
Diagnostic de systèmes à événements discrets à base de cohérence par SAT

Alban Grastien* — Anbu Anbulagan**

*NICTA et The Australian National University
Canberra Research Laboratory, Australia
alban.grastien@nicta.com.au

**Green Intelligent Software Solutions Pty Ltd.
Canberra, Australia
anbulagan@gissio.com.au

RÉSUMÉ. Nous présentons une nouvelle technique pour le diagnostic de systèmes à événements discrets. Cette technique se base sur la notion de cohérence : le système de diagnostic renvoie une hypothèse de diagnostic si celle-ci est considérée comme cohérente avec le modèle et les observations. Notre algorithme se découpe en deux niveaux : le premier niveau transforme le problème de diagnostic en une séquence de problèmes simples appelés questions de diagnostic ; le second niveau résout les questions de diagnostic à l'aide de la satisfiabilité propositionnelle (SAT). Ce découpage permet d'expliquer les observations et de produire un diagnostic du système. Nos expérimentations montrent que les algorithmes utilisant SAT sont capables de résoudre des problèmes de diagnostic que nous ne pouvions pas résoudre avec d'autres techniques.

ABSTRACT. We present a new, two-level-based, technique for the diagnosis of discrete-event systems. The first level transforms the diagnosis problem in a sequence of diagnosis questions. The second level answers the diagnosis questions. We propose to implement this second level with a SAT solver. This two-level algorithm allows to explain the observations and to generate a diagnosis of the system. Our experiments show that the SAT-based approach can solve problems that we could not solve with other techniques.

MOTS-CLÉS : diagnostic à base de modèle, systèmes à événements discrets, SAT

KEYWORDS: model-based diagnosis, discrete-event systems, SAT

1. Introduction

Le *diagnostic* est le problème consistant à détecter sur un système l'occurrence de comportements défectueux et, si ceux-ci sont pressentis, les identifier (*quelles fautes ?*) et les isoler (*dans quelle partie du système ?*) (Hamscher et coll., 1992). Diagnostiquer ces comportements permet ensuite d'intervenir sur le système pour réparer la panne (Sachenbacher et coll., 2005; Torta et coll., 2008; Cordier et coll., 2008) ; si c'est impossible, on peut vouloir minimiser son impact tant pour l'utilisateur (WS-Diamond Team, 2007) (qu'il s'agisse de sécurité ou de qualité de service) que sur le système lui-même, notamment pour éviter les *effets de cascade* (Mayer et coll., 2009). Une autre application du diagnostic est tout simplement de comprendre ce qui s'est passé sur le système. Nous nous intéressons ici au diagnostic à base de modèle qui raisonne à partir d'une description du système.

Les systèmes à événements discrets finis (SÉD (Cassandras et coll., 1999)) représentent la classe de systèmes dynamiques dont les états et les transitions sont modélisés de manière discrète, par exemple par un automate à états finis. Tout comportement du système est alors représenté par un *chemin* sur cet automate. Ce simple formalisme permet de représenter de manière relativement précise nombre de systèmes. Depuis une quinzaine d'années et les premiers travaux sur ce domaine (Sampath et coll., 1995), de nombreuses recherches, citées tout au long de cet article, ont été effectuées dans le domaine du diagnostic de SÉD. Le problème de diagnostic est généralement résolu en calculant tous les chemins sur le modèle compatibles avec les observations, puis en extrayant l'information de diagnostic de ces chemins (les chemins sont-ils *nominaux* ou *anormaux* ?, etc.). Ces approches sont souvent très gourmandes en ressources (temps et mémoire) malgré le niveau d'abstraction des SÉD, et si les différentes techniques développées pour faire face à ce défaut permettent le diagnostic de systèmes d'un ordre de grandeur plus important, elles restent cependant limitées.

Comme présenté précédemment, le diagnostic de SÉD peut souvent se voir comme la recherche de chemins spécifiques dans un modèle. Une technique utilisée avec succès pour résoudre des problèmes semblables en planification classique (Kautz et coll., 1996), en vérification de modèle (*model-checking*) (Biere et coll., 1999) ou en diagnosticabilité (Rintanen et coll., 2007) consiste à traduire ces problèmes en problèmes de satisfiabilité propositionnelle (SAT) et à utiliser des solveurs SAT. La recherche de chemin est transformée en une formule propositionnelle satisfiable si et seulement si un chemin existe ; le solveur SAT vérifie que la formule est effectivement satisfiable et, si c'est le cas, renvoie une affectation possible des variables dont on peut trivialement extraire une solution. Nous proposons d'utiliser ce type de technique pour le diagnostic de SÉD. Plutôt que de calculer tous les chemins, l'algorithme fonctionne sur deux niveaux.

1) Le premier niveau transforme le problème de diagnostic en une séquence de problèmes simples, que nous appelons *questions de diagnostic*. Chaque question a la forme suivante : *Le modèle du système autorise-t-il un comportement compatible avec les observations qui satisfasse une certaine propriété ?* Cette question est donc un

problème de recherche de chemin attendant la réponse *oui* ou *non*. Ce premier niveau pilote le second niveau et choisit la prochaine question en fonction des réponses.

2) Le second niveau répond à chaque question de diagnostic. Les questions étant toutes de type fermé, la réponse est soit *oui* soit *non*. Si la réponse est positive, le second niveau renvoie également la preuve de la réponse (à savoir, le chemin recherché). Dans ce papier, ce second niveau est résolu grâce à un solveur SAT.

Ce découpage en deux niveaux permet de résoudre la plupart des problèmes de diagnostic. L'article présente une implémentation complète de ce système. Les tests expérimentaux conduits au cours de cette étude montrent une supériorité incontestable des solveurs SAT sur les techniques existantes. Cette approche généralise nos précédents travaux (Grastien et coll., 2007a; Grastien et coll., 2007b). L'article se compose comme suit. La prochaine section présente le diagnostic de SÉD. Notre approche est développée en section 3. Le principe d'utilisation de SAT pour résoudre une question de diagnostic est présenté en section 4. La modélisation est donnée en section 5. Une validation expérimentale est présentée en section 6. Des pistes de développement futurs sont données en section 7.

2. Diagnostic de systèmes à événements discrets

2.1. Diagnostic

Que ce soit le fait d'usure, d'imperfections, d'utilisations inadéquates ou d'événements incontrôlables, tout système peut dévier de son comportement nominal et entrer dans un comportement *fautif*. Ces déviations conduisent généralement à une baisse de la qualité de service du système et peuvent endommager le système (effet de cascade) voire mettre en danger des opérateurs humains. Il est donc nécessaire de surveiller le système pour intervenir à temps. Le *diagnostic* est le problème consistant à détecter l'occurrence de comportements fautifs dans un système. Si de tels comportements se produisent, le *diagnostiqueur* doit déterminer précisément quel type de panne s'est produit (identification) et sur quelle partie du système (localisation).

Différentes techniques peuvent s'appliquer pour le diagnostic. Les méthodes les plus satisfaisantes, que ce soit pour la complétude de l'approche ou la réutilisation des outils développés, sont basées sur un *modèle* de comportement du système (Reiter, 1987; Hamscher et coll., 1992). Le modèle est utilisé pour *expliquer* les observations effectuées sur le système. Il existe de nombreux formalismes pour modéliser les systèmes ; dans cet article, nous nous intéressons aux systèmes à événements discrets.

2.2. Systèmes à événements discrets

Un système à événements discrets fini (SÉD) (Cassandras et coll., 1999) est un système dynamique dont l'état peut être modélisé par un ensemble fini de variables

prenant des valeurs parmi un ensemble fini, et dont l'évolution est discrète. Plus précisément, nous considérons que le comportement du système peut être représenté par un langage régulier ou un automate. Ce *modèle* peut être utilisé pour raisonner sur le comportement possible du système, que ce soit pour vérifier ses propriétés (*model-checking*), pour choisir comment l'utiliser (planification, contrôle), ou pour surveiller son comportement (diagnostic).

Il est peu efficace et encore moins pratique de représenter explicitement le modèle du système par un seul automate. Il est possible d'utiliser une représentation *décentralisée* ; dans ce cas, le système est partitionné en *composants* qui sont modélisés chacun séparément ; le modèle global peut être retrouvé en synchronisant chaque modèle local ; l'avantage de cette approche est une décomposition naturelle du système facilitant la modélisation et le découpage en sous-systèmes. Une autre modélisation possible est la représentation *symbolique* ; l'état du système est modélisé par un ensemble de variables discrètes ; les transitions entre états ne sont pas énumérées mais décrites sous forme de règles de transitions ; l'avantage de cette approche est qu'elle autorise le raisonnement par techniques symboliques. Pour ce papier, nous introduisons un nouveau modèle combinant les avantages des représentations décentralisée et symbolique. Le système Γ est un ensemble de composants interconnectés. Chaque composant est modélisé de manière symbolique et les connections sont représentées par des événements de synchronisation.

2.2.1. Composant

Définition 1 (Modèle d'un composant) *Un composant c_i est modélisé par un tuple $\langle V_i, E_i, R_i, I_i \rangle$ où*

- V_i est un ensemble de variables d'état ; chaque variable $v \in V_i$ prend ses valeurs dans un domaine discret fini $d(v)$;
- E_i est un ensemble d'événements partitionné en événements exogènes E_i^{exo} , événements d'entrée E_i^{in} , événements observables E_i^{obs} , et événements de sortie E_i^{out} ;
- R_i est un ensemble de règles ;
- I_i est l'affectation totale initiale des variables V_i .

Une règle $r \in R_i$ est un tuple $\langle pre_r, e_r^I, E_r^O, eff_r \rangle$ où pre_r est une formule propositionnelle sur les variables de V_i , $e_r^I \in E_i^{exo} \cup E_i^{in}$ est l'événement déclencheur de la règle, $E_r^O \subseteq E_i^{obs} \cup E_i^{out}$ est l'ensemble (éventuellement vide) d'événements générés par le tirage de la règle, et eff_r modélise les effets (éventuellement nuls) de la règle par un ensemble d'affectations sur les variables V_i .

L'état q^i (ou simplement q) d'un composant c_i est défini comme une affectation totale des variables d'état du composant : $q \in \prod_{v \in V_i} d(v)$. On note $v(q) \in d(v)$ la valeur associée à v dans l'état q . L'évolution d'un composant est modélisée par le tirage d'une séquence de règles autorisées dans l'état courant et affectant l'état du composant.

Définition 2 (Chemin d'un composant) *Un chemin du composant c_i modélisé par $\langle V_i, E_i, R_i, I_i \rangle$ est une séquence $q_0 \xrightarrow{e_1^I | E_1^O} q_1 \xrightarrow{e_2^I | E_2^O} \dots \xrightarrow{e_n^I | E_n^O} q_n$ telle qu'il existe une séquence de règles r_1, \dots, r_n où :*

- $\forall j \in \{1, \dots, n\}, r_j = \langle pre_{r_j}, e_{r_j}^I, E_{r_j}^O, eff_{r_j} \rangle$,
- $\forall j \in \{1, \dots, n\}, q_{j-1} \models pre_{r_j}$,
- $\forall j \in \{1, \dots, n\}, e_j^I = e_{r_j}^I$ et $E_j^O = E_{r_j}^O$, et
- $\forall j \in \{1, \dots, n\}, \forall v \in V_i$, si $v(eff_{r_j})$ est défini alors $v(q_j) = v(eff_{r_j})$, sinon $v(q_j) = v(q_{j-1})$.

Le modèle du composant c_i représente donc implicitement un automate dont l'ensemble des nœuds est l'ensemble des états du composant, et les transitions relient q et q' s'il existe une règle $r = \langle pre_r, e_r^I, E_r^O, eff_r \rangle$ telle que $q \xrightarrow{e_r^I | E_r^O} q'$ est un chemin possible du système. Si on considère un comportement commençant dans l'état initial, alors le premier état q_0 du chemin doit être l'affectation initiale I_i .

Exemple 3 *Les figures 1a et 1b donnent le modèle du composant i utilisé par la suite dans les expérimentations ; le modèle de chaque composant est identique. Ce composant représente un serveur fictif dans un réseau. L'état de fonctionnement normal est \emptyset . Quand une panne (événement f_i) a lieu sur le composant (transition \emptyset à F), le composant émet un message $red_{i \rightarrow j}$ pour chacun de ses voisins j , leur demandant de redémarrer ; cette transition comporte donc autant d'événements de sortie que de voisins pour le composant. L'événement asy_i modélise l'effet retardé (asynchrone) d'un événement. Les deux événements observables se remarquent par une majuscule sur le nom de l'événement : Red_i correspond au redémarrage du composant, $Retour_i$ correspond à la fin du redémarrage. Finalement, l'événement $red_{i \leftarrow j}$ correspond à la réception d'un message de la part d'un des voisins du composant ; il y a donc autant de transitions entre \emptyset et W que de voisins pour le composant.*

Chaque composant a exactement quatre voisins. Les états du composant sont modélisés par quatre variables booléennes : f représente si une faute vient d'avoir lieu sur le composant ; w indique qu'un voisin a forcé le redémarrage du composant ; a modélise le caractère asynchrone du redémarrage du composant ; r indique que le composant a fini de redémarrer. L'affectation correspondant à chaque état est donnée sur la figure 1c. Par exemple, les transitions menant à \emptyset sont modélisées par la règle suivante : $\langle r = \top, asy_i, \{Retour_i\}, \{r \rightarrow \perp, f \rightarrow \perp\} \rangle$.

2.2.2. Système

Un système est un ensemble de composants interconnectés. Chaque composant suit son propre chemin, mais les connexions entre composants restreignent leurs comportements et les obligent à se synchroniser.

Définition 4 (Modèle du système) *Le modèle du système est un tuple $\langle C, M, S \rangle$ où*

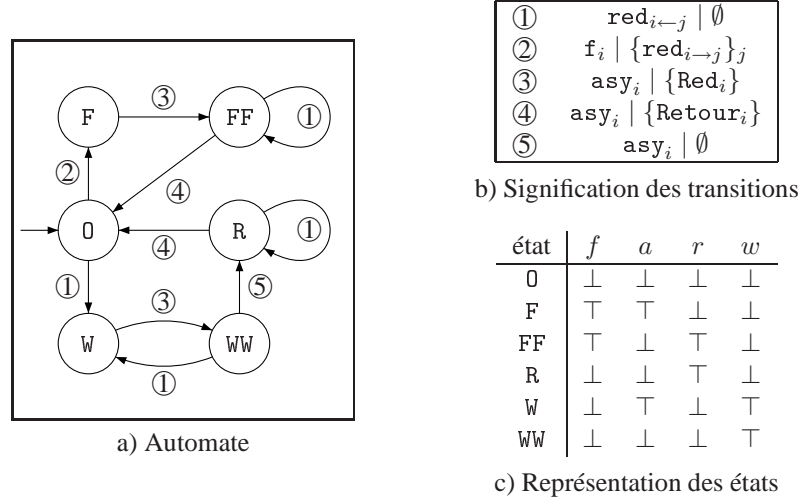


Figure 1. Exemple de composant

- C est un ensemble de m composants $\{c_1, \dots, c_m\}$,
- M est une table qui associe chaque composant c_i avec son modèle $\langle V_i, E_i, R_i, I_i \rangle$ tel que $i \neq j \Rightarrow E_i \cap E_j = \emptyset$,
- $S \subseteq \bigcup_{c_i \in C} E_i^{out} \times \bigcup_{c_j \in C} E_j^{in}$ est un ensemble de paires $\langle e_i^O, e_j^I \rangle$ où e_i^O est un événement de sortie du composant c_i et e_j^I est un événement d'entrée du composant $c_j \neq c_i$ tel que $\{\langle e', e \rangle, \langle e'', e \rangle\} \subseteq S \Rightarrow e' = e''$ et $\{\langle e, e' \rangle, \langle e, e'' \rangle\} \subseteq S \Rightarrow e' = e''$.

On note $E^{exo} = \bigcup_i E_i^{exo}$, $E^{in} = \bigcup_i E_i^{in}$, $E^{obs} = \bigcup_i E_i^{obs}$ et $E^{out} = \bigcup_i E_i^{out}$. La synchronisation $\langle e_i^O, e_j^I \rangle \in S$ indique que l'événement de sortie e_i^O doit avoir lieu au même moment que e_j^I .

L'état q du système est un tuple d'états de chaque composant : $q = \langle q^1, \dots, q^m \rangle$. L'évolution du système est représentée par l'évolution de chaque composant synchronisée par S . Nous donnons donc une définition étendue des chemins d'un composant qui autorise des transitions ne comportant pas d'événement.

Définition 5 (Chemin étendu d'un composant) L'ensemble étendu des règles du composant c_i modélisé par $\langle V_i, E_i, R_i, I_i \rangle$ est défini par $R_i^* = R_i \cup \{\langle \text{vrai}, \varepsilon_i, \emptyset, \{\} \rangle\}$ où $\varepsilon_i \notin E_i$.

Un chemin étendu traj du composant c_i modélisé par $\langle V_i, E_i, R_i, I_i \rangle$ est défini comme un chemin du composant fictif c_i^* modélisé par $\langle V_i, E_i \cup \{\varepsilon_i\}, R_i^*, I_i \rangle$.

Définition 6 (Transitions synchronisées) Soit $\langle C, M, S \rangle$ un système et soit l'ensemble de transitions $t = \{t_1, \dots, t_m\}$ tel que pour tout $i \in \{1, \dots, m\}$, $t_i =$

$q_i \xrightarrow{e_i^I | E_i^O} q'_i$ est un chemin étendu de c_i . L'ensemble de transition t est synchronisé si il existe $i \in \{1, \dots, m\}$ tel que :

- 1) $e_i^I \in E_i^{exo}$,
- 2) $\forall j \neq i, e_j^I \neq \varepsilon_j \Rightarrow e_j^I \in E_j^I \wedge \exists k \in \{1, \dots, m\} : \exists e \in E_k^O : \langle e, e_j^I \rangle \in S$ et
- 3) $\forall j \in \{1, \dots, m\}, \forall e \in E_j^O, \forall e', \langle e, e' \rangle \in S, \forall k, e' \in E_k^I \Rightarrow e' = e_k^I$.

La synchronisation de t est le chemin $q \xrightarrow{E^I | E^O} q'$ défini ainsi :

- $q = \langle q_1, \dots, q_m \rangle$,
- $q' = \langle q'_1, \dots, q'_m \rangle$,
- $E^I = \bigcup_{e_i^I \neq \varepsilon_i} \{e_i^I\}$ et
- $E^O = \bigcup_{i \in \{1, \dots, m\}} \{e_i^O\}$.

La première propriété s'assure qu'un composant génère un événement exogène (le composant c_i). La deuxième propriété s'assure que si une transition est tirée sur le composant c_j ($j \neq i$), alors elle est déclenchée par un événement d'entrée synchronisé avec l'événement de sortie d'un autre composant. La troisième propriété s'assure au contraire que tout événement de sortie génère les événements d'entrée décrits dans S . Par la suite, nous parlons de *macro-transition* pour représenter l'occurrence simultanée de ces transitions.

Définition 7 (Chemin d'un système) Un chemin du système $\langle C, M, S \rangle$ est une séquence $q_0 \xrightarrow{E_1^I | E_1^O} \dots \xrightarrow{E_n^I | E_n^O} q_n$ telle que $\forall i \in \{1, \dots, m\}, \exists q_0^i \xrightarrow{e_1^{iI} | E_1^{iO}} \dots \xrightarrow{e_n^{iI} | E_n^{iO}} q_n^i$ un chemin étendu de c_i tel que chaque chemin $q_{j-1} \xrightarrow{E_j^I | E_j^O} q_j$ est la synchronisation de l'ensemble de chemins $\{q_{j-1}^i \xrightarrow{e_j^{iI} | E_j^{iO}} q_j^i\}$.

Exemple 8 Le système que nous utilisons dans les expérimentations comporte 20 composants. Les synchronisations s'opèrent sur les événements suivants : $\langle red_{i \rightarrow j}, red_{j \leftarrow i} \rangle \in S$ pour toute paire de composants i, j voisins. Les composants sont disposés sous forme de tore ; chaque composant est défini par deux entiers $x \in \{0, \dots, 3\}$ et $y \in \{0, \dots, 4\}$, tel que deux composants $\langle x_1, y_1 \rangle$ et $\langle x_2, y_2 \rangle$ sont voisins si leurs valeurs x et y varient au plus de une unité, formellement :

$$(x_1 = x_2) \wedge ((y_1 + 1 = y_2 \pmod{5}) \vee (y_1 - 1 = y_2 \pmod{5})) \\ \vee \\ (y_1 = y_2) \wedge ((x_1 + 1 = x_2 \pmod{4}) \vee (x_1 - 1 = x_2 \pmod{4}))$$

La topologie est présentée figure 2 ; sur la figure, les composants sont représentés par des rectangles et les connexions par des lignes entre composants ; un composant et ses quatre voisins sont présentés en gris. Le modèle complet du système est disponible à cette adresse : <http://www.grastien.net/ban/data/ria10/>.

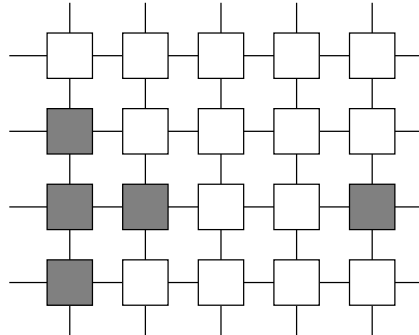


Figure 2. *Topologie du système*

2.3. Le problème

Un comportement sur le système génère des observations. La séquence des observations générée par un chemin est exactement la séquence des ensembles d'événements observables qui ont lieu au cours du chemin.

Définition 9 (Observations)

Les observations du chemin $traj = q_0 \xrightarrow{E_1^I | E_1^O} \dots \xrightarrow{E_n^I | E_n^O} q_n$ sont la séquence $obs(traj) = (E_1^O \cap E^{obs}, \dots, E_n^O \cap E^{obs})$.

Dans cette définition, une transition qui ne comporte aucun événement observable est tout de même observée puisqu'elle génère un ensemble vide d'observations. Ici, les observations correspondent à l'occurrence de certains événements particuliers. On peut considérer des observations sur des variables d'état ; cependant, se limiter à des observations sur les événements n'est pas restrictif car il est trivial de passer d'une notation à une autre ; ainsi si l'état q est censé générer l'observation o , il est possible de considérer o comme un événement observable et d'ajouter o à chaque transition entrant dans l'état q . Il arrive également que la séquence d'observations reçue soit différente de la séquence effectivement émise. Des exemples sont donnés par la suite.

Lorsque le système suit un comportement, il génère des observations qui permettent de s'assurer de la bonne marche du système. Les observations ne fournissent cependant qu'une vue partielle du comportement, et c'est le rôle du diagnostic d'interpréter ces observations.

Définition 10 (Problème de diagnostic)

Un problème de diagnostic est un tuple $\langle \langle C, M, S \rangle, obs \rangle$ où $\langle C, M, S \rangle$ est le modèle d'un système et obs représente les observations d'un chemin.

Étant donné un comportement *traj* inconnu sur le système $\langle C, M, S \rangle$ générant des observations $obs(traj)$, le *diagnostiqueur* doit déterminer ce qui s'est passé. En réalité, la personne en charge de la surveillance a des requêtes plus précises comme : *Des pannes ont-elles eu lieu ? Quel est l'état du système à l'issue des observations ? etc.* Une faute ou panne est traditionnellement définie comme étant un événement particulier sur le système, bien qu'on puisse la définir comme un ensemble d'états particulier ou un enchaînement spécifique d'événements (Jéron et coll., 2006). La compétition de diagnostic (Kurtoglu et coll., 2009) demande l'ensemble des pannes du système. Savoir ce qu'on cherche à diagnostiquer (Sachenbacher et coll., 2005; Cordier et coll., 2008; Torta et coll., 2008) permet de focaliser l'attention du diagnostiqueur. La définition de diagnostic reste donc ici relativement floue. L'implémentation précise de l'algorithme de diagnostic varie selon quel résultat est effectivement demandé.

2.4. Techniques existantes

La technique principalement utilisée pour le diagnostic de SÉD est le dépliage du modèle (Lamperti et coll., 2003). L'automate du modèle est exploré pour obtenir les chemins sur le modèle compatibles avec les observations. Une fois cette liste des chemins calculée, il est relativement facile d'extraire les informations pertinentes pour le diagnostic. La grande faiblesse de cette technique est l'explosion combinatoire associée : la taille de l'automate de modèle est exponentielle par rapport au nombre de composants du système, ce qui rend sa construction impossible même pour un nombre relativement faible de composants (à peine quelques dizaines). Même sans génération explicite du modèle, l'exploration de celui-ci est impossible pour des systèmes de taille réaliste.

Pour permettre le diagnostic de systèmes de taille réaliste, plusieurs approches ont été développées. Le système peut être compilé pour réduire la complexité de la recherche : voir par exemple le diagnostiqueur de Sampath (Sampath et coll., 1995; Rozé et coll., 2002). La compilation est effectuée hors-ligne, mais elle a une taille exponentielle par rapport au modèle d'origine (Rintanen, 2007), ce qui la rend inutilisable en pratique même avec des techniques de compilation (Marchand et coll., 2002; Schumann et coll., 2004). Les récents travaux se sont concentrés sur des approches décentralisées, distribuées ou symboliques (Benveniste et coll., 2003; Wang et coll., 2004; Pencolé et coll., 2005; Su et coll., 2005; Pencolé et coll., 2006; Schumann et coll., 2007; Kan John et coll., 2008; Kan John et coll., 2010). Ces techniques sont parfois très efficaces et permettent de considérer des systèmes de taille bien supérieure aux méthodes naïves. Cependant, dans de nombreux cas elles doivent sacrifier la précision du calcul pour l'efficacité ou sont incapables de terminer le calcul en temps raisonnable.

3. Approche proposée

Notre approche généralise et complète les résultats de (Grastien et coll., 2007a; Grastien et coll., 2007b).

Le but du diagnostic est souvent d'extraire certaines informations relativement simples comme l'occurrence de certains événements de faute ou l'état actuel du système. Les algorithmes actuels calculent souvent tous les chemins possibles sur le modèle conformément aux observations reçues, avant d'extraire le diagnostic. Nous proposons une approche diamétralement opposée où nous choisissons de tester la compatibilité des observations avec diverses hypothèses.

Pour cela, nous proposons de transformer le problème de diagnostic en problèmes plus simples à résoudre et pour lesquels il est suffisant de trouver un seul chemin. Nous appelons ces problèmes des *questions de diagnostic*.

Définition 11 (Question de diagnostic)

Une question de diagnostic Q est un tuple $\langle \langle C, M, S \rangle, obs, \mathcal{C} \rangle$ où $\langle C, M, S \rangle$ est le modèle d'un système, obs est les observations d'un chemin et \mathcal{C} est un ensemble de chemins.

Ce problème attend une réponse booléenne : *oui* s'il existe un chemin $traj$ de $\langle C, M, S \rangle$ tel que $obs(traj) = obs$ et $traj \in \mathcal{C}$, *non* sinon. D'autre part, si la réponse est *oui*, la réponse à la question de diagnostic doit renvoyer le chemin $traj$.

Notre algorithme de diagnostic est basé sur un découpage du problème en questions. Il y a donc deux niveaux pour résoudre le problème de diagnostic.

1) Le premier niveau choisit les questions de diagnostic. Selon les réponses fournies à ces questions, le premier niveau peut poser de nouvelles questions.

2) Le second niveau a en charge de répondre à chaque question.

Par exemple, si le problème de diagnostic consiste à déterminer si une faute particulière f a eu lieu, le premier algorithme va poser deux questions de diagnostic où :

- a) \mathcal{C} est l'ensemble des trajectoires qui ne contiennent pas l'événement f , et
- b) \mathcal{C} est l'ensemble des trajectoires qui contiennent l'événement f .

Si la réponse à la première question est *non*, la deuxième question attend certainement la réponse *oui* et ne nécessite donc pas d'être posée (sauf si un chemin illustrant la réponse est requis). Si une seule réponse est positive, le diagnostic est précis. Finalement, si les deux réponses sont positives, l'occurrence de la faute n'est ni confirmée ni infirmée parce que les observations ne sont pas suffisantes pour désambigüiser le diagnostic.

Considérons à présent que l'on cherche à déterminer le chemin le plus vraisemblable compatible avec les observations, défini comme le chemin comportant le moins d'événements de faute. Il est alors possible de poser la question *Existe-il un chemin*

sur le modèle du système qui soit compatible avec les observations et qui comporte k occurrences d'événements de faute ? pour des valeurs de k de plus en plus grandes.

Le lien avec l'algorithme *diagnose* de (Reiter, 1987) est évident. Dans cet algorithme, le diagnostiqueur dispose d'un certain nombre de diagnostics candidats et teste successivement s'ils sont compatibles avec les observations. Chaque test correspond à une question de diagnostic. Si le diagnostic testé n'est pas compatible avec les observations, la liste des diagnostics candidats est mise à jour grâce au *conflict* renvoyé par le solveur. Dans notre algorithme, le solveur de second niveau ne renvoie pas de conflit mais simplement une réponse négative. Il s'agit là d'une amélioration que nous souhaiterions introduire à l'avenir : que l'algorithme de second niveau renvoie un rapport expliquant pourquoi la réponse est négative.

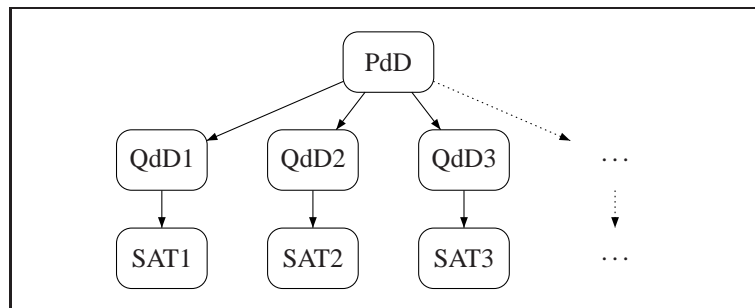


Figure 3. Principe de l'algorithme

Se concentrer sur des questions de diagnostic permet de simplifier grandement le problème. On peut alléguer que les méthodes basées sur le calcul de toutes les trajectoires sont plus ou moins condamnées dans un futur proche. En effet, la complexification des systèmes rend leurs comportements de plus en plus complexes. Il devient impossible d'abstraire le modèle de manière réellement efficace tout en conservant une précision maximale. Par ailleurs, il est possible d'étudier le modèle du système pour déterminer quels capteurs doivent être ajoutés pour pouvoir diagnostiquer précisément l'occurrence de fautes (Sampath et coll., 1995; Brandán Briones et coll., 2008). En revanche, à cause du coût que cela représente, il est moins justifiable d'ajouter des capteurs pour simplifier un problème de diagnostic qui peut être résolu autrement. Une méthode basée sur des questions de diagnostic clairement définies permet d'employer des techniques jusqu'ici inapplicables, en particulier SAT. Le principe de l'algorithme est illustré sur la figure 3. Le problème de diagnostic (PdD) est transformé en un certain nombre de questions de diagnostic (QdD). Chaque question est à son tour traduite en un problème SAT.

Une approche similaire a été développée depuis nos premiers travaux sur le sujet pour les systèmes linéaires (Heintz et coll., 2008). Dans ces travaux, un certain nombre de *tests* peuvent être effectués pour déterminer précisément l'occurrence de

fautes. Un test n'est cependant déclenché que lorsque le système de diagnostic juge qu'il permettra d'améliorer le diagnostic.

4. Résoudre les questions de diagnostic avec SAT

Nous présentons SAT et comment nous comptons résoudre une question de diagnostic avec SAT. Les modélisations précises sont données dans la section suivante.

4.1. SAT

Étant donné un ensemble de variables booléennes \mathcal{V} , étant donnée une formule propositionnelle Φ définie sur \mathcal{V} , le problème de satisfiabilité (SAT) consiste à déterminer s'il existe une affectation $ass : \mathcal{V} \rightarrow \{\top, \perp\}$ qui rende la formule Φ logiquement vraie, et à rendre une affectation si elle existe. La formule est généralement présentée sous forme normale conjonctive (CNF), c'est-à-dire comme la conjonction de *clauses*, chaque clause étant une disjonction de littéraux et de leurs négations. Une clause définit une *contrainte* sur l'ensemble des affectations satisfiables du problème SAT.

Dans les deux dernières décennies, SAT a été fortement étudié au sein de la communauté *intelligence artificielle*. Ces travaux ont conduit au développement rapide d'un grand nombre d'algorithmes et de *solveurs* SAT efficaces, basés sur des techniques de recherche systématique ou au contraire stochastique dont les performances relatives varient selon qu'on souhaite résoudre un problème généré aléatoirement, un problème synthétique ou un problème industriel. Les solveurs stochastiques sont incapables de démontrer qu'une formule n'est pas satisfiable, et ne sont donc pas suffisants pour le diagnostic par SAT ; nous les ignorons donc par la suite.

Les solveurs systématiques SAT actuels sont basés sur l'algorithme DPLL (Davis et coll., 1962) auquel sont ajoutées des extensions permettant d'obtenir plus de propagations unitaires, d'améliorer le choix de la variable de branchement dans la recherche ou pour rendre le retour-arrière plus intelligent. Parmi ces améliorations, citons le *one-step lookahead* (Li et coll., 1997) servant d'heuristique pour le calcul de la variable de branchement, et l'apprentissage dynamique de clauses couplé à des décisions heuristiques basées sur l'activité des clauses, rendu possible grâce à la technique des *watched literals* (littéraux surveillés (Moskewicz et coll., 2001)). Ces deux améliorations ne peuvent cependant pas être intégrées ensemble. Comme démontré par les résultats récents de la *SAT competition*¹, les problèmes industriels sont plus facilement résolus par des solveurs implémentant l'apprentissage de clauses, tels zChaff (Zhang et coll., 2001), MINISAT (Eén et coll., 2004), RSat (Pipatsrisawat et coll., 2007) et glucose (Audemard et coll., 2009). Les algorithmes de lookahead tels Dew_Satz (Anbulagan et coll., 2005), Kcnfs (une amélioration de cnfs (Dubois et coll., 2001)) et March_dl (Heule et coll., 2006) sont cependant plus performants sur

1. Voir www.satcompetition.org/.

les problèmes aléatoires. Tous ces solveurs sont accessibles depuis le site internet de la *SAT competition*.

4.2. Traduire une question de diagnostic en SAT

Une question de diagnostic cherche à déterminer s'il existe un chemin particulier sur un modèle. Nous proposons de traduire ce problème vers un problème SAT. Cette approche a déjà été utilisée avec certains succès pour la planification (Kautz et coll., 1996), la vérification de modèle (Biere et coll., 1999) ou le test de diagnostiquabilité (Rintanen et coll., 2007). La traduction que nous proposons est telle que chaque chemin est associé à une affectation *ass* différente ; les contraintes sur le chemin recherché sont traduites par des contraintes insérées dans la CNF transmise au solveur SAT.

Une difficulté inhérente à l'utilisation de SAT pour la recherche de chemin est qu'il est nécessaire de connaître le nombre de transitions du chemin recherché. Nous laissons ce point imprécis pour l'instant ; il est étudié et résolu en sous-section 5.2, quand l'instance du problème de diagnostic est mieux définie. À ce niveau, nous considérons que le chemin comporte exactement n transitions.

Pour modéliser un chemin $p = q_0 \xrightarrow{E_1^I|E_1^O} \dots \xrightarrow{E_n^I|E_n^O} q_n$, nous définissons l'ensemble \mathcal{V} de variables booléennes suivantes.

- Pour tout composant $c_i \in C$, pour toute variable d'état $v \in V_i$ du composant c_i , pour toute valeur $\nu \in d(v)$ du domaine de v , pour tout pas de temps $j \in \{0, \dots, n\}$, est définie une variable booléenne notée $(v = \nu)[t = j]$. Cette variable est évaluée à *vrai* ssi la variable v du composant c_i est évaluée à ν dans l'état q_j du chemin.

- Pour tout composant $c_i \in C$, pour tout événement $e \in E_i$ du composant c_i , pour tout pas de temps $j \in \{1, \dots, n\}$, est définie une variable booléenne notée $e[t = j]$. Cette variable est évaluée à *vrai* ssi l'événement e a lieu lors de la j ème transition ($e \in E_j^I \cup E_j^O$) du chemin.

Par la suite, nous définissons d'autres variables qui permettent de simplifier le problème CNF.

Pour chaque chemin de taille n sur le modèle, il existe une et une seule affectation des variables du problème SAT satisfaisant la sémantique des variables ; cette transformation est donc correcte. Une fois l'affectation des variables trouvée, le chemin peut être trivialement reconstruit. Nous cherchons un chemin sur le modèle compatible avec les observations et qui satisfasse une condition spécifique. Le problème SAT est donc défini par la conjonction de trois types de contraintes suivantes :

- 1) L'application doit modéliser un chemin sur le modèle : contraintes *Mod*.
- 2) Le chemin doit être compatible avec les observations : contraintes *Obs*.
- 3) Le chemin doit satisfaire la condition : contraintes *Que*.

Si la réponse à la question de diagnostic est *oui*, alors le problème $Mod \wedge Obs \wedge Que$ est satisfiable et le solveur SAT doit renvoyer une affectation *ass* qui permet de reconstruire le chemin *traj*. Si la réponse à la question de diagnostic est *non*, alors le problème SAT n'est pas satisfiable et le solveur doit le déterminer. Il est donc nécessaire d'avoir un solveur complet, capable de détecter les problèmes insatisfiables.

Bien que nous ayons choisi SAT pour ses performances, nous sommes conscients qu'il existe des limites à cette approche.

1) Tout d'abord, SAT est un problème NP-COMPLET en général (Cook, 1971), tandis que résoudre une question de diagnostic devrait être polynomial en n (le type d'observations et de questions peuvent modifier la complexité). Cependant, les algorithmes SAT sont extrêmement efficaces et les expérimentations démontrent que la méthode utilisée permet de résoudre des problèmes de diagnostic hors de portée des autres techniques de diagnostic.

2) Un autre problème est que cette méthode requiert de résoudre un certain nombre de questions de diagnostic. Si chaque question doit être résolue séparément, le calcul peut devenir coûteux. Pour comparer, les techniques existantes calculent tous les chemins compatibles avec les observations avant de répondre au problème de diagnostic ; cette approche, quoique coûteuse, ne nécessite qu'un seul calcul complexe pour tous les problèmes de diagnostic. L'approche présentée dans cet article peut être étendue avec les capacités incrémentales de certains solveurs SAT (Hooker, 1993).

5. Modélisation des problèmes SAT

Cette section présente précisément comment nous proposons de traduire une question de diagnostic en problème SAT. La section précédente a montré que le problème SAT est constitué de trois ensembles de contraintes que nous étudions en détail. Pour chaque ensemble, des variables additionnelles peuvent être définies.

5.1. Traduire le modèle

Les contraintes *Mod* présentées ici s'assurent que toute affectation satisfaisant *Mod* correspond à un chemin sur le modèle. Pour simplifier la CNF, nous définissons des variables SAT auxiliaires : pour chaque composant $c_i \in C$, pour chaque règle $r \in R_i$, pour chaque pas de temps $j \in \{1, \dots, n\}$, est définie la variable $r[t = j]$; cette variable est évaluée à *vrai* ssi la règle r est tirée par le composant c_i à la j ème transition, c'est-à-dire si la transition est due à l'application de cette règle.

Les clauses s'assurant que la solution modélise un chemin sur le modèle sont données sur la figure 4. Voici la signification de chaque contrainte :

[1]: Au moins une valeur doit être associée à chaque variable d'état.

$$\begin{aligned} & \forall c_i \in C, \forall v \in V_i, \forall j \in \{0, \dots, n\}, \\ & \quad \bigvee_{\nu \in d(v)} (v = \nu)[t = j] \end{aligned} \quad [1]$$

$$\forall c_i \in C, \forall v \in V_i, \forall \{\nu, \nu'\} \subseteq d(v) \text{ tels que } \nu \neq \nu', \forall j \in \{0, \dots, n\}, \quad [2]$$

$$\neg(v = \nu)[t = j] \vee \neg(v = \nu')[t = j]$$

$$\forall c_i \in C, \forall r = \langle pre, e^I, E^O, eff \rangle \in R_i, \forall j \in \{1, \dots, n\}, \quad [3]$$

$$r[t = j] \rightarrow pre[t = j - 1]$$

$$\forall c_i \in C, \forall r = \langle pre, e^I, E^O, eff \rangle \in R_i, \forall v \in V_i \text{ telles que } eff(v) \text{ est défini,} \quad [4]$$

$$\forall j \in \{1, \dots, n\}, \quad r[t = j] \rightarrow (v = eff(v))[t = j]$$

$$\forall c_i \in C, \forall v \in V_i, \forall \nu \in d(v), \forall j \in \{1, \dots, n\}, \quad [5]$$

$$\neg(v = \nu)[t = j - 1] \rightarrow (v = \nu)[t = j] \rightarrow \bigvee_{r \in R_{v, \nu}} r[t = j]$$

$$\forall c_i \in C, \forall r = \langle pre, e^I, E^O, eff \rangle \in R_i, \forall j \in \{1, \dots, n\}, \quad [6]$$

$$r[t = j] \rightarrow e^I[t = j]$$

$$\forall c_i \in C, \forall r = \langle pre, e^I, E^O, eff \rangle \in R_i, \forall e \in E^O, \forall j \in \{1, \dots, n\}, \quad [7]$$

$$r[t = j] \rightarrow e[t = j]$$

$$\forall c_i \in C, \forall e \in E_i, \forall j \in \{1, \dots, n\}, \quad [8]$$

$$e[t = j] \rightarrow \bigvee_{r \in R_e} r[t = j]$$

$$\forall \langle e_1, e_2 \rangle \in S, \forall \{k, k'\} = \{1, 2\}, \forall j \in \{1, \dots, n\}, \quad [9]$$

$$e_k[t = j] \rightarrow e_{k'}[t = j]$$

$$\forall c_i \in C, \forall \{r_1, r_2\} \subseteq R_i \text{ telles que } r_1 \neq r_2, \forall j \in \{1, \dots, n\}, \quad [10]$$

$$\neg r_1[t = j] \vee \neg r_2[t = j]$$

$$\forall c_i \in C, \forall v \in V_i, \quad [11]$$

$$(v = v(I_i))[t = 0]$$

Figure 4. *Clauses s'assurant que l'application ass modélise un chemin du système*

[2]: Au plus une valeur peut être associée à chaque variable d'état.²

[3]: Une règle ne peut être tirée que si sa précondition est satisfaite.

[4]: Le tirage d'une règle implique ses effets.

[5]: Le changement de valeur de la variable v implique le tirage d'une règle générant la nouvelle valeur. (L'ensemble des règles qui assignent ν à v est représenté par $R_{v, \nu}$.)

2. Cette contrainte pourrait être traduite sous forme de contrainte de cardinalité en ajoutant des variables auxiliaires et réduisant le nombre de clauses (Marques Silva et coll., 2007).

[6]: Le tirage de la règle implique l'occurrence de son événement d'entrée.

[7]: Le tirage de la règle implique l'occurrence de ses événements de sortie.

[8]: L'occurrence d'un événement implique le tirage d'une règle générant cet événement. (L'ensemble des règles liées à l'événement e est représenté par R_e .)

[9]: Les événements de synchronisation doivent avoir lieu en même temps.

[10]: On ne peut tirer qu'une seule règle par composant et par pas de temps.

[11]: Le chemin doit commencer à l'état initial.

Notons que les contraintes définies ici ne forcent pas l'occurrence d'un et un seul événement exogène par pas de temps. De manière générale, le nombre de transitions dans la trajectoire que nous cherchons est inconnu (voir la discussion en début de 5.2.2) et ne pas inclure cette contrainte dans la traduction du modèle permet de considérer des trajectoires de différentes longueurs. Les observations permettent cependant parfois de connaître le nombre précis de transitions, et cette contrainte est alors incluse dans *Obs*. À ce détail près, il est trivial de montrer que pour toute affectation *ass* satisfaisant *Mod*, il existe un chemin *traj* de taille n sur le système, et réciproquement.

Il est possible d'améliorer le codage du modèle de trois manières :

1) Suppression des variables redondantes : le but est de réduire le nombre de variables dans le problème SAT. Cette technique permet de réduire la charge du solveur tout en garantissant la précision des résultats.

- Quand le domaine d'une variable v est binaire ($d(v) = \{\nu_1, \nu_2\}$), la négation de $(v = \nu_1)[t = j]$ implique $(v = \nu_2)[t = j]$. Une de ces variables peut donc être supprimée.

- Les événements de synchronisation ont toujours lieu ensemble. Ainsi, si $\langle e_1, e_2 \rangle \in S$, alors les affectations de $e_1[t = j]$ et de $e_2[t = j]$ seront toujours les mêmes par la contrainte [9]. Il est donc possible de fusionner ces variables.

- Il arrive fréquemment qu'un événement e soit associé à une seule règle ; le tirage de cette règle r est alors équivalent à l'occurrence de l'événement e . Les variables correspondantes peuvent être fusionnées.

2) Factorisation : le but est de reformuler les contraintes pour réduire le nombre de clauses. Cette opération permet de réduire la mémoire employée par le solveur SAT (Rintanen, 2006). Un exemple typique est la contrainte [10] qui indique qu'une seule règle peut être tirée par composant à une date donnée. Exactement un événement d'entrée e^I est associé par règle ; on peut factoriser les clauses portant sur des règles de différents e^I s en disant qu'un seul e^I est autorisé par pas de temps et par composant. Ainsi, pour j événements d'entrée associés chacun à k règles, on passe de $\frac{(j \times k)(j \times k + 1)}{2}$ à $\left(\frac{j(j+1)}{2} + j \frac{k(k+1)}{2}\right)$ clauses binaires.

3) Information : le but est d'insérer des clauses redondantes qui permettent au solveur d'inférer plus rapidement des résultats intéressants. Considérons les clauses présentées sur la figure 5. La dernière clause peut être inférée à partir des premières ; elle est donc redondante et peut être supprimée. Cependant, si a est affectée à *vrai*,

$$\begin{array}{c}
 a \rightarrow b_1 \vee \dots \vee b_k \\
 b_1 \rightarrow c \\
 \dots \\
 b_k \rightarrow c \\
 \hline
 a \rightarrow c
 \end{array}$$

Figure 5. La règle de l'hyper-résolution

cette dernière clause permet d'inférer immédiatement que c doit également être affectée à *vrai*. Bien que redondante, la clause permet de simplifier la recherche (voir par exemple la notion de *arc consistency* dans (Bailleux et coll., 2006)). Dans notre situation, un événement e peut être associé à plusieurs règles qui conduisent chacune à la même affectation $v := \nu$. Il est alors possible d'insérer cette information directement dans la CNF : $e[t = j] \rightarrow (v = \nu)[t = j + 1]$.

Nous comptons également employer des techniques d'information supplémentaires ; l'une d'entre elles consisterait à indiquer que si l'affectation de la variable v à la date j est ν_1 , alors l'affectation à la date $j + 1$ ne peut être ν_2 . Il y a un compromis à trouver entre ajouter de l'information (pour accélérer la recherche) et ne pas ajouter d'information redondante (pour ne pas engorger la mémoire). À part les dernières propositions évoquées, toutes ces techniques ont été implémentées et ont conduit à une amélioration du temps de calcul pour tous les solveurs utilisés.

Remarquons que la génération des contraintes SAT est symétrique pour tous les pas de temps. Cela signifie d'une part que la complexité de générer les contraintes SAT liées au modèle est linéaire en n , et, d'autre part, qu'il est possible d'employer des techniques de précalcul relativement coûteuses, par exemple hors-ligne, pour améliorer la traduction des contraintes puisque ces améliorations sont utilisées intensivement.

5.2. Traduire les observations

Nous considérons à présent la traduction des observations sur le chemin recherché. Selon le type d'observations, il est plus ou moins facile de représenter les observations. Nous considérons ici des observations de plus en plus incertaines, datées, totalement ordonnées et partiellement ordonnées.

5.2.1. Observations datées

Les observations datées sont les observations retournées par la fonction *obs* présentée précédemment. Dans ce cas, le nombre n de transitions et la date d'occurrence des événements observables sont précisément connus. Il est nécessaire d'ajouter deux types de contraintes supplémentaires :

1) Les contraintes indiquant l'occurrence ou non d'événements observables (figure 6). Ceci est simplement représenté par une clause de taille 1 pour chaque événement observable et chaque pas de temps. Dans les contraintes, $obs[j]$ représente l'ensemble des observations effectuées au pas de temps j .

2) Les contraintes indiquant que exactement un événement exogène a lieu à chaque pas de temps (figure 7). Encore une fois, on peut préférer une implémentation sous forme de contraintes de cardinalité.

$$\forall c_i \in C, \forall e \in E_i^{obs}, \forall j \in \{0, \dots, n\}, \text{ si } e \in obs[j] \quad e[t = j] \quad [12]$$

$$\forall c_i \in C, \forall e \in E_i^{obs}, \forall j \in \{0, \dots, n\}, \text{ si } e \notin obs[j] \quad \neg e[t = j] \quad [13]$$

Figure 6. Contraintes des observations datées

$$\forall c_{i_1}, c_{i_2} \in C, \forall e_1 \in E_{i_1}^{exo}, \forall e_2 \in E_{i_2}^{exo}, \forall j \in \{0, \dots, n\}, \quad \neg e_1[t = j] \vee \neg e_2[t = j] \quad [14]$$

$$\forall j \in \{0, \dots, n\}, \quad \bigvee_{e \in E^{exo}} e[t = j] \quad [15]$$

Figure 7. Contraintes forçant exactement une transition par pas de temps

5.2.2. Observations totalement ordonnées

L'hypothèse la plus utilisée est généralement celle considérant les observations comme étant *totalement ordonnées*, aussi appelées *certaines*. Dans ce cas, les ensembles d'observations vides renvoyés par la méthode *obs* sont inconnus. Les observations sont simplement une séquence de *o salves* $obs[i]$ d'observations, chaque salve étant un ensemble non vide d'observations simultanées. Cela signifie qu'il est impossible *a priori* de connaître précisément le nombre de transitions sans observation ayant été générées entre deux salves consécutives données. Puisqu'on ne connaît pas le nombre exact de transitions, il faut accepter *tous* les chemins du système. Quelle valeur faut-il choisir pour n ? Nous faisons ici deux remarques.

1) Puisque notre modélisation SAT autorise qu'aucune transition n'ait lieu pendant un pas de temps, *Mod* représente tous les chemins de longueur n ou moins. Ainsi, si n vaut au moins la longueur du plus long chemin comprenant o salves d'observations, alors tous les chemins compatibles avec *obs* sont considérés. Quand le système autorise des boucles ne générant aucune observation, le n nombre ne peut être borné. Certaines approches interdisent les boucles invisibles (Jiroveanu et coll., 2005). Cependant les boucles invisibles ne changent pas le diagnostic ou représentent un comportement sans intérêt. Par exemple, si une faute peut survenir avant que le système ne revienne à l'état d'origine sans qu'aucune observation n'ait été générée, cette faute temporaire ne peut de toute façon pas être détectée et on peut donc l'ignorer.

2) Notre modélisation autorise que plusieurs macro-transitions aient lieu de manière concurrente, dès lors qu'elles portent sur des ensembles de composants disjoints. Il est alors possible de diminuer la valeur n . Notons que plusieurs chemins correspondant à différents ordonnancements de macro-transitions concurrentes sont alors modélisés par la même affectation. Ainsi, $\langle q_0, q'_0 \rangle \xrightarrow{a|\emptyset} \langle q_1, q'_0 \rangle \xrightarrow{b|\emptyset} \langle q_1, q'_1 \rangle$ et $\langle q_0, q'_0 \rangle \xrightarrow{b|\emptyset} \langle q_0, q'_1 \rangle \xrightarrow{a|\emptyset} \langle q_1, q'_1 \rangle$ sont modélisés par la même affectation si on comprime les deux transitions en un seul pas de temps. Cela contredit la condition autorisant l'utilisation de SAT. Remarquons cependant que les deux trajectoires sont identiques du point de vue du diagnostic ; laquelle des deux a eu lieu en premier est peu importante et de toute façon indéterminable.

Il est possible d'appliquer deux approches pour le calcul de n .

1) On peut borner n de manière globale et indiquer que chaque salve a pu avoir lieu à n'importe quel pas de temps (tout en maintenant l'ordre total entre les salves).

2) On peut borner par m le nombre de pas de temps entre deux salves consécutives et forcer le pas de temps de chaque salve, conduisant à $n = (m + 1) \times o$ pas de temps (les transitions après la dernière salve sont ignorées (Sampath et coll., 1995)).

La première approche permet de restreindre plus fortement n . En effet, sur l'exemple de la figure 8 où o_1 et o_2 sont observables, le nombre de transitions non observables entre deux observations peut atteindre 2 ; la seconde approche nécessite donc $n = 3o$ pas de temps. La première méthode pourrait descendre à $n(o) = 2o + 1$.

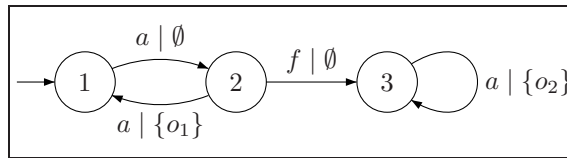


Figure 8. Illustration du problème de pas de temps

Il y a cependant plusieurs raisons pour préférer la seconde solution. Tout d'abord, il est bien plus facile de calculer m qu'un $n(o)$ inférieur à $(m + 1) \times o$. Ensuite, nous faisons remarquer que le pas de temps où l'observation numéro i a lieu varie dans le premier cas. La modélisation des observations en SAT devient alors complexe et réduit les performances du solveur SAT.

Les contraintes SAT pour des observations certaines sont présentées figure 9. Le sens de chaque clause est :

[16] : Aucune observation n'est émise pendant les transitions invisibles.

[17] : Les événements non observés pendant la $\frac{j}{m+1}$ -ème salve n'ont pas lieu date j .

[18] : Les événements observés pendant la $\frac{j}{m+1}$ -ème salve ont lieu date j .

$$\forall j \in \{0, \dots, n\} \text{ tel que } j \neq 0 \pmod{m+1}, \forall c_i \in C, \forall e \in E_i^{obs}, \quad [16]$$

$$\neg e[t = j]$$

$$\forall j \in \{0, \dots, n\} \text{ tel que } j = 0 \pmod{m+1}, \forall e \in E^{obs} \setminus obs[\frac{j}{m+1}], \quad [17]$$

$$\neg e[t = j]$$

$$\forall j \in \{0, \dots, n\} \text{ tel que } j = 0 \pmod{m+1}, \forall e \in obs[\frac{j}{m+1}], \quad [18]$$

$$e[t = j]$$

Figure 9. *Clauses qui représentent les observations certaines*

5.2.3. Observations partiellement ordonnées

Dans de nombreux systèmes décentralisés par nature, il est souvent impossible de déterminer quelle observation a été émise en premier (Benveniste et coll., 2003; Wang et coll., 2004; Pencolé et coll., 2005; Su et coll., 2005). Dans ce cas, les salves d'observations ne sont pas complètement ordonnées. Il est donc nécessaire de modéliser tous les ordres possibles des salves.

Puisque l'indice i de $obs[i]$ n'a plus de signification, nous simplifions les notations par $\xi = obs[i]$. Pour chaque salve d'observations $\xi \in obs$ et pour chaque pas j de temps où les transitions observables sont autorisées (c'est-à-dire telles que $j \pmod{m+1} = 0$), on définit deux variables :

- $\xi[t = j]$ est affecté à vrai si la salve ξ a été générée à la date j , et
- $\widehat{\xi[t = j]}$ est affecté à vrai si la salve ξ a été générée à la date j ou avant.

En plus des contraintes [16], les observations partiellement ordonnées sont présentées en figure 10. Le sens de chaque clause est :

[19] : $\widehat{\xi[t = j]}$ est défini récursivement à l'aide de $\xi[t = j]$.

[20] : Chaque salve ξ n'a lieu qu'une fois. . .

[21] : . . . mais chaque salve ξ est effectivement émise.

[22] : L'ordre entre les salves $\xi \prec \xi'$ est forcé.

[23] : Deux salves $\xi \neq \xi'$ n'ont pas lieu simultanément.

[24] : Les événements observables d'une salve ξ ont effectivement lieu.

[25] : Les événements observables n'ont lieu que pendant une salve ξ .

$$\begin{aligned} & \forall \xi \in obs, \forall k \in \{1, \dots, o\}, \\ \xi[t = \widehat{k \times (m+1)}] & \leftrightarrow \xi[t = \widehat{(k-1) \times (m+1)}] \vee \xi[t = k \times (m+1)] \\ & \text{où } \xi[\widehat{t=0}] \text{ vaut } \perp. \end{aligned} \quad [19]$$

$$\begin{aligned} & \forall \xi \in obs, \forall k \in \{1, \dots, o\}, \\ \xi[t = \widehat{(k-1) \times (m+1)}] & \rightarrow \neg \xi[t = k \times (m+1)] \\ & \text{où } \xi[\widehat{t=0}] \text{ vaut } \perp. \end{aligned} \quad [20]$$

$$\begin{aligned} & \forall \xi \in obs, \\ \xi[t = \widehat{o \times (m+1)}] & \end{aligned} \quad [21]$$

$$\begin{aligned} & \forall \xi, \xi' \in obs \text{ tels que } \xi \prec \xi', \forall k \in \{1, \dots, o\}, \\ \xi'[t = k \times (m+1)] & \rightarrow \xi[t = \widehat{(k-1) \times (m+1)}] \end{aligned} \quad [22]$$

$$\begin{aligned} & \forall \xi, \xi' \in obs \text{ tels que } \xi \neq \xi', \forall k \in \{1, \dots, o\}, \\ \neg \xi[t = k \times (m+1)] & \vee \neg \xi'[t = k \times (m+1)] \end{aligned} \quad [23]$$

$$\begin{aligned} & \forall \xi \in obs, \forall e \in \xi, \forall k \in \{1, \dots, o\}, \\ \xi[t = k \times (m+1)] & \rightarrow e[t = k \times (m+1)] \end{aligned} \quad [24]$$

$$\begin{aligned} & \forall e \in E^{obs}, \forall k \in \{1, \dots, o\}, \\ e[t = k \times (m+1)] & \rightarrow \bigvee_{\xi \in obs | e \in \xi} \xi[t = k \times (m+1)] \end{aligned} \quad [25]$$

Figure 10. Clauses qui représentent les observations partiellement ordonnées

5.3. Traduire le type de chemin recherché

Il est possible de définir des clauses pour représenter le type de chemin recherché. Ces clauses sont définies à partir des variables propositionnelles correspondant à l'occurrence de certains événements pertinents ou à certaines valuations de variables d'états particulières. Ces variables supplémentaires permettent une formulation du problème qui le rend plus facile à résoudre pour le solveur.

Considérons, par exemple, que certains événements $F \subseteq E$ sont qualifiés de *pannes*. Pour déterminer s'il existe une explication pour les observations qui ne comporte aucune panne, il est possible d'insérer les clauses de la figure 11.

$$\begin{aligned} & \forall e \in F, \forall j \in \{0, \dots, n\}, \\ & \quad \neg e[t = j] \end{aligned} \quad [26]$$

Figure 11. Clause s'assurant qu'aucune panne n'a lieu

Si, au contraire, la question de diagnostic requiert de trouver un chemin avec au moins une panne, il est possible d'insérer la clause suivante de la figure 12.

$$\bigvee_{j \in \{1, \dots, n\}} \bigvee_{e \in F} e[t = j] \quad [27]$$

Figure 12. Clause s'assurant que au moins une panne a lieu

Dans le domaine du diagnostic, les pannes sont souvent des événements exceptionnels. Ainsi, si plusieurs hypothèses de diagnostic peuvent générer les observations, l'évolution la plus réaliste est souvent celle qui implique le moins de pannes. Il est donc judicieux de chercher l'explication qui minimise le nombre de pannes (diagnostic de cardinalité minimale (Reiter, 1987)). Dans ce cas, on peut poser des questions de diagnostic avec un nombre croissant d'occurrences de panne. Cette requête correspond à une contrainte de cardinalité sur l'ensemble des variables $\{e[t = j] \mid e \in F \wedge j \in \{1, \dots, n\}\}$ (Bailleux et coll., 2003).

Il est possible de représenter des types de chemins plus complexes comme proposé par (Jéron et coll., 2006). D'autres types de requêtes peuvent être aussi facilement représentées. Il est possible de coder différemment en CNF la même contrainte, mais le choix de ce codage peut avoir un impact critique sur l'efficacité du solveur SAT comme nous l'avons montré dans (Anbulagan et coll., 2009) ; un soin particulier doit donc y être apporté.

6. Expérimentations

Nous avons effectué une série d'expérimentations pour valider l'approche présentée dans cet article. La littérature du diagnostic de SÉD ne dispose pas de benchmark permettant de comparer les caractéristiques des différentes approches ; nous avons donc décidé de créer un exemple artificiel. Nous utilisons le réseau modélisé et présenté en section 2. Ce réseau a été construit pour rendre le diagnostic particulièrement difficile. Bien que compact (il se modélise par 80 variables booléennes), il comprend beaucoup d'états : près de la totalité des 6^{20} états potentiels sont accessibles. De plus, les pannes sont très difficiles à diagnostiquer : elles ne sont pas diagnosticables (Rintanen et coll., 2007) ; elles sont interdépendantes : l'occurrence d'une panne peut influencer sur les effets d'une autre panne ; les différentes pannes partagent un grand nombre de symptômes. Enfin, le modèle comprend un certain nombre d'événements concurrents particulièrement difficiles à suivre. Nos implémentations des algorithmes classiques de la littérature échouent à résoudre ces problèmes : soit ils sont confrontés à l'explosion combinatoire, soit ils ne sont pas capables de rendre un résultat précis.

Nous avons simulé un certain nombre de scénarios sur le système. Selon le comportement du système, les problèmes sont plus ou moins faciles à résoudre. En particulier, quand les composants retournent rapidement à leur état initial (état 0 sur la figure 1), le problème est bien plus difficile à résoudre que si les composants restent

dans les autres états (auquel cas les conséquences des fautes se chevauchent). Aussi, notre générateur aléatoire introduit 10 niveaux de complexité basés sur cette propriété. En outre, nous générons des scénarios avec un nombre k de fautes variant de 1 à 20 ; chaque scénario comporte environ $8 \times k$ observations. Enfin, nous considérons les trois types d'observations présentées en sous-section 5.2 ; pour les observations partiellement ordonnées, l'ordre entre l'observation numéro i et l'observation numéro j est connu ssi $|j - i| \geq 5$. Nos expérimentations comportent donc $10 \times 20 \times 3 = 600$ problèmes de diagnostic. Les données utilisées pour les expérimentations sont accessibles à cette adresse : <http://www.grastien.net/ban/data/ria10/>.

Notre problème de diagnostic consiste à déterminer l'explication la plus plausible des observations. Cette explication est le chemin du SÉD qui minimise le nombre d'événements improbables, lesquels correspondent à la transition entre l'état 0 et l'état F. Pour ce faire, nous cherchons tout d'abord une explication comportant $k = 0$ faute, puis nous incrémentons cette valeur jusqu'à trouver une explication.

Pour chaque problème, nous accordons au diagnostiqueur 30 minutes (1 800 secondes). La traduction de la question de diagnostic en problème SAT est très rapide et peut se faire hors-ligne (à part la formule Obs) : nous ne donnons donc ici que le temps de calcul du solveur SAT, MINISAT 2.0 version *core* (<http://www.minisat.se/>). Nous avons choisi ce solveur parce que son code est disponible, parce qu'il est relativement simple et permettra donc d'implémenter et de tester des fonctionnalités spécifiques pour le diagnostic et enfin parce que les performances des meilleurs solveurs sont finalement très comparables. Les expérimentations ont été effectuées sur un processeur Intel Xeon CPU E5405 à 2.00 GHz avec un minimum de 4 GB de mémoire, fonctionnant sous Linux.

Résultats Les temps de calcul sont donnés sur la figure 14. Pour chaque catégorie de problèmes, les temps sont classés par ordre croissant. Par ailleurs, le temps moyen de résolution (où 1 800s sont attribuées si le problème n'est pas résolu), le temps médian et le nombre de problèmes non résolus sont donnés dans les tableaux annexes.

a) Le diagnostic pour des observations partiellement ordonnées est bien plus difficile que pour des observations datées ou totalement ordonnées. On note un facteur 10 pour les derniers problèmes résolus, et l'écart ne cesse d'augmenter avec la complexité.

b) L'instance du problème a également un fort impact sur le temps de calcul. Ici encore, on note un facteur 10.

c) Le dernier paramètre est le nombre de fautes. On voit que la complexité de diagnostiquer k fautes n'est pas proportionnelle par rapport à k .

Analyse La raison de la dégradation des temps de calcul est la même pour chaque catégorie. L'une des forces de SAT est sa capacité à propager des inférences. Ceci est illustré à l'aide de la figure 13 où a et b sont observables et u n'est pas observable. Étant donné la séquence d'observations $[a, b]$, un algorithme à base de dépliage commence par expliquer a avant de chercher à expliquer b . Au contraire, un algorithme utilisant SAT infère immédiatement de l'observation b que l'état avant cette

observation est 2 et que la transition ayant généré a est la transition $0 \xrightarrow{a} 2$. Ces inférences permettent d'éviter une grande partie de la recherche. Les inférences de SAT sont cependant limitées, comme nous l'avons vu avec l'hyper-résolution. Une fois les pré-calculs terminés, seule la *propagation unitaire* (le *modus ponens*) est appliquée.

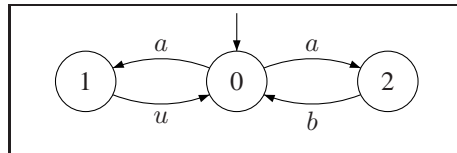


Figure 13. Exemple de modèle

Lorsque les observations sont partiellement ordonnées, la date des observations est inconnue ce qui empêche les inférences. Pour reprendre l'exemple de la figure 13, considérons que la date d'occurrence de a est $t_a = 1$ et la date d'occurrence de b est $t_b \in \{2, 3\}$; de ce problème, il est possible de prouver que la transition associée à a est $0 \xrightarrow{a} 2$, mais SAT n'est pas capable de l'inférer facilement dans le cas général.

La difficulté du scénario a une influence similaire. Les scénarios difficiles sont spécifiquement construits pour ne pas permettre d'inférence facile. Les fautes sont intriquées, ce qui oblige d'effectuer une recherche pour dénouer les alarmes.

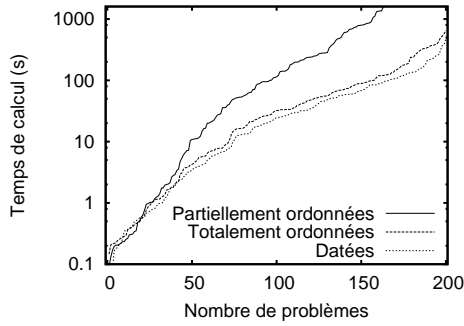
Finalement, le nombre de fautes influe de deux manières sur les observations. D'une part, et pour le système utilisé lors des expérimentations, un plus grand nombre de fautes implique un scénario plus long (puisque les observations ne sont générées que lorsqu'il y a des pannes). D'autre part, lorsque le nombre de fautes augmente, la complexité de la formule *Que* augmente. Nous avons étudié comment formuler une telle requête de manière efficace ; différentes modélisations conduisent à des temps de calcul différant de plusieurs ordres de grandeur (Anbulagan et coll., 2008).

Les résultats présentés ici sont très encourageants. Ils démontrent que SAT peut être utilisé pour résoudre des problèmes de diagnostic difficiles. Nous pensons qu'il est possible d'améliorer encore grandement les performances, et nous travaillons actuellement sur ce point.

7. Travaux futurs

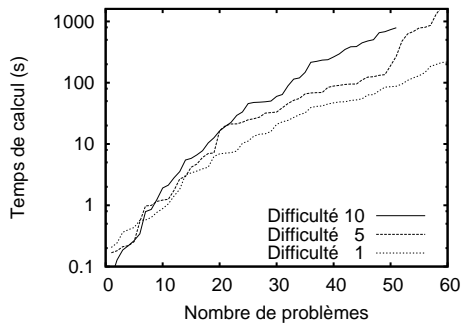
Les développements actuels et futurs de ces travaux peuvent être grossièrement classés en deux catégories qui concernent chacune les deux niveaux de l'algorithme :

- premier niveau de l'algorithme : expressivité,
- second niveau de l'algorithme : complexité.



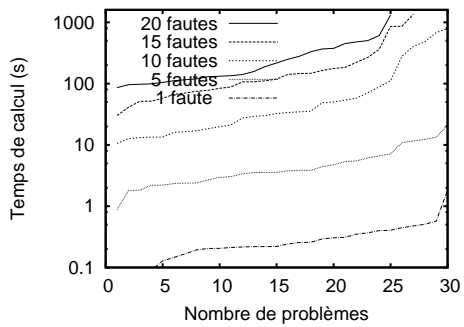
	Tps moy	Tps med	Non rés.
Dat.	52s	25s	0
T. O.	77s	32s	0
P. O.	506s	63s	34/200

a) Temps de calcul pour différentes observations.



	Tps moy	Tps med	Non rés.
1	44s	23s	0
5	192s	33s	1/60
10	402s	48s	9/60

b) Temps de calcul pour différentes difficultés.



	Tps moy	Tps med	Non rés.
1	<1s	<1s	0
5	5s	4s	0
10	117s	34s	0
15	381s	111s	3/30
20	530s	157s	5/30

c) Temps de calcul pour différents nombres de fautes.

Figure 14. Temps de calcul selon différents critères

7.1. *Expressivité*

Cet article s'est principalement concentré sur le second niveau de l'algorithme et en particulier la transformation d'une question de diagnostic en problème SAT. Comment traduire le problème de diagnostic en une série de questions de diagnostic reste, de manière générale, un problème ouvert. Les questions générées par la traduction doivent satisfaire un certain nombre de critères.

– Elles doivent être raisonnablement peu nombreuses. Si le diagnostic est défini comme l'ensemble des hypothèses compatibles avec les observations, et si cet ensemble est de taille exponentielle, il est possible de définir une question pour chaque hypothèse. En revanche, il sera impossible de tester chacune de ces questions. Une autre traduction est donc nécessaire.

– Elles doivent être raisonnablement faciles à résoudre. En particulier, des travaux récents (Sachenbacher et coll., 2005; Pencolé et coll., 2006; Kan John et coll., 2010) visent à déterminer quelles sont les parties du système qu'il est suffisant de surveiller pour répondre à une question précise. Il est important d'être conscient des capacités du second niveau de l'algorithme de diagnostic lorsque les questions sont définies.

– La transformation en questions de diagnostic n'est pas toujours triviale. Par exemple, si le but du diagnostic est de trouver les composants en panne dans le système, chaque composant peut être testé séparément. On passe alors d'un nombre d'hypothèses exponentiel par rapport au nombre de composant à un nombre linéaire. Cependant, on perd alors la corrélation entre les composants fautifs. Par exemple, si les observations permettent de déterminer qu'un composant inconnu est en panne, alors l'étude séparée de chaque composant ne produit aucun diagnostic utile. Si on cherche à appliquer des traductions non triviales, il devient nécessaire d'étudier dans quelles circonstances elles sont applicables, et quelle précision elles peuvent fournir.

7.2. *Complexité*

Ici, le but est d'améliorer les temps de calcul du diagnostic. Les présents travaux ont été initiés pour étudier comment utiliser SAT pour le diagnostic de SÉD. Les améliorations consistent principalement dans la traduction du problème en CNF.

– Contraintes de cardinalité : nous avons expliqué que pour certains problèmes de diagnostic, le but est de minimiser le nombre de fautes sur le système. La formulation de la contrainte de cardinalité est extrêmement importante et nous avons déjà déterminé qu'utiliser des techniques naïves peut multiplier le temps de calcul par plusieurs ordres de grandeur (Anbulagan et coll., 2008; 2009).

– Diagnostic incrémental : lorsque le nombre d'observations est trop important, les problèmes SAT deviennent trop grands et impossibles à résoudre. C'est en particulier le cas pour le diagnostic en-ligne (c.-à-d. quand de nouvelles observations sont continuellement fournies par le système), mais également pour certains problèmes de diagnostic *a posteriori*. L'idée du diagnostic incrémental est de calculer le diag-

nostic sur les premières observations (soit que les autres sont trop nombreuses, soit qu'elles ne sont pas encore disponibles), et de simplement mettre à jour celui-ci avec les observations suivantes, plutôt que de recommencer le calcul à zéro. Cela pose des problèmes de cohérence que nous avons étudiés (Grastien et coll., 2009).

– SAT incrémental : le problème de SAT incrémental (Hooker, 1993) est le suivant. Étant donné une CNF $\Phi_1 \wedge \Phi_2$ ayant déjà été évaluée par le solveur, évaluer la CNF $\Phi_1 \wedge \Phi_3$. Les deux CNFs sont très semblables il est donc possible de réutiliser partiellement le premier calcul pour accélérer le calcul du second. Beaucoup de questions de diagnostic sont très similaires : les clauses *Mod* et *Obs* sont identiques d'une question à l'autre. L'apport des techniques incrémentales requiert d'être correctement étudié ; nos premières expérimentations ne montrent pas d'amélioration sensible.

– Solveur informé : nous avons déjà mentionné qu'ajouter des clauses redondantes dans la CNF permet d'accélérer la procédure de recherche, notamment grâce à la propagation unitaire. Il y a là un compromis qu'il est nécessaire d'étudier pour éviter d'ajouter trop de clauses et réduire les performances du solveur.

D'autre part, il est possible d'effectuer des techniques de précalcul pour réduire l'espace de recherche. Ainsi, le diagnostic local a un coût ridiculement faible mais n'est pas complet (voir (Su et coll., 2005)) ; il pourrait cependant être utilisé pour déterminer facilement un certain nombre d'implications relativement triviales.

Enfin, nous pensons que l'ordre d'affectation des variables par le solveur SAT peut être spécialisé pour les problèmes de diagnostic.

8. Conclusion

Cet article présente une nouvelle technique de diagnostic de système à événements discrets. Plutôt que de calculer tous les chemins du modèle compatibles avec les observations, nous proposons une approche où il est suffisant de calculer *un* ou *plusieurs* chemins particuliers. Le problème de diagnostic est résolu en testant différentes hypothèses de diagnostic, chacune consistant à déterminer s'il existe un chemin du modèle compatible avec les observations. La recherche de chemin est effectuée de manière efficace par un solveur de contrainte SAT.

La plupart des travaux récents sur le diagnostic de SÉD cherchent à permettre le diagnostic de systèmes de grande taille, l'espace d'état d'un système étant exponentiel en nombre de composants du système. Les expérimentations effectuées au cours de cette étude ont permis de résoudre des problèmes de diagnostic impossible à résoudre par les autres techniques de la littérature.

Enfin, nous avons énuméré un certain nombre d'extensions permettant à la fois d'améliorer l'efficacité du calcul et d'augmenter le champ d'application de SAT.

Nous comptons employer les techniques présentées dans cet article pour le projet *AI for the Smart Grid* de NICTA.

Remerciements

NICTA est subventionné par le gouvernement australien représenté par le *Department of Broadband, Communications and the Digital Economy* et l'*Australian Research Council* au travers du programme *ICT Centre of Excellence*.

9. Bibliographie

- Anbulagan, Grastien A., « Importance de la sémantique dans le codage CNF de contraintes de cardinalité : application au diagnostic de SED », *Journées francophones de programmation par contraintes (JFPC-08)*, 2008.
- Anbulagan, Grastien A., « Importance of Variables Semantic in CNF Encoding of Cardinality Constraints », *Symposium on Abstraction, Reformulation and Approximation (SARA-09)*, 2009.
- Anbulagan, Slaney J., « Lookahead Saturation with Restriction for SAT », *International Conference on Principles and Practice of Constraint Programming (CP-05)*, p. 727-731, 2005.
- Audemard G., Simon L., « Predicting Learnt Clauses Quality in Modern SAT Solvers », *International Joint Conference on Artificial Intelligence (IJCAI-09)*, p. 399-404, 2009.
- Bailleux O., Boufkhad Y., « Efficient CNF Encoding of Boolean Cardinality Constraints », *International Conference on Principles and Practice of Constraint Programming (CP'03)*, p. 108-122, 2003.
- Bailleux O., Boufkhad Y., Roussel O., « A translation of pseudo boolean constraints to SAT », *Satisfiability, Boolean Modeling and Computation*, vol. 2, p. 191-200, 2006.
- Benveniste A., Fabre É., Haar St., Jard Cl., « Diagnosis of asynchronous discrete event systems, a net unfolding approach », *IEEE Transactions on Automatic Control*, vol. 48, n° 5, p. 714-727, 2003.
- Biere A., Cimatti A., Clarke E., Zhu Y., « Symbolic model-checking without BDDs », *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-99)*, p. 193-207, 1999.
- Brandán Briones L., Lazovik A., Dague Ph., « Optimal observability for diagnosability », *International Workshop on Principles of Diagnosis (DX-08)*, 2008.
- Cassandras C., Lafortune St., *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, 1999.
- Cook S., « The complexity of theorem-proving procedures », *IEEE Symposium on the Foundations of Computer Science*, p. 151-158, 1971.
- Cordier M.-O., Pencolé Y., Travé-Massuyès L., Vidal Th., « Caractérisation des systèmes autoguérissants : diagnostiquer ce que l'on va réparer », *Congrès Reconnaissance des Formes et Intelligence Artificielle (RFIA-08)*, 2008.
- Davis M., Logemann G., Loveland D., « A Machine Program for Theorem Proving », *Communication of ACM*, vol. 5, p. 394-397, 1962.
- Dubois O., Dequen G., « A Backbone-search Heuristic for Efficient Solving of Hard 3-SAT Formulae », *International Joint Conference on Artificial Intelligence (IJCAI-01)*, p. 248-253, 2001.

- Eén N., Sorensson N., « An Extensible SAT-solver », *Revised Selected Papers of 6th SAT, LNCS 2919 Springer*, p. 502-518, 2004.
- Grastien A., Anbulagan, « Résolution d'un problème de diagnostic de systèmes à événements discrets par SAT », *Journées francophones de programmation par contraintes (JFPC-07)*, 2007a.
- Grastien A., Anbulagan, « Incremental diagnosis of DES with a Non-Exhaustive Diagnosis Engine », *International Workshop on Principles of Diagnosis (DX-09)*, p. 345-352, 2009.
- Grastien A., Anbulagan, Rintanen J., Kelareva E., « Diagnosis of Discrete-Event Systems using Satisfiability Algorithms », *National Conference on Artificial Intelligence (AAAI-07)*, 2007b.
- Hamscher W., Console L., de Kleer J., *Readings in Model-Based Diagnosis*, Morgan Kaufmann Publishers Inc., 1992.
- Heintz F., Krysanter M., Roll J., Frisk E., « FlexDx: a reconfigurable diagnosis framework », *International Workshop on Principles of Diagnosis (DX-08)*, p. 79-86, 2008.
- Heule M., van Maaren H., « March_dl: Adding Adaptive Heuristics and a New Branching Strategy », *Satisfiability, Boolean Modeling and Computation*, vol. 2, p. 47-59, 2006.
- Hooker J., « Solving the incremental satisfiability problem », *Logic Programming*, vol. 15, n° 1-2, p. 177-186, 1993.
- Jéron Th., Marchand H., Cordier M.-O., « Motifs de surveillance pour le diagnostic de systèmes à événements discrets », *Congrès Reconnaissance des formes et Intelligence Artificielle (RFIA-2006)*, 2006.
- Jiroveanu G., Boël R., « Petri Net model-based Distributed Diagnosis for large interacting systems », *International Workshop on Principles of Diagnosis (DX-05)*, p. 25-30, 2005.
- Kan John P., Grastien A., « Local Consistency and Junction Tree for Diagnosis of Discrete-Event Systems », *European Conference on Artificial Intelligence (ECAI'08)*, p. 209-213, 2008.
- Kan John P., Grastien A., Pencolé Y., Ribot P., « Synthèse d'un diagnostiqueur distribué et précis », *Congrès francophone AFRIF-AFIA (RFIA-10)*, p. 646-653, 2010.
- Kautz H., Selman B., « Pushing the envelope: planning, propositional logic, and stochastic search », *National Conference on Artificial Intelligence (AAAI-96)*, p. 1194-1201, 1996.
- Kurtoglu T., Narasimhan S., Poll Sc., Garcia D., Kuhn L., de Kleer J., van Gemund A., Feldman A., « First International Diagnosis Competition – DXC'09 », *International Workshop on Principles of Diagnosis (DX-09)*, p. 383-396, 2009.
- Lamperti G., Zanella M., *Diagnosis of Active Systems*, Kluwer Academic Publishers, 2003.
- Li C. M., Anbulagan, « Heuristics Based on Unit Propagation for Satisfiability Problems », *International Joint Conference on Artificial Intelligence (IJCAI-97)*, p. 366-371, 1997.
- Marchand H., Rozé L., « Diagnostic de pannes sur des systèmes à événements discrets : une approche à base de modèles symboliques », *Congrès Reconnaissance des formes et Intelligence Artificielle (RFIA-2002)*, p. 191-200, 2002.
- Marques Silva J., Lynce I., « Towards robust CNF encodings of cardinality constraints », *International Conference on Principles and Practice of Constraint Programming (CP-07)*, p. 483-497, 2007.
- Mayer W., Stumptner M., « Modeling Context-Dependent Faults for Diagnosis », *International Workshop on Principles of Diagnosis (DX-09)*, p. 211-218, 2009.

- Moskewicz M. W., Madigan C. F., Zhao Y., Zhang L., Malik S., « Chaff: Engineering an Efficient SAT solver », *Design Automation Conference (DAC-01)*, p. 530-535, 2001.
- Pencolé Y., Cordier M.-O., « A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks », *Artificial Intelligence*, vol. 164, p. 121-170, 2005.
- Pencolé Y., Kamenetsky D., Schumann A., « Towards low-cost diagnosis of component-based systems », *IFAC Symposium on Fault Detection, Supervision and Safety of Technical Process (SAFEPROCESS)*, 2006.
- Pipatsrisawat K., Darwiche A., RSat 2.0: SAT Solver Description, Technical Report n° D-153, Automated Reasoning Group, Computer Science Department, UCLA, 2007.
- Reiter R., « A theory of diagnosis from first principles », *Artificial Intelligence*, vol. 32, n° 1, p. 57-95, 1987.
- Rintanen J., « Compact representation of sets of binary constraints », *European Conference on Artificial Intelligence (ECAI-06)*, p. 143-147, 2006.
- Rintanen J., « Diagnosers and Diagnosability of Succinct Transition Systems », *International Joint Conference on Artificial Intelligence (IJCAI-07)*, p. 538-544, 2007.
- Rintanen J., Grastien A., « Diagnosability Testing with Satisfiability Algorithms », *International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2007.
- Rozé L., Cordier M.-O., « Diagnosing discrete-event systems : extending the "diagnoser approach" to deal with telecommunication networks », *Discrete Event Dynamical Systems*, vol. 12, n° 1, p. 43-81, 2002.
- Sachenbacher M., Struss P., « Task-dependent qualitative domain abstraction », *Artificial Intelligence*, vol. 162, n° 1-2, p. 121-143, 2005.
- Sampath M., Sengupta R., Lafortune St., Sinnamohideen K., Teneketzis D., « Diagnosability of discrete-event systems », *IEEE Transactions on Automatic Control*, vol. 40, n° 9, p. 1555-1575, 1995.
- Schumann A., Pencolé Y., Thiébaux S., « Diagnosis of Discrete-Event Systems using Binary Decision Diagrams », *International Workshop on Principles of Diagnosis (DX-04)*, p. 197-202, 2004.
- Schumann A., Pencolé Y., Thiébaux S., « A Spectrum of Symbolic On-line Diagnosis Approaches », *National Conference on Artificial Intelligence (AAAI-07)*, 2007.
- Su R., Wonham W., « Global and local consistencies in distributed fault diagnosis for discrete-event systems », *IEEE Transactions on Automatic Control*, vol. 50, n° 12, p. 1923-1935, 2005.
- Torta G., Theseider Dupré D., Anselma L., « Hypothesis Discrimination with Abstractions based on Observation and Action Costs », *International Workshop on Principles of Diagnosis (DX-08)*, p. 189-196, 2008.
- Wang Y., Yoo T.-S., Lafortune St., « New Results on Decentralized Diagnosis of Discrete Event Systems », *Annual Allerton Conference on Communication, Control, and Computing*, 2004.
- WS-Diamond Team, « WS-DIAMOND: Web Services – DIAGNOSABILITY, MONITORING and DIAGNOSIS », *International Workshop on Principles of Diagnosis (DX'07)*, p. 243-250, 2007.
- Zhang L., Madigan C. F., Moskewicz M. W., Malik S., « Efficient Conflict Driven Learning in a Boolean Satisfiability Solver », *International Conference on Computer Aided Design (ICCAD-01)*, 2001.