i\}$.
2. **Pred**(q) $= \{(q_i, \sigma_i) \in q \times \Sigma \mid q_i \xrightarrow{\sigma_i} q\}$.

Une fois que l'algorithme est exécuté, le LTS généré sera $G_s = \langle Q, \Sigma, \rightarrow, q_0 \rangle$ tel que Q=$Q_N \cup Q_F$ avec $Q_N = \{q \mid q.\text{Tag} = \text{N}\}$ et $Q_F = \{q \mid q.\text{Tag} = \text{F}\}$.

Exemple 1. *Un exemple de cette transformation est illustré dans la figure 1. A droite (a), un LTS à base d'évènements G, avec $\Sigma_o = \{a, c\}$, $\Sigma_u = \{u, u_1, f\}$ avec $f \in \Sigma_f$. A gauche (b), le LTS G_s généré par l'algorithme 1, avec $Q_N = \{1, 2, 3\}$ et $Q_F = \{4, 5, 2', 3'\}$. Les états $2', 3'$ sont, respectivement, les dupliqués des états $2, 3$. Les états $1, 4, 5$ nous sont pas dupliqués parce qu'ils sont atteignables uniquement pas un seul type de séquence (normal ou fautif).*



(a) Le LTS à base d'évènements G

(b) Le LTS à base d'états G_s

FIGURE 1 – Transformation d'un LTS à base d'évènements à un LTS à base d'états

Remarque : Une caractéristique importante du LTS à base d'états généré, réside dans le fait que, une fois que les états sont partitionnés en deux ensembles, normal et fautif, les séquences d'évènements seront également partitionnées en deux types, des séquences d'évènements normales et des séquences d'évènements fautives. L'approche dite de décomposition, que nous introduisons, ci-après, pour la réduction de l'espace d'états, est basée essentiellement sur ce partitionnement des séquences d'évènements.

4.2 La technique de décomposition du modèle

L'analyse de la diagnosticabilité par les techniques *twin plant* [9] et *verifier* [16] revient à la recherche des cycles composés uniquement d'états incertains (F-incertain) dans des modèles intermédiaires [9, 16]. Nous rappelons qu'un modèle intermédiaire *twinplant* est construit par la composition synchrone 2 entre deux copies de la Σ_o-*clôture* 3 du modèle de système (voir

Algorithm 1 Transformer un LTS à base d'évènements à un LTS à base d'états.

Entrée : G : Un LTS à base d'évènements ; q_0 : L'état initial.
Sortie : G_s : Un LTS à base d'états
procedure :

1: $q = q_0$
2: **DFS**(G, q) ▷ La fonction récursive Depth First Search
3: q.IsVisited = True ;
4: **pour tout** $(\sigma_i\ ,\ q_i) \in$ **Succ**(q) **faire**
5: **si** $((q.\text{Tag} = \text{F}\ )\ ||\ (\sigma_i \in \Sigma_f))$ **alors** ▷ La condition pour que l'état q_i soit fautif
6: **si** q_i.IsDuplicated = False **alors** ▷ Vérifier si l'état q_i n'est pas encore dupliqué
7: **Définir** q_i_F ; ▷ Créer un nouvel état (l'état dupliqué de q_i)
8: q_i_F.Tag = F ;
9: **Définir** $(q \xrightarrow{\sigma_i} q_i\text{F})$; ▷ Créer une nouvelle transition vers q_i_F
10: **Supprimer** $(q \xrightarrow{\sigma_i} q_i)$; ▷ Supprimer l'ancienne transitions vers q_i
11: q_i.IsDuplicated = True ;
12: q_i_F.IsDuplicated = True ;
13: **pour tout** $(\sigma_j\ ,\ q_j) \in$ **Succ**(q_i) **faire**
14: **si** $(q_i = q_j)$ **alors** ▷ Vérifier si l'état q_i contient un self-loop
15: **Définir** $(q_i\text{F} \xrightarrow{\sigma_j} q_i\text{F})$;
16: **sinon**
17: **Définir** $(q_i\text{F} \xrightarrow{\sigma_j} q_j)$;
18: **fin si**
19: **fin pour**
20: **si** **Pred**$(q_i) = \{q_i\} \vee \{\}$ **alors** ▷ Le cas où l'état est devenu non-accessible
21: **Supprimer** (q_i) ;
22: **fin si**
23: **DFS**$(G, q_i\text{F})$;
24: **sinon** ▷ Le cas où l'état q_i est déjà dupliqué
25: **Définir** $(q\text{F} \xrightarrow{\sigma_i} q_i\text{F})$;
26: **Supprimer** $(q\text{F} \xrightarrow{\sigma_i} q_i)$;
27: **si** **Pred**$(q_i) = \{q_i\} \vee \{\}$ **alors**
28: **Supprimer** (q_i) ;
29: **fin si**
30: **fin si**
31: **sinon**
32: **si** q_i.IsVisited = False **alors** ▷ Vérifier si l'état q_i n'est pas encore traité
33: **DFS**(G, q_i) ;
34: **fin si**
35: **fin si**
36: **fin pour**

[9] pour plus de détails). Étant donné que le LTS à diagnostiquer contient des exécutions normales et fautives, alors les modèles intermédiaires (les vérificateurs) contiennent trois types d'exécutions :

1. Les exécutions certainement normales : c'est l'ensemble des traces menant ***uniquement*** vers des états de type N-certain.

2. Les exécutions certainement fautives : c'est l'ensemble des traces qui atteignent, après un nombre fini d'évènements, des états de type F-certain.
3. Les exécutions ambiguës (incertaines) : c'est l'ensemble des traces qui contiennent des cycles incertains.

Du point de vue pratique, on peut constater que ce dernier type d'exécution, dans le twin plant, représente la partie nécessaire pour l'analyse de la diagnosticabilité, L'idéal est de pouvoir construire un modèle intermédiaire générant uniquement ce type d'exécution, ce qui permet de réduire partiellement l'espace d'états généré ainsi que le temps d'analyse. Nous proposons dans cette section une technique pour générer ce modèle intermédiaire contenant uniquement le comportement ambigu. Cette technique est basée sur la décomposition du modèle (le LTS G_s) en deux LTSs : un LTS noté G_N, contient uniquement les exécutions normales du modèle G_s et un LTS noté G_F, contient uniquement les exécutions fautives de G_s. Le fait d'avoir généré le LTS à base d'états nous aide énormément dans la génération des deux modèles G_N et G_F, dans la mesure où les séquences d'évènements ainsi que les états sont déjà partitionnées en normales et fautives dès la transformation. La génération des deux modèles G_N et G_F se fait de la manière suivante :

1. **Génération du LTS** G_N : il suffit de supprimer tous les états fautifs ainsi que les transitions correspondantes dans le LTS à base d'états, i.e. le LTS G_N représente la partie *accessible* à partir de l'état initial de G_s, uniquement vers les états normaux.
2. **Génération du LTS** G_F : il suffit de garder uniquement la partie *accessible*, à partir de l'état initial de G_s, uniquement vers les états fautifs.

Nous utilisons le LTS de l'exemple précédent (Exemple 1), pour illustrer cette technique de décomposition.

Exemple 2. *La figure 2(a) représente les deux LTSs générés après la décomposition du LTS G_s (voir Figure 1(b), G_N (en haut) représente le comportement normal et G_F (en bas) représente le comportement fautif. Le produit synchrone de leurs observateurs (i.e. l'approche twin plant, en prenant la propriété de symétrie en compte [8]) est illustré par la figure 2(b). Il est clair que ce produit comporte uniquement les exécutions ambiguës.*

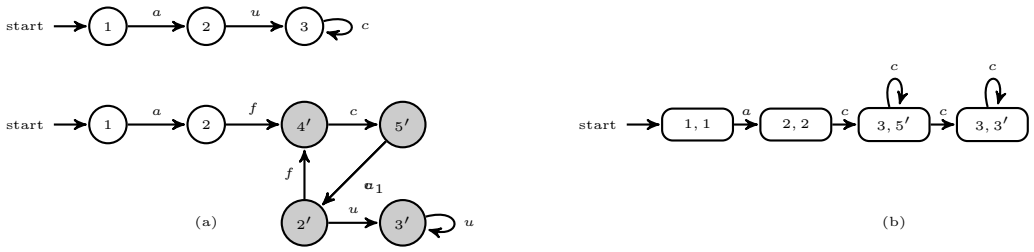


FIGURE 2 – Technique de décomposition du LTS pour générer un twin plant réduit

L'analyse de la diagnosticabilité par rapport au modèle intermédiaire (twin plant) généré par cette approche de décomposition peut se faire de la manière présentée dans la section 3.2, i.e. comme un problème de model-checking $K_{(P\times P)} \models \neg\, EF\, EG\, (c_1(x_1) \wedge (c_2(x_2))$.

4.3 La vérification de la $\mathcal{K}_{min}$-diagnosticabilité

Nous avons présenté, dans [2], une reformulation de l'analyse de la $\mathcal{K}$-diagnosticabilité en un problème de model-checking. L'idée était d'associer à la structure de Kripke (qui correspond au twin plant) une fonction $delay()$ qui permet de compter le nombre de fois (en terme d'états) où le modèle est resté dans une situation ambiguë (voir la section 4.3 dans [2]).

Afin de vérifier la $\mathcal{K}$-diagnosticabilité, pour une valeur donnée de $\mathcal{K}$, il suffit de s'assurer que la valeur retournée par fonction $delay()$ reste toujours inférieure à $\mathcal{K}$. Cette propriété peut être formulée par le problème de model-checking suivant :

$$K_{(P\times P)} \models AG\, delay() \leq \mathcal{K}$$

Si la structure de Kripke satisfait cette spécification CTL alors le modèle est $\mathcal{K}$-diagnosticable, sinon il ne l'est pas.

La $2^{e\grave{m}e}$ question dans l'analyse de la $\mathcal{K}$-diagnosticabilité est de déterminer la valeur minimale de $\mathcal{K}$ garantissant la diagnosticabilité (la $\mathcal{K}_{min}$-diagnosticabilité). Une technique, basée sur l'analyse incrémentale de la $\mathcal{K}$-diagnosticabilité, a été utilisée dans [12]. L'adaptation directe de cette technique dans le contexte de model-checking consiste à introduire une incrémentation dans la spécification à vérifier, i.e. par exemple, pour $\mathcal{K} = 0$: $\neg EF\phi$, pour $\mathcal{K} = 1$: $\neg EF(\phi \wedge EX\phi)$, et ainsi de suite, avec ϕ qui représente la condition de diagnostic. La valeur de $\mathcal{K}$ correspondant à la première spécification satisfaite par le modèle représente le $\mathcal{K}_{min}$ (voir [2] pour plus de détails).

Cette méthode reste inefficace pour la vérification des systèmes à grandes tailles. D'une part, à cause de la complexité exponentielle du model-checking par rapport à la taille de la propriété (le nombre d'opérateurs dans la propriété) et d'une autre part, à cause des multiples exécutions du processus de vérification, ce qui augmente drastiquement le temps d'analyse.

En analysant le concept de la $\mathcal{K}_{min}$-diagnosticabilité dans le contexte de model-checking, on peut constater que la valeur $\mathcal{K}_{min}$ correspond au nombre maximal d'états successivement ambigus qu'une séquence d'évènements peut atteindre. Ceci peut être vu comme le problème de la génération du contre-exemple de longueur maximale par le model-checker. Nous pouvons donc formuler ce problème avec une spécification de la logique temporelle RT-CTL (Real-Time CTL). Cette logique permet d'exprimer des propriétés impliquant des informations quantitatives en termes de nombre d'évènements ou d'états. Par exemple, la détermination de délai maximal (ou minimal) entre deux spécifications CTL. Parmi les outils qui acceptent ce type de spécification, le model-checker NuSMV (que nous allons utiliser, par la suite, dans la partie expérimentation).

Pour l'analyse de la $\mathcal{K}_{min}$-diagnosticabilité, la spécification correspondante est de cette forme : MAX $[\phi\,,\, \neg\phi]$, avec l'expression "MAX" indiquant la recherche du délai maximal et ϕ représente la spécification CTL qui traduit le fait de rentrer dans une situation d'ambiguïté, i.e. un état incertain $(c_1(x_1) \wedge c_2(x_2))$, et $\neg\phi$ indiquant la disparition de la situation d'ambiguïté.

Le problème du model-checking qui reformule donc l'analyse de la $\mathcal{K}_{min}$-diagnosticabilité est le suivant :

$$K_{(P\times P)} \models MAX[\phi\,,\, \neg\phi]$$

Après la vérification, le résultat est soit un entier fini, qui représente donc la valeur $\mathcal{K}_{min}$, soit l'infini pour dire qu'il n'existe pas de délai maximal et par conséquent que le système est non-diagnosticable.

5 Expérimentation

L'approche de décomposition du modèle ainsi que l'algorithme de transformation, présentés dans cet article, ont été implémentés en *C#* et utilisés pour la vérification de la diagnosticabilité et la $\mathcal{K}_{min}$-diagnosticabilité d'un benchmark issu du domaine de contrôle/commande ferroviaire. Dans le but d'évaluer l'apport de notre approche de décomposition de modèles, en terme de réduction de l'espace d'états de twin plant, nous présentons également les résultats de simulation en utilisant le modèle global (voir [2]) et nous comparons les différents résultats.

Pour la vérification, nous avons utilisé le model-checker symbolique NuSMV [4] (version 2.5.4). Considéré parmi les plus performants model-checkers existants, NuSMV représente une ré-implémentation, avec plus d'extensions, du model-checker CMU SMV.

5.1 Présentation du benchmark

Le benchmark représente un modèle d'un système de contrôle automatique d'un passage à niveau équipé de barrières. Il est modélisé par un réseau de Petri borné et vivant. Le modèle contient deux classes de fautes, la $1^{ère}$ classe de fautes, notée Σ_{f_1}, représente une défaillance des capteurs utilisés pour la détection de l'arrivée du train à la zone d'intersection. La $2^{ème}$ classe de fautes, notée Σ_{f_2}, représente la défaillance au niveau de la barrière (une montée de la barrière avant la sortie du train de la zone d'intersection). Une caractéristique intéressante de ce benchmark est son extensibilité à plusieurs rails sans perdre ses propriétés, i.e. la vivacité et la bornitude, ce qui représente un bon outil pour l'évaluation de l'efficacité et la scalabilité de notre approche. Plus de détails par rapport au benchamrk, son fonctionnement, la modélisation et les caractéristiques, se trouvent sur un site dédié au benchmark [11].

Étant donné que le benchmark est modélisé par un réseau de Petri, nous utilisons l'outil TINA pour générer le graphe de marquage (qui sera considéré comme notre modèle LTS de départ). Nous utilisons l'algorithme 1 de transformation pour générer le LTS à base d'états. Un script, développé en *C#* également, nous permettra de générer directement les modules NuSMV qui correspondent aux deux approches à évaluer (approche avec le modèle complet [2] et notre approche avec la décomposition du modèle). Une fois les modules NuSMV sont générés, nous pouvons analyser la diagnosticabilité et chercher la valeur $\mathcal{K}_{min}$ garantissant la diagnosticabilité. Les expérimentations seront effectuées sur le benchmark, en incrémentant à chaque fois le nombre de rails (de 1 à 3 rails).

5.2 Résultats expérimentaux

Les résultats expérimentaux pour la vérification de la diagnosticabilité et la $\mathcal{K}_{min}$-diagnosticabilité de l'approche avec le modèle complet sont représentés dans le tableau 1. Le tableau 2 montre les résultats obtenus avec notre méthode de décomposition (*twin plant réduit*). Les colonnes des tableaux, de la gauche vers la droite, représentent : les classes de fautes, le nombre de rails dans le benchmark, le nombre d'états du graphe de marquage, le nombre d'états du LTS à base d'états (généré par l'algorithme 1), le nombre d'états de la structure de Kripke correspondante au twin plant, le temps d'exécution pour générer l'espace d'états de la structure de Kripke, le verdict sur la diagnosticabilité, le temps d'exécution pour la vérification de la diagnosticabilité avec le model-checker NuSMV, la valeur $\mathcal{K}_{min}$ garantissant la diagnosticabilité. Les expérimentations sont faites sur un 64-bit PC, Ubuntu 14.04, Intel Core i5, 4 coeurs cadencé à 2.5 GHz et 4 GB de RAM.

Σ_{Fi}	n	|MG|	|Q|	|RS|	T_RS	Diag	T_Diag	K_{min}
Σ_{F1}	1	24	38	105	00.02 s	YES	00.01 s	4
	2	216	416	4607	00.20 s	NO	02.65 s	∞
	3	1632	3240	109728	16.88 s	NO	421.60 s	∞
Σ_{F2}	1	24	38	144	00.01 s	YES	00.01 s	6
	2	216	352	5125	00.31 s	YES	06.67 s	13
	3	1632	2604	81583	32.64 s	YES	866.66 s	20

TABLE 1 – Résultats expérimentaux pour le n-LC benchmark avec l'approche du modèle complet

Σ_{Fi}	n	|MG|	|X|	|RS|	T_RS	Diag	T_Diag	K_{min}
Σ_{F1}	1	24	38	21	00.01 s	YES	00.01 s	4
	2	216	416	484	00.14 s	NO	00.06 s	∞
	3	1632	3240	6245	05.91 s	NO	13.65 s	∞
Σ_{F2}	1	24	38	25	0.01 s	YES	00.01 s	6
	2	216	352	484	0.20 s	YES	00.06 s	13
	3	1632	2604	5322	07.45 s	YES	20.41 s	20

TABLE 2 – Résultats expérimentaux pour le LC benchmark avec l'approche de décomposition

5.2.1 Discussion

Concernant l'analyse de la diagnosticabilité, nous remarquons que la classe de fautes Σ_{F_1} est uniquement diagnosticable pour une seule rail, contrairement à la classe de fautes Σ_{F_2} qui est toujours diagnosticable. Dans le cas où le modèle est diagnosticable, une valeur de $\mathcal{K}_{min}$ est retournée, sinon c'est la valeur infinie qui est générée. Ceci est conforme à notre analyse dans la section 4.3.

Une comparaison entre les deux approches nous montre clairement l'efficacité de l'approche avec la décomposition du modèle. Par rapport à l'espace d'états de la structure de Kripke (RS), notre approche présente une réduction importante de l'espace d'états comparée à l'approche avec le modèle complet (une réduction de 80 à 95 %). L'efficacité de notre approche se traduit aussi en terme de temps de vérification (T_Diag). Nous remarquons que le temps de vérification reste dans l'ordre des secondes pour notre approche, contrairement à l'approche avec le modèle complet.

Concernant la scalabilité de notre approche, nous remarquons que le nombre d'états de la structure de Kripke générée explose exponentiellement par rapport au nombre de rails. Cela s'explique par l'augmentation exponentielle du nombre d'états de graphe de marquage. Donc, l'explosion exponentielle de l'espace d'états est une caractéristique du benchmark lui-même et non pas un inconvénient de l'approche.

Remarque : Nous pensons que l'utilisation d'un model-checker à la volée peut améliorer nettement les résultats obtenus par notre approche, étant donné qu'il permet une exploration dynamique de l'espace d'états de twin plant.

6 Travaux connexes

Cet article s'inscrit dans la continuité des travaux relatifs à l'analyse de la diagnosticabilité dans le contexte du model-checking. Une reformulation du problème de la diagnosticabilité comme un problème d'atteignabilité a été proposée par [5], la diagnosticabilité est exprimée par une spécification CTL et vérifiée avec un model-checker symbolique. Dans [10], les auteurs ont utilisé du LTL model-checking, ils ont proposé des spécifications décrivant la violation de la dignosticabilité et un algorithme polynomial pour la vérification et la synthèse de diagnostiqueur. Dans [2], nous avons proposé une extension du travail de [5], dans le but de se conformer avec la définition classique de la diagnosticabilité [15]. Une approche similaire, a été proposée dans [1] où la seule différence était la représentation symbolique du modèle de système. Notre contribution, par rapport à ces travaux, réside dans la reformulation et la vérification de la $\mathcal{K}$-diagnosticabilité, plus précisément la $\mathcal{K}_{min}$-diagnosticabilité.

Dans le but de combattre l'explosion combinatoire de l'espace d'états, durant la construction des modèles intermédiaires, plusieurs techniques ont été développées. Dans [8], l'auteur a essayé de tirer parti de la propriété de symétrie du twin plant en considérant l'occurrence des évènements fautifs uniquement sur l'une des copies du modèle. Par conséquent, il élimine, a priori, le comportement certainement fautif du twin plant. [14, 13] ont proposé une approche, dans le cadre de l'analyse de la codiagnosticabilité, basée sur de multiples compositions synchrones entre le modèle et des motifs de supervision, dont le but est de générer aussi le comportement ambigu. La technique a une complexité polynomiale ($\mathcal{O}(|Q|^2 \times (\Sigma - \Sigma_f))$), avec $|Q|$ le nombre d'états du modèle. L'originalité de notre approche par rapport à ces travaux consiste en la décomposition à priori de modèles tout en tirant part de la transformation des modèles à base d'évènements en des modèles à base d'états.

7 Conclusion

Cet article fait partie de nos travaux sur la reformulation des problèmes de diagnostic et de diagnosticabilité dans le contexte du model-checking. Nous avons, d'abord, présenté un algorithme pour générer un LTS à base d'états à partir de son correspondant à base d'évènements. Par la suite, nous avons proposé une technique qui permet de réduire drastiquement l'espace d'états du twin-plant qui est utilisé pour la vérification de la diagnosticabilité à proprement-parlé, ce qui améliore nettement le processus de la vérification. Finalement, nous avons fourni une spécification dans la logique temporelle RT-CTL pour analyser la $\mathcal{K}_{min}$-diagnosticabilité comme un problème de model-checking.

Par ailleurs, dans l'optique de proposer une reformulation générale et unifiée des problèmes de diagnostic et diagnosticabilité des fautes, nous envisageons d'étendre nos travaux pour prendre en compte des types plus complexes de fautes, comme les fautes répétitives/intermittentes, ou plus généralement des motifs de surveillance. En plus, nous souhaitons utiliser des méthodes d'optimisation connexes au model-checking afin d'améliorer l'efficacité des techniques développées.

Références

[1] G. Bourgne, P. Dague, F. Nouioua, and N. Rapin. Diagnosability of Input Output Symbolic Transition Systems. 1^{st} *Int. Conference on Advances in System Testing and Validation Lifecycle*, pages 147–154, September 2009.

[2] A. Boussif and M. Ghazel. Diagnosability analysis of input/output discrete event system using model checking. *The 5th International Workshop on Dependable Control of Discrete Systems*, 2015.

[3] C. G. Cassandras and S. Lafortune. Introduction to discrete event systems. *Second Edition, Springer*, 2007.

[4] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NUSMV : a new symbolic model checker. *Int. Journal on Software Tools for Technology Transfer*, 2000.

[5] A. Cimatti, C. Pecheur, and R. Cavada. Formal verification of diagnosability via symbolic model checking. *Int. Joint Conference on Artificial Intelligence*, 2003.

[6] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[7] E. Dallal and S. Lafortune. Efficient computation of most permissive observers in dynamic sensor activation problems. In *Int. Workshop on Logical Aspects of Fault-Tolerance*, pages 1–28, 2011.

[8] A. Grastien. Symbolic testing of diagnosability. *20th International Workshop on Principles of Diagnosis (DX-09)*, pages 131–138, 2009.

[9] S. Jiang and Z. Huang. A polynomial algorithm for testing diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, pages 46(8), 1318–1321, 2001.

[10] S. Jiang and R. Kumar. Failure diagnosis of discrete-event systems with linear-time temporal logic specifications. *Automatic Control, IEEE Transactions on*, 49(6) :934–945, 2004.

[11] B. Liu. http ://6814.is-programmer.com/posts/ 45619. 2014.

[12] B. Liu, Mohamed Ghazel, and A. Toguyéni. Toward an efficient approach for diagnosability analysis of DES modeled by Labeled Petri Nets. *Proceeding of the* 13^{th} *European Control Conference*, 2014.

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

[14] Wenbin Qiu and R. Kumar. Decentralized failure diagnosis of discrete event systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A : Systems and Humans*, 36(2) :384–395, 2006.

[15] M. Sampath, R. Sengupta, and S. Lafortune. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, pages 1555–1575, 40(9), 1995.

[16] T. S. Yoo and S. Lafortune. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Transactions on Automatic Control*, pages 47(9), 1491–1495, 2002.

[17] S. H. Zad, R .H. Kwong, and W. M. Wonham. Fault diagnosis in discrete-event systems : Framework and model reduction. *IEEE Transactions on Automatic Control*, 48(7) :1199–1212, 2003.

[18] J. Zaytoon and S. Lafortune. Overview of fault diagnosis methods for Discrete Event Systems. *Annual Reviews in Control*, pages 308–320, 2013.