Estimabilité à état unique des systèmes à évènements discrets

Valentin Bouziat[1] **Xavier Pucel**[1] **Stéphanie Roussel**[1] **Louise Travé-Massuyès**[2]

[1] ONERA / DTIS, Université de Toulouse, F-31055 Toulouse – France
[2] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

prenom.nom@onera.fr louise@laas.fr

Résumé

Les systèmes autonomes prennent une place de plus en plus importante dans nos environnements quotidiens et imposent des spécifications particulières. Il est donc important de définir et vérifier des propriétés liées à la sécurité, la sûreté ou encore la fiabilité de tels systèmes. Dans cet article, nous adoptons un formalisme de modélisation basé sur les systèmes à évènement discrets (SED) et nous intéressons à des propriétés relatives au diagnostic. Une nouvelle propriété appelée *estimabilité à état unique* est introduite. Lorsque les observations produites par un système ne permettent pas d'être certain de l'état dans lequel il se trouve (plusieurs états sont compatibles avec les observations) l'estimation de son état est ambiguë. Cette propriété exprime alors la possibilité de limiter l'estimation à un seul état sans rencontrer d'incohérence dans la suite de l'exécution. Limiter l'estimation à un seul état facilite entre autres la décision notamment par l'utilisation éventuelle d'un planificateur déterministe dans une architecture autonome complète. Une condition nécessaire et suffisante pour l' *estimabilité à état unique* des SED est fournie ainsi qu'un algorithme récursif permettant de la vérifier et, testé sur différents jeux de données.

Abstract

Specific requirements must guide the design of autonomous systems as they are increasingly present in our everyday environment. Their properties must be carefully defined and checked to guarantee safety, security and dependability. In this paper, we adopt a discrete event system modelling framework and focus on properties that are related to diagnosis. A new property called *single state trackability* is introduced. When available observations may lead to an ambiguous estimate, i.e. several admissible state candidates, this property assesses the possibility of reducing the estimate to a single state without this leading to a dead-end in the continuation of the execution. A single state estimate advantageously facilitates decision making and allows the use of a deterministic planner in the autonomous architecture. We provide a necessary and sufficient condition for *single state trackability* of a discrete event system and we propose a recursive algorithm to check this property. The algorithm is validated with a set of benchmarks.

1 Introduction

La capacité d'un système autonome à réagir à des événements imprévus dépend de sa capacité à estimer son état interne. Ceci influe fortement sur la prise de décision et peut être nécessaire à sa survie. Nous nous concentrons sur des systèmes dont la dynamique peut être représentée par un formalisme de modélisation à événements discrets à temps échantillonné.

Dans ce contexte, l'estimation d'état et le diagnostic ont fait l'objet de nombreux travaux [17, 18, 6, 5]. En général, les observations issues du système ne suffisent pas à garantir l'observabilité de l'état [12, 10], ce qui signifie que l'estimation d'état est ambiguë : plusieurs estimations sont possibles à chaque pas de temps. Par conséquent, le nombre d'états candidats augmente de façon exponentielle au fil de l'exécution du système. La plupart des travaux s'attaquent à ce problème en sélectionnant un nombre limité de meilleurs candidats en fonction d'un critère de préférence, par exemple des probabilités comme dans [16, 8]. Cependant, lorsque l'état réel ne fait pas partie du sous-ensemble sélectionné, cela peut conduire à une impasse dans la suite de l'exécution. La solution proposée par [8] pour ce problème est de revenir en arrière et de récupérer la trajectoire de l'état qui permet à l'estimation de reprendre.

Dans ce travail, nous réduisons à l'extrême le nombre d'estimations et nous proposons de n'en conserver qu'une seule, comme dans [1]. Cela découle de plusieurs raisons. Premièrement, les architectures autonomes ont une quantité très limitée de mémoire et il n'est pas souhaitable de stocker l'historique complet de l'exécution du système puisque celle-ci croît au fil du temps. Deuxièmement,

l'estimation doit être incrémentale et fondée uniquement sur l'estimation précédente et sur l'observation actuelle à chaque étape. Enfin, l'estimation d'un état unique facilite avantageusement la prise de décision et permet l'utilisation de planificateurs déterministes dans l'architecture autonome.

Nous pensons qu'un retour en arrière en cas d'impasse n'est pas une solution viable compte tenu des contraintes de temps réel. C'est pourquoi nous introduisons une nouvelle propriété appelée *estimabilité à état unique* qui évalue la possibilité de réduire l'estimation à un seul état sans que cela n'aboutisse à une impasse dans la poursuite de l'exécution. Des observations ultérieures peuvent prouver que l'état estimé était erroné, invalidant ainsi les décisions précédemment prises. Bien qu'il soit souvent impossible d'assurer l'exactitude de l'état estimé, la propriété présentée dans cet article évalue s'il est possible de sélectionner une estimation unique de l'état du système de manière à ce qu'elle ne soit jamais remise en cause par une séquence d'observations ultérieure. Notre principale contribution est de fournir une condition nécessaire et suffisante pour l'*estimabilité à état unique* d'un système à événements discrets et nous proposons un algorithme récursif pour vérifier cette propriété à partir du modèle du système et de son estimateur.

Cet article est structuré comme suit. Les travaux connexes sont présentés en Section 2. La Section 3 présente le formalisme de modélisation pour les systèmes à événements discrets et les estimateurs incrémentaux. Ensuite, la propriété d'*estimabilité à état unique* est présentée en Section 4. Les concepts utiles à la vérification de cette propriété sont introduits en Section 5. Enfin, un algorithme de vérification est proposé en Section 6 et validé sur des systèmes expérimentaux en Section 7.

2 Travaux connexes

Bien que la propriété d'*estimabilité à état unique* soit nouvelle, les notions connexes d'observabilité et de diagnosticabilité des SED ont fait l'objet de plusieurs travaux. [12] traite de l'observabilité dans le cadre de l'observation partielle de l'état et de l'observation totale des événements. La notion d'états (faiblement) indiscernables tient compte de toutes les observations futures, et l'observabilité (faible) est obtenue lorsque toutes les paires d'états sont (faiblement) indiscernables. La notion de coobservabilité est définie de la même façon pour des observations antérieures. Une forte coobservabilité implique que l'état actuel peut être déterminé de façon certaine à partir de l'historique des observations, ce qui est beaucoup plus fort que l' *estimabilité à état unique* car celle-ci n'impose pas à l'état estimé de correspondre parfaitement à l'état du système.

Dans [10], seuls quelques événements sont observés, et un système est dit observable lorsqu'un observateur qui suit tous les états possibles visite régulièrement un état connu avec certitude. La notion d'observabilité avec délai est similaire, mais permet à l'état unique connu de se situer dans le passé. La définition des observateurs résilients reflète le principe de robustesse aux observations perturbées. L'observabilité exige la connaissance de l'état exact du système, ce qui n'est pas nécessaire pour l'*estimabilité à état unique*. Inversement, l'*estimabilité à état unique* exige que, parmi deux états indiscernables, l'un explique toutes les observations produites par l'autre, ce qui n'est pas nécessaire dans les définitions de l'observabilité.

La diagnosticabilité, telle que définie dans [13], diffère de notre approche par plusieurs aspects. Tout d'abord, elle cible les fautes permanentes, tandis que notre approche peut également être appliquée pour estimer la présence de fautes intermittentes. Deuxièmement, elle nécessite la construction complète d'un diagnostiqueur, ce qui pose un problème de passage à l'échelle ([14] atténue ce problème). Enfin, elle tient compte d'un délai limité entre l'apparition d'une faute et son diagnostic. Bien que cette idée soit intéressante et constitue une exigence réaliste, il n'existe aucun moyen universel de l'appliquer aux phénomènes intermittents. Les méthodes issues de [3] et [7], ont été difficile à appliquer à notre contexte de systèmes autonomes.

3 Estimation d'état dans les SED

Dans cet article, nous adoptons une approche de modélisation similaire à celle du *model-checking*, dans laquelle les états sont définis par des affectations sur un ensemble de variables booléennes S. Certaines variables sont observées (toujours les même à chaque pas de temps) et forment un ensemble O tel que $\mathsf{O} \subseteq \mathsf{S}$. Les variables qui ne sont pas observés sont alors estimées. On appelle observation une affectation aux variables d'états observés. Nous supposons que le système suit une dynamique discrète et que tous les pas de temps sont de même durée.

définition 1 (Système à évènements discrets) *Un système à évènements discrets (SED) est représenté par un tuple (S, Δ, s_0) où S est l'espace d'état, $\Delta \subseteq S \times S$ est la relation de transition et $s_0 \in S$ est l'état initial.*

On suppose qu'il n'existe aucune contrainte à propos de la vivacité du système, cet hypothèse n'influant pas le problème traité et les définitions associées.

définition 2 (Langage d'exécution) *Pour un SED (S, Δ, s_0), le langage d'exécution $\mathcal{L}(\Delta) \subseteq S^*$ est l'ensemble des séquences finis d'états cohérentes :*

$$\mathcal{L}(\Delta) = \{(s_0, \ldots, s_n) | n \in \mathbb{N}^+, \forall i \in [0, n-1], (s_i, s_{i+1}) \in \Delta\}$$

Notations : Soit $seq \in \mathcal{L}(\Delta)$ une séquence d'états, $seq[i]$ dénote le $i^{\text{ème}}$ état de la séquence commençant à l'état initial $seq[0]$, la taille de la séquence est notée $|seq|$.

Une affectation sur les variables de 0 est appelé une observation. On note O l'ensemble des observations possibles et on note *obs* la fonction d'observation de S vers O qui projette un état sur son observation et nous l'étendons naturellement de S^* vers O^* comme suit : $|obs(seq)| = |seq|$ et $obs(seq)[i] = obs(seq[i])$ pour $i \in [0, |seq|[$.

Nous définissons ensuite le langage des observations pour un SED.

définition 3 (Langage des observations) *Pour un SED, le langage des observations $\mathcal{L}_{obs}(\Delta) \subseteq O^*$ est l'ensemble des séquences d'observations cohérentes.*

$$\mathcal{L}_{obs}(\Delta) = \{obs(seq) | seq \in \mathcal{L}(\Delta)\}$$

Pour une séquence d'observations *sobs* de $\mathcal{L}_{obs}(\Delta)$ et une observation o de O, on note *sobs.o* la concaténation de *sobs* et o.

Nous définissons maintenant le problème de l'estimation d'état dans les SED. L'estimation est un processus incrémental qui base son raisonnement uniquement sur l'état précédent (estimé) et l'observation actuelle.

définition 4 (Candidats à l'estimation) *Étant donnés un SED $(S, \Delta, s_0,)$, un état $s \in S$ et une observation $o \in O$, nous définissons l'ensemble des candidats à l'estimation $cands(s, o)$ comme l'ensemble des états dans lesquels le système pourrait se trouver, en supposant qu'il était dans l'état s à l'instant précédent, et produit l'observation o à l'instant courant. Formellement :*

$$cands(s, o) = \{\hat{s} \in S | (s, \hat{s}) \in \Delta \text{ et } obs(\hat{s}) = o\}$$

Pour une paire donnée (s, o), il existe souvent plusieurs estimations possibles. Dans notre approche, un seul candidat est sélectionné à chaque étape. Ainsi, l'estimation d'état peut être modélisée comme suit.

définition 5 (Fonction d'estimation) *Soit (S, Δ, s_0) un SED. Une fonction d'estimation est une fonction $estim : (S \times O) \to S$ qui sélectionne un état unique parmi les candidats, c.a.d. pour chaque $s \in S$ et $o \in O$, si $cands(s, o) \neq \emptyset$ alors $estim(s, o) \in cands(s, o)$.*

Un estimateur est complètement défini par sa fonction d'estimation. Il reçoit les observations en entrée, et l'état estimé est réutilisé à l'étape suivante pour calculer l'estimation.

définition 6 (Séquence d'estimations) *Soient (S, Δ, s_0) un SED, $estim : (S \times O) \to S$ une fonction d'estimation, et sobs une séquence d'observations de $\mathcal{L}_{obs}(\Delta)$. La séquence d'estimations pour sobs est l'unique séquence d'états sest telle que :*

- *$sest[0] = s_0$ et*
- *pour $i \in [1, |sobs|[$, si $cands(sest[i-1], sobs[i]) \neq \emptyset$ alors $sest[i] = estim(sest[i-1], sobs[i])$ sinon sest est indéfinie.*

4 Estimabilité à état unique

Le simple fait qu'un estimateur sélectionne une estimation unique crée des scénarios dans lesquels l'estimation peut différer de l'état réel du système, le système peut par la suite produire une observation incompatible avec l'état précédent estimé. Dans de tels scénarios, l'estimateur est incapable de produire une estimation, car l'ensemble des candidats à l'estimation est vide et la fonction d'estimation est alors indéfinie [1]. Formellement, si nous notons s l'état réel du système et $\hat{s}$ l'estimation de l'état faite par l'estimateur, si $s \neq \hat{s}$, le système peut évoluer dans un état s' et produire une observation $obs(s') = o$ telle que $cands(\hat{s}, o) = \emptyset$. Nous appelons une telle situation une impasse, et le chemin observable est appelé un chemin sans issue.

définition 7 (Chemin sans issue) *Soient (S, Δ, s_0) un SED, $estim : (S \times O) \to S$ une fonction d'estimation, $k \in \mathbb{N}$, sobs une séquence d'observations de $\mathcal{L}_{obs}(\Delta)$ telle que $k = |sobs|$ et o une continuation de sobs telles que $sobs.o \in \mathcal{L}_{obs}(\Delta)$. sobs.o est un chemin sans issue si et seulement si :*

- *il existe une séquence d'estimations $sest = (\hat{s_0}, \ldots, \hat{s_k})$ pour sobs ;*
- *il n'existe aucun candidat à l'estimation pour o, c.a.d. $cands(\hat{s_k}, o) = \emptyset$.*

Les chemins sans issue illustrent le problème dans lequel l'estimateur fait une hypothèse au sujet de l'état réel du système, et découvre plus tard que celle-ci était fausse. Cela peut poser problème si une décision importante a été prise à la suite de cette hypothèse. La plupart du temps, il est impossible (et pas nécessaire) de connaître avec certitude l'état réel du système pour le faire fonctionner. C'est pourquoi nous introduisons le concept d'*estimabilité à état unique*, c.a.d. la capacité d'estimer de manière unique l'état du système et de ne jamais rencontrer un chemin sans issue.

définition 8 (Estimabilité à état unique) *Un SED est* estimable à état unique [2] *si et seulement s'il existe une fonction d'estimation $estim : (S \times O) \to S$ telle qu'aucune séquence d'observations $sobs \in \mathcal{L}_{obs}(\Delta)$ n'est un chemin sans issue.*

exemple 1 *Considérons le SED représenté en figure 1, décrit comme un automate de Moore [9] sans alphabet d'entrée.*

Au début de l'exécution, le système produit la séquence d'observations (a, a, b). Pour les 3 premiers pas de temps,

1. Notons que de tels scénarios peuvent se produire puisque l'estimateur ne garde pas en mémoire tous les candidats à l'estimation.

2. Dans cet article, « estimable » et « estimabilité » font respectivement référence à estimable à état unique et estimabilité à état unique

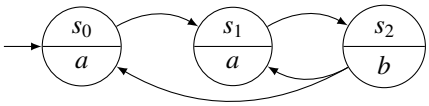

FIGURE 1 – Un SED non estimable avec comme états $S = \{s_0, s_1, s_2\}$ et observations $O = \{a, b\}$. Δ et *obs* sont représentés comme dans une machine de Moore.

l'estimation est triviale puisque le système ne peut qu'emprunter la séquence d'états (s_0, s_1, s_2)*. Mais pour la séquence d'observations* (a, a, b, a)*, l'ensemble des candidats est alors* $cands(s_2, a) = \{s_0, s_1\}$*. Un choix doit alors être fait. Considérons donc les deux fonctions d'estimation possibles* $estim_0(s_2, a) = s_0$ *et* $estim_1(s_2, a) = s_1$*, ainsi que les deux séquences d'observations* (a, a, b, a, a) *et* (a, a, b, a, b) *respectivement produites par les séquences d'états* $(s_0, s_1, s_2, s_0, s_1)$ *et* $(s_0, s_1, s_2, s_1, s_2)$*.*

Avec $estim_0$*, la séquence d'observations* (a, a, b, a) *est estimée comme* (s_0, s_1, s_2, s_0)*, mais du fait que* $cands(s_0, b) = \emptyset$*, la séquence d'observations* (a, a, b, a, b) *est une impasse.*

Avec $estim_1$*, la séquence d'observations* (a, a, b, a) *est estimée comme* (s_0, s_1, s_2, s_1)*, mais du fait que* $cands(s_1, a) = \emptyset$*, la séquence d'observations* (a, a, b, a, a) *est une impasse.*

Puisque toutes les fonctions d'estimation possibles rencontrent des impasses, le système n'est pas estimable.

5 Vérification de l'estimabilité à état unique

Vérifier l'estimabilité d'un système est une tâche difficile. Nous introduisons dans cette section certaines conditions nécessaires et une condition d'équivalence.

définition 9 (État atteignable) *Un état* $\hat{s}$ *est atteignable par la séquence d'observations* $sobs \in \mathcal{L}_{obs}(\Delta)$ *si et seulement si il existe une séquence d'états* $seq \in \mathcal{L}(\Delta)$ *telle que* $obs(seq) = sobs$*, et* seq *se termine par* $\hat{s}$*. L'ensemble des états atteignables via* $sobs$ *est noté* $reach(sobs)$*.*

Notons que si l'ensemble des séquences d'observations est infini, l'ensemble des états atteignables est toujours fini car c'est un sous-ensemble de 2^S. Cet ensemble peut être énuméré en utilisant une construction par sous-ensembles cohérents avec *obs*.

définition 10 (État non-bloquant) *Soient* (S, Δ, s_0) *un SED, et* $sobs \in \mathcal{L}_{obs}(\Delta)$ *une séquence d'observations. Un état* $\hat{s}$ *est* non-bloquant *pour* $sobs$ *si celui-ci est atteignable par* $sobs$ *et pour chaque observation ultérieure* o*, il existe au moins un candidat. Formellement,* $\hat{s} \in reach(sobs)$ *est* non-bloquant *si et seulement si :*

$$\forall o \in O \text{ si } sobs.o \in \mathcal{L}_{obs}(\Delta), \text{ alors } cands(\hat{s}, o) \neq \emptyset$$

Un état atteignable via *sobs* mais qui n'est pas non-bloquant pour *sobs* est appelé *état bloquant*. Notons qu'un état $\hat{s}$ peut être non-bloquant pour une séquence d'observations $sobs_1$ et bloquant pour une autre séquence $sobs_2$. Les états non-bloquants sont importants car les estimateurs doivent toujours les sélectionner, sinon ils pourraient rencontrer un chemin sans issue.

proposition 1 (Condition sur les états non-bloquants) *S'il existe une séquence d'observations* $sobs \in \mathcal{L}_{obs}(\Delta)$ *telle que* $reach(sobs)$ *ne contient que des états bloquants, alors le système n'est pas estimable.*

Intuitivement, la proposition 1 signifie que si l'ensemble des états atteignables $reach(sobs)$ pour une séquence d'observations *sobs* de $\mathcal{L}_{obs}(\Delta)$ ne contient que des états bloquants, alors quel que soit l'état $\hat{s} \in reach(sobs)$ choisi, il existe une continuation $sobs.o \in \mathcal{L}_{obs}(\Delta)$ qui est une impasse. Il est donc impossible, dans cette configuration, de construire un estimateur qui ne rencontrerai pas d'impasse.

proposition 2 (Transitions entre états non bloquants) *Soit* $sobs \in \mathcal{L}_{obs}(\Delta)$ *une séquence d'observations et* $o \in O$ *une observation telle que* $sobs.o \in \mathcal{L}_{obs}(\Delta)$*. Si le système est estimable, alors il existe une paire d'états* $(s_1, s_2) \in \Delta$ *telle que* s_1 *est non-bloquant pour* $sobs$ *et* s_2 *est non-bloquant pour* $sobs.o$*.*

La proposition 2 étend la proposition 1 aux transitions, et pourrait être étendue à des chemins de longueur arbitraire. D'une part, lors de la construction de l'ensemble de tous les $reach(sobs)$ afin de vérifier la proposition 1, et en particulier lors de la construction d'une transition entre $reach(sobs)$ et $reach(sobs.o)$, il est facile de vérifier la proposition 2 à la volée. La recherche de chemins plus longs nécessiterait une recherche spécifique.

D'autre part, la vérification de cette propriété pour les chemins de n'importe quelle longueur ne fournit pas une condition suffisante pour l'estimabilité. Dans le SED présenté en figure 2, il est toujours possible pour une séquence d'observations de construire des paires d'états respectant la proposition 2 cependant le SED n'est pas estimable. En effet, si nous prenons la séquence d'observations (a, b, a, b, c) produite par la séquence d'états non bloquants $(s_0, s_1, s_0, s_2, s_4)$, il est cependant impossible de construire une fonction $estim : (S \times O) \rightarrow S$ permettant d'emprunter cette séquence puisque qu'a partir de s_0 et en recevant l'observation b, il faudrait choisir tantôt s_1, tantôt s_2.

Afin de fournir une condition nécessaire et suffisante pour assurer l'estimabilité, il est requis de s'assurer que le langage des observations (l'ensemble des séquences d'observations) est supporté par un estimateur.

définition 11 (Langage accepté par l'estimateur) *Soit* (S, Δ, s_0) *un SED,* $sobs \in \mathcal{L}_{obs}(\Delta)$ *une séquence d'observations et* $estim : (S \times O) \rightarrow S$ *une fonction d'estimation.*

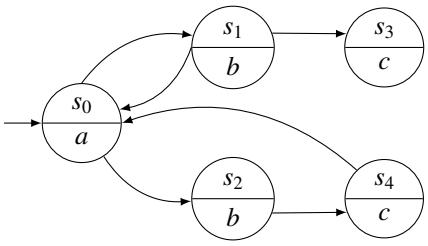

FIGURE 2 – Un SED non estimable mais satisfaisant la condition de la proposition 2.

Le langage accepté par cette fonction d'estimation, noté $\mathcal{L}_{obs}(estim)$*, est l'ensemble de toutes les séquences d'observations possibles pour lesquelles il existe une séquence d'estimations. Formellement :*

$$\mathcal{L}_{obs}(estim) = \{sobs \in \mathcal{L}_{obs}(\Delta) \mid |sobs| > 1 \ \ et \ \exists sest \in \mathcal{L}(\Delta)$$

$$t.q.\ sobs = obs(sest)$$

$$et\ pour\ i \in [1, |sobs|], estim(sest[i-1], sobs[i]) = sest[i]\}$$

Nous définissons ensuite une condition nécessaire et suffisante pour l'estimabilité à état unique.

proposition 3 (Condition d'estimabilité) *Un SED* (S, Δ, s_0) *est* estimable à état unique *si et seulement s'il existe une fonction d'estimation dont le langage accepté est égal au langage des observations du système :*

$$\exists\, estim : (S \times O) \to S \ telle\ que\ \mathcal{L}_{obs}(estim) = \mathcal{L}_{obs}(\Delta)$$

Il devient maintenant évident qu'il est possible de trouver des impasses pour un estimateur donné à l'aide d'un algorithme semblable à celui de la vérification de l'égalité des langages réguliers.

La preuve est assez directe : si la condition de la proposition 3 est satisfaite, alors la fonction d'estimation fournit une séquence d'estimations pour tous les éléments de $\mathcal{L}_{obs}(\Delta)$. Il n'y a donc pas d'impasse. La principale difficulté est de trouver une telle fonction d'estimation ou de prouver qu'elle n'existe pas.

Afin de vérifier efficacement l'estimabilité, notre approche consiste à construire récursivement des fonctions d'estimation partiellement définies et les tester. Nous définissons les impasses pour ces fonctions d'estimation partielles et introduisons une proposition utilisée dans notre algorithme.

définition 12 (Prolongement d'une fonction d'estimation) *Soient estim et pestim deux fonctions d'estimation de* $(S \times O)$ *vers* S*. On dit que estim* prolonge *pestim si et seulement si pour tout couple* (s, o) *de* $S \times O$ *tel que* $pestim(s, o)$ *est définie, alors* $estim(s, o)$ *est également définie et* $pestim(s, o) = estim(s, o)$

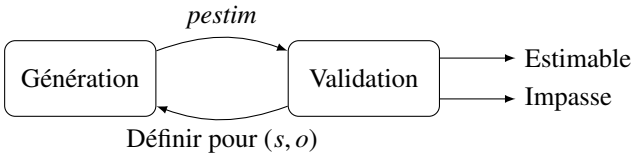

FIGURE 3 – Structure de l'algorithme.

proposition 4 *Soient sobs une séquence d'observations de* $\mathcal{L}_{obs}(\Delta)$*, pestim et estim deux fonctions d'estimation de* $(S \times O)$ *vers* S *telles que estim prolonge pestim. Si sobs est un chemin sans issue pour pestim, alors sobs est également un chemin sans issue pour estim.*

6 Algorithme

Nous décrivons un algorithme pour vérifier l'estimabilité d'un système. Le principe est de trouver des impasses pour des fonctions d'estimation partiellement définies. Notre algorithme est structuré en deux composantes : la première produit des fonctions d'estimation partiellement définies et la seconde vérifie si une fonction d'estimation partielle donnée satisfait la proposition 3. La structure de l'algorithme est illustré en figure 3.

La composante de génération produit récursivement des fonctions d'estimation partielles *pestim* et les envoie pour validation. La composante de validation a trois types de retours : « Estimable » signifie que le système peut être surveillé avec la *pestim* courante ; « Impasse » signifie que *pestim* rencontre une impasse ; « Définir pour (s, o) » signifie que *pestim* doit être prolongée pour inclure une estimation pour (s, o). Selon le résultat de la validation, la composante de génération peut terminer ou produire récursivement d'autres fonctions d'estimation partielles.

6.1 Génération de fonction d'estimation

La composante de génération passe par une fonction GENESTIM décrite dans l'algorithme 1 qui lance le processus de vérification et appelle la composante de validation. Lors de la première itération, la composante de génération produit la fonction d'estimation partielle vide (c.a.d. définie sur $\emptyset$), et l'envoie à la composante de validation.

Si la composante de validation retourne « Estimable », cela signifie que la fonction d'estimation partielle courante *pestim* définit un estimateur qui accepte $\mathcal{L}_{obs}(\Delta)$, et que les paires d'états pour lesquelles *pestim* n'est pas définie ne sont pas nécessaires pour l'estimation. De ce fait, toute extension de *pestim* satisfait la proposition 3, et le système est estimable.

Si la composante de validation retourne « Impasse », cela signifie qu'il existe un chemin sans issue pour *pestim*. Par la proposition 4, aucune extension de cette fonction ne peut satisfaire la proposition 3. La récursion se termine.

Algorithme 1 Composante de génération : fonction GENESTIM

```
1:  Entrees
2:    Σ = (S, Δ, s_0) : un SED
3:  Sorties
4:    pestim : une fonction d'estimation partielle
5:  function GENESTIM(Σ, pestim)
6:    switch CHECKESTIM(Σ, pestim, s_0, {s_0}) :
7:      case Estimable : return true
8:      case Impasse : return false
9:      case Définir pour (s,o) :
10:       for c in cands(s, o) do
11:         extension ← pestim ∪ ((s, o), c)
12:         if GENESTIM(Σ, extension) then
13:           return true
14:       end for
15:       return false
```

Si la composante de validation retourne « Définir pour (s, o) », cela signifie qu'il existe une paire (s, o) telle que s est atteignable, $cands(s, o)$ contient plusieurs candidats, et *pestim* n'est pas définie pour (s, o). Dans ce cas, nous devons vérifier s'il existe une option d'estimation pour (s, o) qui satisfait la proposition 3, et les extensions de *pestim* sont alors générées récursivement.

L'algorithme explore récursivement toutes les options d'estimation, Ainsi s'il termine sans qu'aucune fonction d'estimation satisfaisant la proposition 3 n'ait été trouvée, alors le système n'est pas estimable.

6.2 Validation de fonction d'estimation

La composante de validation contient une fonction CHECKESTIM qui vérifie si le langage accepté par une fonction d'estimation partielle est égal au langage des observations du système. L'algorithme est basé sur une légère modification de l'algorithme classique pour tester l'égalité des langages réguliers [2] afin de tenir compte des cas où plusieurs candidats à l'estimation existent et pour lesquelles la fonction d'estimation partielle est indéfinie.

L'approche consiste à simuler l'exécution de l'estimateur et du système, tout en s'assurant qu'ils sont synchronisés sur les mêmes séquences d'observations. Puisque pour chaque séquence d'observations *sobs*, il existe (au plus) une séquence d'estimations unique, nous n'avons besoin de garder trace que d'un seul état dans l'estimateur. Ainsi, l'état de l'estimateur atteint via une séquence d'observations *sobs* est associé à l'ensemble des états systèmes $reach(sobs)$ (voir définition 9).

L'algorithme explore récursivement des paires $(estSt, sysSts)$ où $sysSts = reach(sobs)$ pour une $sobs \in \mathcal{L}_{obs}(\Delta)$, et où $estSt \in sysSts$. Pour s'assurer que l'algorithme termine, une variable globale « *visited* »

Algorithme 2 Composante de validation : fonction CHECKESTIM

```
1:  Entrees
2:    Σ = (S, Δ, s_0) : un SED
3:    estim : une fonction d'estimation partielle
4:    estSt ∈ S : état de l'estimateur
5:    sysSts ⊆ S : états du système
6:  Sorties
7:    « Estimable » , « Impasse » ou « Définir pour (s, o) »
8:  Global
9:    visited ∈ S × 2^S ← ∅
10: function CHECKESTIM(Σ, estim, estSt, sysSts)
11:   if (estSt, sysSts) ∈ visited then
12:     return Estimable
13:   visited ← visited ∪ (estSt, sysSts)
14:   nextObs ← {obs(s')|∃s ∈ sysSts, (s, s') ∈ Δ}
15:   for o in nextObs do
16:     estNxts ← NEXTESTSTATES(estSt, o)
17:     sysNxts ← {s' | obs(s') = o ∧
18:                                  ∃s ∈ sysSts, (s, s') ∈ Δ}
19:     if estNxts = ∅ then
20:       return Impasse
21:     else if estNxts = {estNxt} then
22:       rec ← CHECKESTIM(Σ, estim, estNxt, sysNxts)
23:       switch rec :
24:         case Impasse : return Impasse
25:         case Définir pour (s, o) :
26:           return Définir pour (s, o)
27:         case Estimable : continue
28:     else
29:       return Définir pour (estSt, o)
30:   end for
31:   return Estimable
```

stocke les paires qui ont déjà été explorées. L'algorithme recherche les séquences *sobs* pour lesquelles l'estimateur n'est pas défini, de sorte que la condition de terminaison n'est remplie que si une telle séquence n'existe pas. Dans ce cas, les langages sont égaux, et nous retournons « Estimable » (lignes 11 à 12).

A chaque itération, CHECKESTIM calcule les observations que le système peut produire (ligne 14). Ensuite, pour chaque observation produite, il appelle une fonction `NextEstStates`$(estSt, o)$ (ligne 16) définie comme suit. Si *pestim* est définie pour $(estSt, o)$, celle-ci retourne $\{pestim(estSt, o)\}$ sinon $cands(estSt, o)$. L'algorithme teste ensuite le nombre d'état possibles pour l'estimateur :

— Si $estNxts = \emptyset$ (ligne 19) : cela signifie que si *sobs* est la séquence d'observations courante (dans l'appel récursif), *sobs*.*o* est un chemin sans issue (voir définition 7),« Impasse » est retourné.

— si $estNxts = \{estNxt\}$ (ligne 21) : cela signifie que

pour la paire d'état courante $(estSt, o)$, soit il n'y a qu'un unique successeur, soit *pestim* est définie. Dans ce cas la recherche continue avec des appels récursifs.

— si $|estNxts| > 1$ (ligne 28) : cela signifie qu'il y a plusieurs candidats à l'estimation, et que *pestim* n'est pas définie pour $(estSt, o)$. « Définir pour $(estSt, o)$ » est retourné, afin que la composante de génération produise des fonctions d'estimation définies pour cette paire.

6.3 Améliorations

Les performances de l'algorithme décrit ci-dessus peuvent être considérablement améliorées avec quelques mécanismes. Une vérification préliminaire est ajoutée pour vérifier les propositions 1 et 2 pour chaque ensemble d'états qui peut être atteint via une séquence d'observations.

Dans nos expériences, la principale source de complexité est le nombre de fonctions d'estimation partielles à tester. Le seul mécanisme dont nous disposons pour élaguer les fonctions d'estimations partielle est de trouver des impasses le plus tôt possible.

Tout d'abord, dans l'algorithme 1, lors de la détection d'une impasse, les triplets d'estimation qui ne sont pas utilisés dans le chemin sans issue peuvent être supprimés de *pestim*, et nous pouvons mémoriser uniquement cette fonction d'estimation partielle tronquée. De cette façon, par la proposition 4, nous pouvons tester à la ligne 11 si certaines extensions sont compatibles et étendre celle que nous venons de mémoriser afin d'épargner quelques appels à l'algorithme 2.

Deuxièmement, il existe des fonctions d'estimation partielles qui contiennent à la fois un chemin sans issue et des paires non définies (s, o). Dans ce cas, dans l'algorithme 2, soit « Impasse » , soit « Définir pour (s, o) » est retourné selon l'ordre de traversée de la ligne 15. Afin de favoriser « Impasse » , au lieu de retourner « Définir pour (s, o) » , nous stockons la fonction partielle dans une variable globale, et continuons la recherche. Lorsque la recherche récursive globale est terminée, si le résultat est « Impasse » nous le retournons, sinon si la variable globale contient une paire (s, o) nous retournons « Définir pour (s, o) » et sinon nous retournons « Estimable » . De cette façon, nous détectons les impasses beaucoup plus tôt dans la recherche.

Les propriétés sur les états bloquants des propositions 1 et 2 peuvent également être utilisées pour réduire l'espace de recherche : dans l'algorithme 1 à la ligne 10, les états bloquants peuvent être ignorés car ils mèneront forcément à une impasse. Puisque les fonctions d'estimation sont générées en largeur pour un état précédent et une observation, cela permet d'élaguer une branche dans l'espace de recherche pour chaque état bloquant.

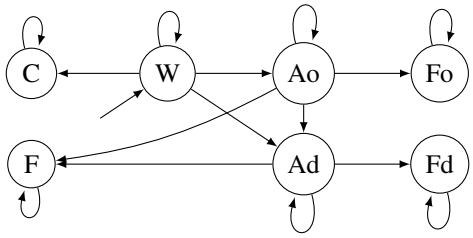

FIGURE 4 – Le workflow des états des actions qui peuvent être (W)aiting, (C)ancelled, (A)ctive (o)k, (F)inished (o)k, (F)ailed, (A)ctive (d)elayed, ou (F)inished (d)elayed.

7 Expérimentations

Nous avons testé notre approche sur un exemple inspiré du langage de robotique autonome PLEXIL. [15]. Ce cadre est organisé autour du concept d'actions, qui ont un workflow hiérarchique complexe. Nous utilisons un workflow d'actions séquentielles simplifié décrit dans l'exemple 2.

exemple 2 *Nous considérons un cadre robotique où les plans des robots sont représentés sous forme de séquence d'actions. La figure 4 illustre le workflow des actions. Nous considérons un robot avec un plan constitué de deux actions séquentielles : move et inspect, dont les états sont représentés par les variables mv et ins. L'état de santé du robot est décrit par trois variables booléennes hnav, hsens et hpow représentant respectivement si les fonctions de navigation, de capteur et d'alimentation électrique fonctionnent normalement. Une autre variable booléenne, pert, indique si le robot est sujet à des perturbations (terrain glissant, obstacle, vent) à chaque pas de temps. Nous notons move* $= W$ *pour exprimer que l'action move passe dans l'état W. Nous notons* $Y(v)$ *la valeur à l'étape précédente pour la variable* $v \in \{mv, ins, hnav, hsens, hpow, pert\}$. *Nous utilisons la fonction* $start(v) = K$ *qui signifie* $Y(v) \neq K \wedge v = K$. *Le comportement du système est décrit par l'automate pour chaque action, associé aux contraintes suivantes :*

$$mv \in \{W, Ao, Ad\} \rightarrow ins = W \quad (1)$$

$$mv \in \{C, F\} \rightarrow ins = C \quad (2)$$

$$start(mv = Fo) \rightarrow ins = Ao \quad (3)$$

$$start(mv = Fd) \rightarrow ins = Ad \quad (4)$$

$$start(mv = Ad) \rightarrow (\neg hnav \vee \neg hpow \vee pert) \quad (5)$$

$$start(mv = F) \rightarrow \neg hpow \quad (6)$$

$$start(ins = Ad) \rightarrow (\neg hsens \vee \neg hpow \vee pert) \quad (7)$$

$$start(ins = F) \rightarrow \neg hpow \quad (8)$$

$$hnav \vee hsens \rightarrow hpow \quad (9)$$

L'action ins doit rester dans W lorsque l'action mv s'exécute (1). *Si mv échoue ou est annulé, ins est annulé*

Modèle	États	Succ.	Résultat	Temps(s)
Exemple 2	112	13.7	oui	66
Exemple 2-v2	112	13.7	non	0.4
Valve Controller	209	15.5	non	0.4
Valve Driver	51	22.3	oui	56

TABLE 1 – Temps de calcul pour vérifier l'estimabilité. La colonne « États »indique le nombre d'états du système, « Succ. »le nombre moyen de transitions pour chaque état, « Résultat »indique si le système est estimable, « Temps »le temps de calcul en secondes.

(2). *ins commence au moment où mv finit* (3), (4). *Un retard dans mv (resp. ins) peut s'expliquer par un problème de navigation (resp. capteur), d'alimentation, ou une perturbation* (5). *(resp.* (7)*). Une défaillance dans mv ou ins ne peut s'expliquer que par un problème d'alimentation* (6), (8). *Un problème d'alimentation électrique se propage au capteur et à la navigation* (9).

Les variables mv et ins sont observables, c'est-à-dire que l'état de chaque action est connu. Les variables hnav, hsens, hpow et pert sont estimées. L'ensemble d'états S est l'ensemble des évaluations pour toutes les variables, la fonction obs limite une évaluation aux variables mv et ins. Par exemple, l'état initial $(mv = W, ins = W, hnav, hsens, hpow, \overline{pert})$ *produit l'observation* $(mv = W, ins = W)$.

Notre algorithme est implémenté en Scala, et exécuté sur un processeur Intel® Xeon(R) W-2123, 3.60GHz 8 coeurs, avec une mémoire limitée à 4 Go. Le système tel que décrit dans l'exemple 2 est estimable, nous avons donc introduit quelques modifications pour le rendre non estimable. Nous avons fait en sorte que les actions produisent la même observation dans leurs états « Ao » et « Ad ». Nous avons aussi modélisé les systèmes d'exemple de [13] (Valve Controller) et [16] (Valve Driver).

Les résultats présentés dans la table 1 montrent que les systèmes non estimables sont détectés très rapidement. Ceci est dû à la vérification préliminaire introduite dans la Section 6.3 qui capte les impasses plus tôt. Bien que les propositions 1 et 2 ne suffisent pas à assurer l'estimabilité, elles sont utiles dans une grande majorité des cas. Dans le cas des systèmes estimables, le temps de calcul est lié au nombre de fonctions d'estimation partielle testées. Notons qu'il peut être améliorer en tenant compte des besoins opérationnels pour limiter l'espace des fonctions d'estimation à examiner.

8 Conclusion

Cet article motive et définit l'*estimabilité à état unique* pour les systèmes d'événements discrets partiellement observables. Cette propriété indique s'il est possible de suivre l'exécution d'un système en ne gardant en mémoire qu'un unique état, sans jamais perdre de cohérence avec sa dynamique. Certaines conditions sont fournies avec un algorithme pour vérifier cette propriété. Les résultats expérimentaux montrent que cet algorithme peut être appliqué à des exemples tels que des plans de robots autonomes. L'algorithme constitu une validation de concept et pourrait être amélioré de nombreuses façons, par exemple en définissant des heuristiques spécifiques pour le rendre plus efficace. Des tests sur des jeux de données plus larges viendront étoffer les expérimentations.

Dans cet article, l'*estimabilité à état unique* est obtenue en trouvant une fonction d'estimation dont le langage des observations correspond à celui du système. En général, comme il peut exister un nombre important de telles fonctions, nous pourrions chercher la meilleure fonction d'estimation par rapport à d'autres propriétés, par exemple la correction de l'estimation pour certaines variables ou à certains moments.

Les travaux sur l'observabilité et la diagnostibilité tiennent souvent compte d'un délai limité possible entre les événements, leur observation ou leur diagnostic. Si nous pouvions tenir compte des retards dans le processus d'estimation, nous pourrions nous pencher sur une définition plus générale de l'*estimabilité à état unique*. L'étude de la complexité théorique du problème de l'estimabilité à état unique fait aussi partie de nos perspectives.

De plus, nous sommes convaincus qu'il existe un lien très fort entre certains travaux issus de la théorie des jeux [4], ainsi que de la synthèse de contrôleur [11] ce qui laisse entrevoir des perspectives d'application et de comparaison avec ces différents domaines.

Enfin, la synthèse automatique ou semi-automatique des fonctions d'estimation est une application directe de ce travail, par exemple en représentant la fonction d'estimation avec des langages compacts comme le font les auteurs dans [1].

Références

[1] Bouziat, V., X. Pucel, S. Roussel et L. Travé-Massuyès: *Preferential discrete model-based diagnosis for intermittent and permanent faults*. Dans *Proceedings of the 29th International Workshop on Principles of Diagnosis DX'18*, Warsow, Poland, Août 2018. CEUR Workshops Proceedings. https://hal.archives-ouvertes.fr/hal-01796310.

[2] Cassandras, C.G. et S. Lafortune: *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[3] Contant, O., S. Lafortune et D.Teneketzis: *Diagnosis of intermittent faults*. Discrete Event Dynamic Systems, 14(2) :171–202, 2004.

[4] Doyen, L. et J.F. Raskin: *Games with Imperfect Information : Theory and Algorithms*. Dans *Lectures in Game Theory for Computer Scientists*, page 185–212, Cambridge, 2011.

[5] Grastien, A., R. Anbulagan, J. Rintanen et E. Kelareva: *Diagnosis of discrete-event systems using satisfiability algorithms*. Dans *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, tome 22, page 305, Vancouver, British Columbia, 2007.

[6] Grastien, A., M.O. Cordier et C. Largouët: *Incremental diagnosis of discrete-event systems*. Dans *Proceedings of the 29th International Workshop on Principles of Diagnosis DX'05*, Pacific Grove, CA, USA, 2005.

[7] Kleer, J. De: *Diagnosing multiple persistent and intermittent faults*. Dans *Proceedings of the 21th International Joint Conference on Artificial Intelligence IJCAI'19*, Pasadena, CA, USA, Juin 2009.

[8] Kurien, J. et P. Nayak: *Back to the future for consistency-based trajectory tracking*. Dans *Proceedings of the 17th AAAI Conference on Artificial Intelligence*, pages 370–377, Austin, Texas, USA, 2000.

[9] Moore, E.F.: *Gedanken-Experiments on Sequential Machines*. Dans Shannon, Claude et John McCarthy (rédacteurs) : *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956.

[10] Ozveren, C. M. et A. S. Willsky: *Observability of discrete event dynamic systems*. IEEE Transactions on Automatic Control, 35(7) :797–806, Juillet 1990.

[11] Pralet, C., G. Verfaillie, M. Lemaître et G. Infantes: *Constraint-Based Controller Synthesis in Non-Deterministic and Partially Observable Domains*. Dans *Proceedings of the 2010 Conference on ECAI 2010 : 19th European Conference on Artificial Intelligence*, pages 681–686, Amsterdam, The Netherlands, 2010.

[12] Ramadge, P.J.: *Observability of discrete event systems*. Dans *Proceedings of the 25th IEEE Conference on Decision and Control CDC'86*, pages 1108–1112, Athens, Greece, 1986.

[13] Sampath, M., R. Sengupta, S. Lafortune, K. Sinnamohideen et D.Teneketzis: *Diagnosability of discrete-event systems*. IEEE Transactions on automatic control, 40(9) :1555–1575, 1995.

[14] Schumann, A., Y. Pencolé *et al.*: *Scalable Diagnosability Checking of Event-Driven Systems.* Dans *Proceedings of the 20th International Joint Conference on Artificial Intelligence IJCAI'00*, tome 7, pages 575–580, Hyderabad, India, 2007.

[15] V.Verma, T.Estlin, A.Jónsson, C. Pasareanu, R. Simmons et K. Tso: *Plan execution interchange language (PLEXIL) for executable plans and command sequences*. Dans *Proceedings of the 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space iSAIRAS'05*, Munich, Germany, 2005.

[16] Williams, B.C. et P. Nayak: *A Model-based Approach to Reactive Self-Configuring Systems*. Dans *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, Portland, Oregon, février 1996.

[17] Wonham, W.M., K. Cai et K. Rudie: *Supervisory control of discrete-event systems : A brief history*. Annual Reviews in Control, 45 :250–256, 2018.

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