DITO: a CSP-based diagnostic engine

Yannick Pencolé[1]

Abstract. We present a new generic diagnostic engine to diagnose systems that are modelled with first order logic. The originality of this engine is that it takes benefit of recent advances in constraint programming to perform satisfiability tests by the use of an off-the-shelf CSP solver. The second contribution of this paper is the definition of an incremental search strategy that deals with the intrinsic complexity of the problem to look for minimal diagnoses in complex digital systems. The use of the DITO engine and its strategy is fully illustrated on the c6288 circuit that is part of the classical ISCAS85 benchmark.

1 INTRODUCTION

Nowadays, the business around system maintenance has tremendously raised. For an industrial company (like aeronautical, automotive, energy companies), selling the way to maintain a complex system is as crucial as selling the system itself. With the help of the new technologies, new type of smart and embedded sensors are available and provide more and more accurate piece of information about the current health of a system that is operating. This is especially the case in the aeronautical field where tools for diagnosis and maintenance assistance is an economical need [1].

Diagnostic reasoning is in the heart of this new economical era, but there is a pitfall: complexity. Diagnostic reasoning has been addressed for many years in the academic world and especially in Artificial Intelligence where the community of Model-Based Diagnosis has developed many theoretical frameworks and algorithms. For instance, solving diagnosis problems based on consistency-checking is a pretty mature field where there exist classical algorithms usually called Generic Diagnosis Engines (GDE for short) to solve these problems [10, 6, 8, 9, 7].

The point of view we address in this paper is the following one. Constraint programming, constraint satisfaction problem are scientific fields with many recent advances that keep improving the performance of constraint solvers. The question is how Model-Based diagnosis could benefit of these advances, as Model-Based diagnosis usually requires a sound and complete theorem prover [10]? We propose DITO a generic diagnosis engine that models diagnosis problem as CSPs and uses an *off-the-shelf* CSP solver to solve the encoded problems. As the ultimate goal of DITO is to solve realistic problems, we also need to address the problem of the complexity that is inherent to any diagnosis problem and independent from any theorem prover. Our strategy to use such a tool is to always provide relevant and complete results even if they are not the most precise ones, this strategy is conservative and is well-suited in the context of maintenance, especially in aeronautics (for certification reasons).

[1] LAAS, CNRS, Univ. de Toulouse, FRANCE, email:yannick.pencole@laas.fr

This paper is organized as follows. Section 2 presents the necessary background about Model-based Diagnosis that is needed to understand how DITO works. Section 3 describes in details how DITO solves diagnosis problems. Section 4 proposes a strategy to effectively and realistically solve diagnosis problems with DITO. Section 5 provides an experimental illustration of how DITO performs on a realistic system, namely the c6288 circuit of the ISCAS85 benchmark [4]. A global discussion about this work is finally proposed in Section 6.

Throughout this paper, any experimental results has been made with the same settings. Even if DITO is modular and can use different CSP solvers, the one that has been used here is Choco 2 [11]. All the experiments were on an i5core (4GB memory). Time measurements always include everything (encoding + constraint solving).

2 BACKGROUND

The theoretical framework of Model-based diagnosis has been introduced by [10]; it is briefly recalled here. First, a *diagnosis model* describes the underlying system as a set of components and a set of behaviours. The model is defined with first-order logics.

Definition 2.1 (Diagnosis Model) *A* diagnosis model *is a pair* $(\mathsf{SD}, \mathsf{COMPS})$ *where:*

- SD *is a finite set of first-order sentences, called the* system description*;*
- COMPS *is the set of components of interest.*

We shall come back to the notion COMPS later on in Section 4, but for now COMPS is a set of terms of SD that represents the set of *components* under monitoring in the system. The *health state* of a component $\mathsf{c} \in \mathsf{COMPS}$ is represented by the predicate $\mathsf{Ab}(\mathsf{c})$ (which means that c is abnormal). Throughout this paper, SD only describes nominal behaviours (faulty behaviours are out of the topic of this paper); it follows that any sentence of SD containing a predicate $\mathsf{Ab}(\mathsf{c})$ can always be equivalently written as a sentence starting like $\neg\mathsf{Ab}(\mathsf{c}) \Rightarrow \ldots$. Figure 2 illustrates such a system description for the circuit depicted in Figure 1, introduced in [5]. The set of components for this circuit is $\mathsf{COMPS} = \{\mathsf{M}_1, \mathsf{M}_2, \mathsf{M}_3, \mathsf{A}_1, \mathsf{A}_2\}$. Each sentence of Figure 2 describes the nominal behaviour of the circuit. For instance the sentence $\mathsf{M}_1\mathsf{desc}$ states that if the component M_1 is normal then the valuation of its output x is the product of the valuation of its inputs a and c.

The objective of the diagnostic reasoning is to determine the set of possible health states of the system which is represented as a conjunction

$$\sigma(\Delta) \stackrel{\mathbf{def}}{=} \bigwedge_{\mathsf{c}\in\Delta} \mathsf{Ab}(\mathsf{c}) \wedge \bigwedge_{\mathsf{c}\in\mathsf{COMPS}\setminus\Delta} \neg\mathsf{Ab}(\mathsf{c}).$$

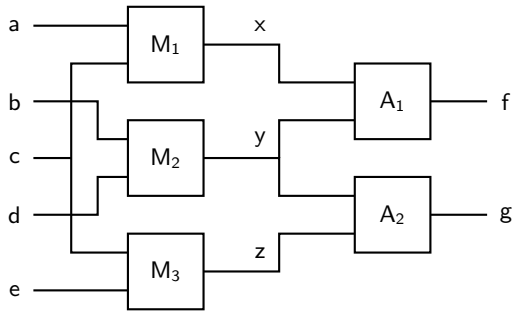

Figure 1. A logical circuit with 3 multipliers (M_1, M_2 and M_3) and 2 adders (A_1 and A_2).

$$
\begin{array}{ll}
\mathsf{M_1desc}: & \neg\mathsf{Ab}(\mathsf{M_1}) \Rightarrow (\mathsf{v(x)} = \mathsf{v(a)} * \mathsf{v(c)}) \\
\mathsf{M_2desc}: & \neg\mathsf{Ab}(\mathsf{M_2}) \Rightarrow (\mathsf{v(y)} = \mathsf{v(b)} * \mathsf{v(d)}) \\
\mathsf{M_3desc}: & \neg\mathsf{Ab}(\mathsf{M_3}) \Rightarrow (\mathsf{v(z)} = \mathsf{v(c)} * \mathsf{v(e)}) \\
\mathsf{A_1desc}: & \neg\mathsf{Ab}(\mathsf{A_1}) \Rightarrow (\mathsf{v(f)} = \mathsf{v(x)} + \mathsf{v(y)}) \\
\mathsf{A_2desc}: & \neg\mathsf{Ab}(\mathsf{A_2}) \Rightarrow (\mathsf{v(g)} = \mathsf{v(y)} + \mathsf{v(z)})
\end{array}
$$

Figure 2. One system representation of the circuit of Figure 1.

where Δ is the set of abnormal components in the state: Δ implicitly represents a global health state and Δ often denotes abusively the corresponding health state.

An *observation* is a piece of information resulting from the measurement of the system. In the logical framework, an observation is then represented as a logical fact. Usually, there are several measurement points in the system that induce a set of logical facts at a given time. In this framework, we call observation the conjunction of these logical facts and is commonly denoted OBS. Recalling the example of Figure 1, if the inputs $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}$ and the outputs f, g are observable, the conjunction

$$\mathsf{v(a)} = 1 \wedge \mathsf{v(b)} = 2 \wedge \mathsf{v(c)} = 3 \wedge \mathsf{v(d)} = 4 \wedge \mathsf{v(e)} = 5$$
$$\wedge \mathsf{v(f)} = 11 \wedge \mathsf{v(g)} = 23$$

is one observation of the system. Moreover, it can easily be noticed that this observation is *consistent* with our expectation about what the system is supposed to do. The consistency-based diagnosis problem is finally defined as follows.

Definition 2.2 (Diagnostic problem) *A diagnostic problem* DP *is a 3-uple* $\mathsf{DP} \stackrel{\mathbf{def}}{=} (\mathsf{SD}, \mathsf{COMPS}, \mathsf{OBS})$.

The *solution* of the problem (also called the *diagnosis*) is the set of diagnostic *candidates*.

Definition 2.3 (Candidate) *Let* $\mathsf{DP} = (\mathsf{SD}, \mathsf{COMPS}, \mathsf{OBS})$*, a* candidate *of* DP *is a health state* $\sigma(\Delta)$ *such that* $\mathsf{SD} \wedge \mathsf{OBS} \wedge \sigma(\Delta)$ *is satisfiable.*

By definition, if $\sigma(\Delta)$ is a candidate of DP then any heath state $\sigma(\Delta')$ with $\Delta \subset \Delta'$ is a candidate. Indeed, for any component $\mathsf{c} \in \Delta' \setminus \Delta$, the sentences $\neg\mathsf{Ab(c)} \Rightarrow \ldots$ are definitively satisfiable independently of OBS so if $\mathsf{SD} \wedge \mathsf{OBS} \wedge \sigma(\Delta)$ is satisfiable, $\mathsf{SD} \wedge \mathsf{OBS} \wedge \sigma(\Delta')$ certainly is. From this, it follows that solving the DP problem consists in determining the set of *minimal candidates*.

Definition 2.4 (Minimal Candidate) *A candidate* $\sigma(\Delta)$ *is minimal if there does not exist another candidate* $\sigma(\Delta')$ *such that* $\Delta' \subset \Delta$.

3 DITO: A CSP-BASED ENGINE

The objective of DITO is to propose to solve generic diagnosis problems by the use of an off-the-shelf CSP solver that benefits of the most recent advances in the constraint programming literature [2], [3]. As for any algorithms solving DP problems proposed in the literature, there is a need of an efficient satisfiability solver that is successively called to check whether a health state is a candidate [10, 6, 8, 9, 7].

The DITO engine implements, for the moment, a conflict-driven algorithm as introduced in [6]. This algorithm relies on the encoding of sub-problems as a CSP.

3.1 Constraint Satisfaction Problem

Definition 3.1 *A* constraint satisfaction problem *(CSP for short) is a 3-uple* (X, D, C) *where:*

- $X = \{x_1, \ldots, x_n\}$ *is a set of* variables.
- $D = \{Dom(x_1), \ldots, Dom(x_n)\}$ *is a set of* domains. $Dom(x_i)$ *is the domain of the variable* x_i.
- $C = \{c_1, \ldots, c_m\}$ *is a set of constraints. A constraint* c_i *is a subset of* $Dom(x_1) \times \cdots \times Dom(x_n)$.

An assignment $(v_1, \ldots, v_n)$ is a n-uple that belongs to $Dom(x_1) \times \cdots \times Dom(x_n)$, any value v_i being assigned to any variable x_i.

Definition 3.2 (consistent assignment) *An assignment* A *is* consistent *if*

$$A \in \bigcap_{i=1}^{m} c_i.$$

A consistent assignment is a solution of the CSP. Let $Sol(CSP)$ denote the set of solutions of the CSP. The CSP is then unsatisfiable iff

$$Sol(CSP) = \emptyset$$

that is there does not exist any consistent assignment of the CSP.

Any CSP solver proposes a language to specify (X, D, C) with a set of operators to define the expressions and the constraints over these expressions. Whatever the CSP solver that is in use, its language can express the following constraint operators: $\langle\text{and}\rangle$, $\langle\text{or}\rangle$, $\langle\text{not}\rangle$, $\langle\text{eq}\rangle$, $\langle\text{neq}\rangle$, $\langle\text{lt}\rangle$, $\langle\text{gt}\rangle$, $\langle\text{leq}\rangle$, $\langle\text{geq}\rangle$; and the following arithmetic operators: $\langle\text{plus}\rangle$, $\langle\text{minus}\rangle$, $\langle\text{mult}\rangle$, $\langle\text{div}\rangle$, $\langle\text{mod}\rangle$, $\langle\text{neg}\rangle$. This is this set of operators that DITO uses to encode the problems.

3.2 Logical sentence encoding

To solve any diagnostic problem, DITO *encodes* sub-problems in the CSP framework, by the use of the function $encode$:

$$encode : \mathcal{L} \rightarrow \texttt{constraints}$$

where $\mathcal{L}$ is the set of finite sentences of first-order logic and `constraints` is the set of constraints that can be expressed within the CSP framework.

To define $encode$, consider first a sentence ℓ of $\mathcal{L}$. The resulting CSP of ℓ is denoted (X_ℓ, D_ℓ, C_ℓ). The sentence ℓ may contain arithmetic notations (as any sentence in Figure 2). Any arithmetic variable v of ℓ is represented by a variable $x_v \in X_\ell^{var}$ associated with

$Dom(x_v) \in D_\ell$ which encodes the domain of v on x_v. Any predicate $\mathsf{P(t)}$ where t is a closed logical term[2] is represented by a CSP variable $x_{p_t} \in X_\ell^{pred}$ with $Dom(x_{p_t}) = \{0,1\} \in D_\ell$. Finally, $X_\ell = X_\ell^{var} \oplus X_\ell^{pred}$.

The function *encode* is recursively defined as follows.

1. $encode(\ell) = \langle \text{and} \rangle(encode(\ell_1), encode(\ell_2))$ if $\ell = \ell_1 \wedge \ell_2$ (conjunction).
2. $encode(\ell) = \langle \text{or} \rangle(encode(\ell_1), encode(\ell_2))$ if $\ell = \ell_1 \vee \ell_2$ (disjunction).
3. $encode(\neg\ell) = \langle \text{not} \rangle(encode(\ell))$ (negation).
4. $encode(P(A, B, C, \dots)) = \langle \text{eq} \rangle(p_{ABC\dots}, 1)$ with $p_{ABC\dots} \in X_\ell^{pred}$ (predicate).
5. $encode(\forall y, f(y)) = \langle \text{and} \rangle(encode(f(Y_1)), encode(f(Y_2)), ...)$ with $Y_i \in \bigcup_{x \in X_\ell^{var}} Dom(x)$ (universal quantification).
6. $encode(\exists y, f(y)) = \langle \text{or} \rangle(encode(f(Y_1)), encode(f(Y_2)), ...)$ with $Y_i \in \bigcup_{x \in X_\ell^{var}} Dom(x)$ (existential quantification).
7. $encode(t1\ op\ t2) = \langle \text{op} \rangle(encTerm(t1), encTerm(t2))$ with $(op, \langle \text{op} \rangle) \in \{(=, \langle \text{eq} \rangle), (\neq, \langle \text{neq} \rangle), (<, \langle \text{lt} \rangle), (>, \langle \text{gt} \rangle), (\leq, \langle \text{leq} \rangle), (\geq, \langle \text{geq} \rangle)\}$ (arithmetic constraints).

The function $encTerm$:

$$\mathcal{T} \rightarrow \texttt{expression}$$

encodes any logical arithmetical term into a CSP solver expression.

1. $encTerm(t1\ op\ t2) = \langle \text{op} \rangle(encTerm(t1), encTerm(t2))$ with $(op, \langle \text{op} \rangle) \in \{(+, \langle \text{plus} \rangle), (-, \langle \text{minus} \rangle), (*, \langle \text{mult} \rangle), (/, \langle \text{div} \rangle), (\%, \langle \text{mod} \rangle)$ (binary operators).
2. $encTerm(-t) = \langle \text{neg} \rangle(encTerm(t))$ (unary operator).
3. $encTerm(v) = v$ with $v \in X_\ell^{var}$ and $Dom(v) = \{V^-, V^+\}$ (variable).
4. $encTerm(C) = C$ (integer).

By definition of the *encode* function, checking the satisfiability of any sentence ℓ of $\mathcal{L}$ is equivalent to checking for the existence of a consistent assignment of the CSP $encode(\ell)$.

Property 1 *For any finite sentence $\ell \in \mathcal{L}$,*

$$\ell \text{ is satisfiable } \equiv Sol(encode(\ell)) \neq \emptyset.$$

3.3 DITO Algorithm

This algorithm aims at computing the set of *minimal candidates* of a consistency-based diagnosis problem. DITO implements a conflict-driven algorithm by coupling a set enumeration tree search with successive CSPs to look for *minimal conflicts* and, as a second stage, by coupling another set enumeration tree search with successive CSPs to look for *minimal hitting sets*.

3.3.1 Conflict-driven algorithm: background

Before going into the details of DITO algorithm, let first recall the basic principle of a conflict-driven algorithm.

[2] ℓ is a logical sentence so any term variable is either universally or existentially quantified.

Definition 3.3 (conflict) *Let $\Delta, \Delta' \subseteq \mathsf{COMPS}$, a* conflict $\mathsf{cf}(\Delta, \Delta')$ *of* DP *is a disjunction of* $\mathsf{Ab}()$ *literals:*

$$\mathsf{cf}(\Delta, \Delta') \stackrel{\mathbf{def}}{=} \bigvee_{c \in \Delta} \neg\mathsf{Ab(c)} \vee \bigvee_{c \in \Delta'} \mathsf{Ab(c)}$$

such that:

$$\mathsf{SD} \wedge \mathsf{OBS} \models \mathsf{cf}(\Delta, \Delta').$$

A positive conflict $\mathsf{pcf}(\Delta)$ *is a conflict such that* $\mathsf{pcf}(\Delta) \stackrel{\mathbf{def}}{=} \mathsf{cf}(\Delta, \emptyset)$.

In other words, for a given system description SD and a given observation OBS, a conflict is a necessary condition about the health of the components to achieve consistency in $\mathsf{SD} \wedge \mathsf{OBS}$. In particular, in the case of positive conflict, it asserts that at least one component of the conflict is necessary abnormal.

Definition 3.4 (minimal conflict) *A* conflict $\mathsf{cf}(\Delta, \Delta')$ *of* DP *is* minimal *if there does not exist $\Delta_1 \subseteq \Delta$ and $\Delta_2 \subseteq \Delta'$ with $\Delta_1 \cup \Delta_2 \subset \Delta \cup \Delta'$ such that $\mathsf{cf}(\Delta_1, \Delta_2)$ is a conflict.*

For a given problem DP, MCS^+ denotes the set of minimal positive conflicts. Any conflict-driven algorithm then relies on the following result:

Theorem 1 *A health state $\sigma(\Delta)$ is a minimal candidate of* DP *iff:*

$$\sigma(\Delta) \models \bigwedge_{\mathsf{cf} \in MCS^+} \mathsf{cf}$$

and there is no $\Delta' \subset \Delta$ such that $\sigma(\Delta') \models \bigwedge_{\mathsf{cf} \in MCS^+} \mathsf{cf}$

3.3.2 Search for minimal candidates

Theorem 1 leads the way to develop a conflict-driven algorithm to determine the set of minimal candidates. As $\sigma(\Delta) \models \bigwedge_{\mathsf{cf} \in MCS^+} \mathsf{cf}$ then, for each conflict cf of MCS^+, there must exist in Δ a component involved in cf (i.e. Δ is a hitting set of MCS^+). Moreover as there is no $\Delta' \subset \Delta$ such that $\sigma(\Delta') \models \bigwedge_{\mathsf{cf} \in MCS^+} \mathsf{cf}$, it means that this Δ is a minimal hitting set.

Algorithms 1 and 2 describe the conflict-driven algorithm implemented in DITO. Algorithm 1 simply describes the two stages: the computation of the minimal positive conflict set and the computation of the resulting minimal hitting set. Both computations rely on CSPs and on the same set enumeration tree search: this search is described in Algorithm 2.

Algorithm 1 Algorithm to search for minimal candidates

input: DP
$MCS^+ \leftarrow TreeSearch(encode(\mathsf{DP}), \mathsf{COMPS})$
$MHS \leftarrow TreeSearch(encode(MCS^+), \bigcup_{C \in MCS^+}(C))$
return $\{\sigma(\Delta) : \Delta \in MHS\}$

The set enumeration tree search implemented in DITO was proposed in [12]. The basic principle is to enumerate the subsets S of COMPS and check that:

1. $\mathsf{SD} \wedge \mathsf{OBS} \wedge \neg\left(\bigvee_{c \in S} \mathsf{Ab(c)}\right)$ is not satisfiable (so $\bigvee_{c \in S} \mathsf{Ab(c)}$ is a positive conflict);
2. $\bigwedge_{\mathsf{cf} \in MCS^+} \mathsf{cf} \wedge \neg\left(\bigvee_{c \in S} \mathsf{Ab(c)}\right)$ is not satisfiable (so S is a hitting set of MCS^+).

The enumeration strategy proposed in [12] determines the minimal subsets S that make the initial CSP (either DP or MCS^+) non-satisfiable. The enumeration is a breadth-first search from the singleton sets, the sets of size 2, etc; it is implemented with the help of a queue $\mathcal{Q}$. Each node of this queue is a 3-uple (S, c, Ch) where S is the current selected set of components, c is the last selected component and Ch is an ordered set of components that are not selected yet. For each visited node (S_i, c_{pi}, Ch_i), the algorithm tests the satisfiability of the problem $CSP \cup encode(\neg S_i)$. If the problem is not satisfiable S_i is then part of the final result. In order to improve the efficiency, the algorithm takes into account the fact that any superset $S' \supset S_i$ of any minimal non satisfiable subset S_i is not part of the solution. So if the visited node (S_i, c_{pi}, Ch_i) is satisfiable, the set of components in Ch_i that would lead to the selection of such a superset S' are pruned.

Algorithm 2 Set enumeration tree search.

Function TreeSearch
input 1: CSP
input 2: C
$\mathcal{N} \leftarrow \varnothing$
if $Sol(CSP) \neq \varnothing$ **then**
  $\mathcal{Q}$: queue
  $push((\varnothing, \varnothing, C), \mathcal{Q})$
  while $\mathcal{Q} \neq \varnothing$ **do**
    $(S, c_p, \{c_{q1}, \ldots, c_{qk}\}) \leftarrow pop(\mathcal{Q})$
    for i **from** k **to** 1 **do**
      $(S_i, c_{qi}, Ch_i) \leftarrow (S \cup \{c_{qi}\}, c_{qi}, \{c_{qi+1}, \ldots, c_{qk}\})$
      $CSP' \leftarrow CSP \cup encode(\neg S_i)$
      if $Sol(CSP') = \varnothing$ **then**
        $\mathcal{N} \leftarrow \mathcal{N} \cup \{(S_i, c_{qi}, Ch_i)\}$
      else
        for all $(S_\mathcal{N}, c_\mathcal{N}, Ch_\mathcal{N}) \in \mathcal{N}$ **do**
          if $c_\mathcal{N} \in \{c_{qi+1}, \ldots, c_{qk}\}$ **then**
            if $Ch_\mathcal{N} \subseteq \{c_{qi}, \ldots, c_{qk}\}$ **then**
              $Ch_i \leftarrow Ch_i \setminus \{c_\mathcal{N}\}$
            end if
          end if
        end for
        $push((S_i, c_{qi}, Ch_i), \mathcal{Q})$
      end if
    end for
  end while
end if
return $\{S : (S, c, Ch) \in \mathcal{N}\}$

Theorem 2 *Algorithm 1 is correct.*

Proof The algorithm relies on Theorem 1. Satisfiability is implemented as a CSP by the function *encode* (see Property 1) and Algorithm 2 returns, by construction (see the proof in [12]) the minimal sets of components such that a given CSP (either a diagnosis problem DP or a minimal positive conflict set MCS^+) is not satisfiable. ∎

3.3.3 Example

Going back to the problem defined by Figures 1 and 2, let suppose that

$$\mathsf{OBS} \equiv \mathsf{v(a)} = 1 \wedge \mathsf{v(b)} = 2 \wedge \mathsf{v(c)} = 3$$
$$\wedge \mathsf{v(d)} = 4 \wedge \mathsf{v(e)} = 5 \wedge \mathsf{v(f)} = 11 \wedge \mathsf{v(g)} = 22.$$

DITO arbitrarily chooses an order for the set of components that is $\{\mathsf{A1} < \mathsf{A2} < \mathsf{M1} < \mathsf{M2} < \mathsf{M3}\}$. The enumeration to get the conflicts proceeds as follows: $\varnothing, \mathsf{M3}, \mathsf{M2}, \ldots, \mathsf{A1}, \mathsf{M2M3}, \mathsf{M1M3}, \mathsf{M1M2}, \mathsf{A2M3}, \ldots$. For instance M2M3 is visited first as a set of two components because M3 is the maximal component and M2 is the maximal component that is lower than M3. M1M3 is visited second as the only component greater than M2 is M3 so the next selected component is M1 associated with the maximal component M3, etc. Following this search, DITO detects two minimal conflicts, namely:

$$\{\mathsf{A2}, \mathsf{M2}, \mathsf{M3}\} \text{ and } \{\mathsf{A1}, \mathsf{A2}, \mathsf{M1}, \mathsf{M3}\}.$$

From these sets, with the use of the same tree-search, we enumerate the components to look for minimal hitting sets and the minimal candidates of the problem follow:

$$\sigma(\mathsf{M3}), \sigma(\mathsf{A2}), \sigma(\mathsf{M1}, \mathsf{M2}), \sigma(\mathsf{A1}, \mathsf{A2})$$

DITO solves this problem in 523ms and uses 9MB.

4 SEARCH STRATEGY FOR MINIMAL DIAGNOSES

Intrinsically, the complexity of Algorithm 1 is in $O(2^n)$ where n is the size of COMPS which makes this tool useless if we try to naively solve a DP problem with a realistic set of components COMPS. We propose in this section a strategy to cope with this issue. As stated above, SD describes the nominal behaviour of the underlying system. COMPS characterises on one hand the set of components of the system but also on the other hand the granularity of the diagnosis result. Suppose, for instance, that the underlying system is composed of only one component $|\mathsf{COMPS}_1| = 1$ (the system itself), the corresponding $\mathsf{DP}_1 = (\mathsf{SD}_1, \mathsf{OBS}, \mathsf{COMPS}_1)$ is a detection problem (is the system abnormal or not?). Now suppose that the *same* system than above can be partitioned into two components $c_1, c_2 \in \mathsf{COMPS}_2$, then it corresponds to another diagnosis problem $\mathsf{DP}_2 = (\mathsf{SD}_2, \mathsf{OBS}, \mathsf{COMPS}_2)$. DP_2 is an isolation problem in the sense that it can detect whether c_1 is abnormal or c_2 is abnormal. DP_1 and DP_2 are logically different problems but they are defined on the same system and the same observation. Only the precision has changed. The main difference is that DP_1 is simpler to solve than DP_2 as $|\mathsf{COMPS}_1| < |\mathsf{COMPS}_2|$.

Let $\Pi \in 2^{\mathsf{COMPS}}$ be a partition of COMPS. We design SD_Π as follows. We first copy SD into SD_Π. Any predicate $\mathsf{Ab(c)}$ in SD_Π is then renamed $\mathsf{AbComp(c)}$ and for any subset S of components in the partition Π, we add the sentence $\neg\mathsf{Ab(S)} \Rightarrow \bigwedge_{c \in S} \neg\mathsf{AbComp(c)}$. Given a partition Π, we can then define a new diagnosis problem $\mathsf{DP}_\Pi = (\mathsf{SD}_\Pi, \mathsf{OBS}, \Pi)$. Let $\Delta \subseteq \mathsf{COMPS}$, we define $\mathcal{R}_\Pi(\Delta) \stackrel{\text{def}}{=} \{S \in \Pi : S \cap \Delta \neq \varnothing\}$.

Theorem 3 *For any minimal candidate $\sigma(\Delta_\Pi)$ of DP_Π there exists a non-empty set of minimal candidates $\sigma(\Delta)$ of DP such that $\Delta_\Pi = \mathcal{R}_\Pi(\Delta)$.*

Proof $\sigma(\Delta_\Pi)$ is a candidate of DP_Π so $\mathsf{SD}_\Pi \wedge \mathsf{OBS} \wedge \bigwedge_{S \in \Delta_\Pi} \mathsf{Ab(S)} \wedge \bigwedge_{S \notin \Delta_\Pi} \neg\mathsf{Ab(S)}$ is satisfiable. By construction of SD_Π, $\mathsf{SD}_\Pi \wedge \mathsf{OBS} \wedge (\bigwedge_{S \in \Delta_\Pi} \bigvee_{c \in S} \mathsf{Abcomp(c)}) \wedge (\bigwedge_{S \notin \Delta_\Pi} \bigwedge_{c \in S} \neg\mathsf{Abcomp(c)})$ is also satisfiable which leads to the satisfiability of $\mathsf{SD} \wedge \mathsf{OBS} \wedge (\bigwedge_{S \in \Delta_\Pi} \bigvee_{c \in S} \mathsf{Ab(c)}) \wedge (\bigwedge_{S \notin \Delta_\Pi} \bigwedge_{c \in S} \neg\mathsf{Ab(c)})$. By construction of SD, $\sigma(\Delta)$ with $\Delta =$

$\{c \in S : S \in \Delta_\Pi\}$ (i.e. any components in Δ_Π are said to be abnormal) is a candidate of DP so there must exist $\Delta_{min} \subseteq \Delta$ such that $\sigma(\Delta_{min})$ is a minimal candidate of DP.

As $\sigma(\Delta_\Pi)$ is minimal, for any $\Delta'_\Pi \subset \Delta_\Pi$, $\mathsf{SD}_\Pi \wedge \mathsf{OBS} \wedge \bigwedge_{S \in \Delta'_\Pi} \mathsf{Ab(S)} \wedge \bigwedge_{S \notin \Delta'_\Pi} \neg\mathsf{Ab(S)}$ is not satisfiable therefore $\mathsf{SD}_\Pi \wedge \mathsf{OBS} \wedge \bigwedge_{S \notin \Delta'_\Pi} \neg\mathsf{Ab(S)}$ is not satisfiable. It finally leads to the non-satisfiability of $\mathsf{SD} \wedge \mathsf{OBS} \wedge (\bigwedge_{S \notin \Delta'_\Pi} \bigwedge_{c \in S} \neg\mathsf{Ab(c)})$ so there cannot be any minimal candidate $\sigma(\Delta_{min})$ of DP such that $\mathsf{COMPS} \setminus \Delta_{min} \subseteq \{c \in S : S \notin \Delta'_\Pi\}$.

Finally, any minimal candidate Δ_{min} is such that $\Delta_{min} \subseteq \{c \in S : S \in \Delta_\Pi\}$ and it must contain at least a component $c \in S \in \Delta'_\Pi$ for any $\Delta'_\Pi \subset \Delta_\Pi$, so Δ_{min} is necessarily such that $\Delta_\Pi = \mathcal{R}_\Pi(\Delta_{min})$. ∎

Corollary 4 *Let $\sigma(\Delta^1_\Pi), \ldots, \sigma(\Delta^n_\Pi)$ be the set of minimal candidates of DP_Π, the set of minimal candidates of DP is the union of the minimal candidates $\sigma(\Delta)$ of DP such that $\Delta^i_\Pi = \mathcal{R}_\Pi(\Delta)$ for any $i \in \{1, \ldots, n\}$.*

Proof If a minimal candidate $\sigma(\Delta)$ of DP is not part of this union then $\mathsf{SD} \wedge \mathsf{OBS} \wedge \bigwedge_{c \in \Delta} \mathsf{Ab(c)} \wedge \bigwedge_{c \notin \Delta} \neg\mathsf{Ab(c)}$ is satisfiable and there exists $\Delta_\Pi = \mathcal{R}_\Pi(\Delta)$ a candidate of DP_Π such that Δ_Π is different from any of the Δ^i_Π. As $\sigma(\Delta^1_\Pi), \ldots, \sigma(\Delta^n_\Pi)$ is the set of minimal candidates of DP_Π, Δ_Π must then strictly include at least one $\sigma(\Delta^i_\Pi)$ which is contradictory as it would mean Δ is not minimal, since there is at least one minimal candidate Δ_i such that $\Delta^i_\Pi = \mathcal{R}_\Pi(\Delta_i)$. ∎

Theorem 3 and Corollary 4 offer a global set of strategies to search for minimal candidates that take into account that the complexity of Algorithm 1 is in $o(2^n)$. By aggregating the initial set of components into subsystems, we can define DP problems that can drastically minimize the enumeration for the conflict search. Moreover, for each DP solved, even if the DP is not the most precise one, the result of the DP is correct in the sense that it provides a correct piece of information about *all* minimal candidates and not just for a few of them. We will illustrate this point in the section below.

5 DIAGNOSTIC SESSIONS ON A C6288 CIRCUIT

This section aims at illustrating the use of DITO on more realistic problems than the one of Figure 1. We call *diagnostic sessions* a sequence of diagnosis problems $\mathsf{DP}_{\Pi_1}, \mathsf{DP}_{\Pi_2}, \ldots$ issued from the same initial diagnosis problem $\mathsf{DP} = (\mathsf{SD}, \mathsf{COMPS}, \mathsf{OBS})$. For this illustration, we use the C6288 circuit of the ISCAS85 benchmark.

5.1 DESCRIPTION OF THE C6288 CIRCUIT

The full details about this circuit are available in [4]. This circuit is a 16x16 multiplier that contains 2406 gates (mostly NOR-gates and AND-gates, and a few INV-gates). The input of this circuit consists of two buses (A and B), each bus being a vector of 16 bits. The output is a 32-bit bus P. The purpose of this circuit is to perform $\mathsf{P} = \mathsf{A} \times \mathsf{B}$. We choose this circuit as an example, as it is one of the largest available in the set of ISCAS85 benchmarks and its global function is rather elementary so extremely useful in practice. The circuit is implemented as a 15x16 matrix of 240 full and half adder modules (top half adder modules tham, bottom half adder modules bham, and full adder modules fam). In the following, this matrix will be denoted as a set of 15 columns $\mathsf{C1}, \ldots, \mathsf{C15}$. Each column $\mathsf{C_i}$ is composed of 16 adders denoted Ciadj. Each adder is composed of a set of gates, a gate will be denoted CiadjGk (the k^{th} gate of j^{th} adder in Ci). As far as the encoding of the diagnosis problems are concerned, the circuit c6288 has been encoded with 11904 variables and 11932 constraints between these variables. As opposed to the encoding of the problems associated to Figure 1, where there were arithmetic variables, the encoding of the c6288 circuit only contains boolean variables.

5.2 DIAGNOSTIC SESSION

We present here an interactive diagnostic session on the c6288 circuit. The diagnostic problem to solve is the following one. A circuit c6288 receives as a input on its A-bus the value 29449 and the value 17006 on its B-bus. The output of the circuit (P-bus) is 500809692. In this problem, we consider that the set of components COMPS are the 2406 gates of the circuit. The available observations are the A, B and P buses, which means that OBS will always denote the observation

$$\mathsf{OBS} \equiv (A = 29449 \wedge B = 17006 \wedge P = 500809692)$$

in the rest of this section. This problem has been selected by simulation. Indeed, we have randomly injected a faulty gate in the circuit among the 2406 gates present in the circuit. The selected gate is C1ad1G7, it is a NOR-gate that belongs to the top half adder module C1ad1 of the column C1. Notice that, as $29449 \times 17006 = 500809694 \neq 500809692$, the problem is obviously detectable. Hereafter, we present a diagnostic session with experimental results that is based on this problem and that relies on the strategy proposed in Section 4.

1. **Fault Detection Problem**. The first problem that is solved is the fault detection problem which consists in detecting whether the system behaves correctly or not. The first partition is therefore:

   $$\Pi_0 = \{S^0_0 = \{\mathsf{COMPS}\}\}.$$

   DITO returns the minimal diagnoses $\{\sigma(S^0_0)\}$ in 114670ms (114s) which states that there is definitely an inconsistency between the model and the observation.
2. **Column Isolation Problem 1**: the circuit c6288 is arranged as a set of 15 columns of adders (namely C1 to C15) and a column of 16 input and-gates (denoted C0). Partitioning into 16 components requires endless computations, therefore the first partition that is chosen is:

   $$\Pi_1 = \{S^0_1 = \{\mathsf{C0}\}, S^1_1 = \{\mathsf{C1}, \ldots, \mathsf{C8}\}, S^2_1 = \{\mathsf{C9}, \ldots, \mathsf{C15}\}\}$$

   which means that DITO needs to solve a 3-component problem. DITO returns the minimal diagnoses $\{\sigma(S^1_1)\}$ in 285408ms (285s).
3. **Column Isolation Problem 2**: the previous problem asserts that if there are minimal candidates, they only involve components in $\sigma(S^1_1)$, any components in $\mathsf{COMPS} \setminus S^1_1$ are necessarily considered as normal within the minimal candidates. The next step is to decompose $\sigma(S^1_1)$ in smaller parts to get:

   $$\Pi'_2 = \{S^0_2 = \{\mathsf{C1}, \ldots, \mathsf{C3}\},$$
   $$S^1_2 = \{\mathsf{C4}, \mathsf{C5}\}, S^2_2 = \{\mathsf{C6}, \ldots, \mathsf{C8}\}\}.$$

   The global partition of the system is actually $\Pi_2 = \Pi'_2 \oplus \{S^0_1 \cup S^2_1\}$ but we already know that $\{S^0_1 \cup S^2_1\}$ is not involved in any minimal candidates. Therefore DITO will solve the problem $(\mathsf{SD}_{\Pi'_2}, \mathsf{OBS}, \Pi'_2)$ where $\mathsf{SD}_{\Pi'_2} \equiv \mathsf{SD}_{\Pi_2} \wedge \neg\mathsf{Ab}(\mathsf{S}^0_1) \wedge \neg\mathsf{Ab}(\mathsf{S}^2_1)$. It finally returns $\{\sigma(S^0_2)\}$ in 131735ms (131s).

	Partition	Solution	Time(ms)
0	$\{S_0^0 = \{\mathsf{COMPS}\}\}$	$\sigma(S_0^0)$	114670
1	$\{S_1^0 = \{\mathsf{C0}\}, S_1^1 = \{\mathsf{C1}, \ldots, \mathsf{C8}\}, S_1^2 = \{\mathsf{C9}, \ldots, \mathsf{C15}\}\}$	$\sigma(S_1^1)$	285408
2	$\{S_2^0 = \{\mathsf{C1}, \ldots, \mathsf{C3}\}, S_2^1 = \{\mathsf{C4}, \mathsf{C5}\}, S_2^2 = \{\mathsf{C6}, \ldots, \mathsf{C8}\}$	$\sigma(S_2^0)$	131735
3	$\{S_3^0 = \{\mathsf{C1}\}, S_3^1 = \{\mathsf{C2}\}, S_3^2 = \{\mathsf{C3}\}\}$	$\sigma(S_3^0)$	56098
4	$\{S_4^0 = \{\mathsf{C1ad1}, \ldots, \mathsf{C1ad15}\}, S_4^1 = \{\mathsf{C1gi1}, \ldots, \mathsf{C1gi15}, \mathsf{C1gj0}\}\}$	$\sigma(S_4^0), \sigma(S_4^1)$	22850
5	$\{S_5^0 = \{\mathsf{C1ad1..8}, \mathsf{C1gi1..8}\}, S_5^1 = \{\mathsf{C1ad9..16}, \mathsf{C1gi9..16}\}$	$\sigma(S_5^0)$	57195
6	$\{S_6^0 = \{\mathsf{C1ad1..2}, \mathsf{C1gi1..2}\}, \ldots, S_6^3 = \{\mathsf{C1ad7..8}, \mathsf{C1gi7..8}\}\}$	$\sigma(S_6^0)$	33760
7	$\{S_7^0 = \{\mathsf{C1ad1}\}, S_7^1 = \{\mathsf{C1ad2}\}, S_7^2 = \{\mathsf{C1gi1}\}, S_7^3 = \{\mathsf{C1gi2}\}\}$	$\sigma(S_7^0)$, $\sigma(S_7^2)$	36861
8	$\{S_8^0 = \{\mathsf{C1ad1G1..4}\}, S_8^2 = \{\mathsf{C1ad1G5..9}\}\}$	$\sigma(S_8^0)$, $\sigma(S_8^1)$	17373
9	$\{S_9^0 = \{\mathsf{C1ad1G1}\}, \ldots S_9^3 = \{\mathsf{C1ad1G4}\}\}$	$\sigma(S_9^1), \sigma(S_9^2), \sigma(S_9^3)$	37498
10	$\{S_{10}^0 = \{\mathsf{C1ad1G5}\}, \ldots S_{10}^4 = \{\mathsf{C1ad1G9}\}\}$	$\sigma(S_{10}^1), \sigma(S_{10}^2), \sigma(S_{10}^3)$	66456

Table 1. Diagnostic session on a c6288 circuit.

4. **The following problems:** the next problems of this session follow exactly the same principles that are described here above. The result of each problem is presented on Table 1. At the problem 4, DITO returns two possible minimal candidates $\sigma(S_4^0), \sigma(S_4^1)$ which means that from this problem, we can derive two problems, one stating that $\neg\mathsf{Ab}(\mathsf{S}_4^0) \wedge \mathsf{Ab}(\mathsf{S}_4^1)$ and the other one stating that $\mathsf{Ab}(\mathsf{S}_4^0) \wedge \neg\mathsf{Ab}(\mathsf{S}_4^1)$. Table 1 does not present all the alternatives but only a few of them. However, based on the problem 8, problem 9 (based on $\mathsf{Ab}(\mathsf{S}_8^0) \wedge \neg\mathsf{Ab}(\mathsf{S}_8^1)$) and problem 10 (based on $\neg\mathsf{Ab}(\mathsf{S}_8^0) \wedge \mathsf{Ab}(\mathsf{S}_8^1)$) are solved in order to show all the minimal candidates for the initial candidates that involves the top half adder module C1ad1 where the fault was actually injected. These minimal candidates are $\sigma(S_9^1), \sigma(S_9^2), \sigma(S_9^3)$ and $\sigma(S_{10}^1), \sigma(S_{10}^2), \sigma(S_{10}^3)$. Finally, let us notice that the minimal candidate $\sigma(S_{10}^2)$ is actually the health state that was injected for this scenario.

This session fully details a diagnostic session based on the injection of one fault in the circuit c6288. Due to lack of space, we cannot detail the case where several faults are injected in the circuit. However the principles remain the same and the time results are similar for each problem to solve as long as the number of components is small. There is one difference that the presented scenario does not show properly: it is the case when for a given partition Π, the minimal candidates are such that $\sigma(S_{i1}^\Pi \wedge \cdots \wedge S_{ik}^\Pi)$. In this case, there are basically two options depending on the strategy of the user. If the user wants to investigate other parts of the system before detailing $S_{i1}^\Pi \cup \cdots \cup S_{ik}^\Pi$, the best way to do it is to make a new partition whose $S_{i1}^\Pi \cup \cdots \cup S_{ik}^\Pi$ is an element of it. In the other case, it is required to split $S_{i1}^\Pi \cup \cdots \cup S_{ik}^\Pi$ to always minimize the number of components in the problem to solve.

6 DISCUSSION

The purpose of DITO is to benefit of constraint programming to boost its performance without the need of more adhoc techniques like in [6, 8]. A preliminary version of DITO was actually embedded into the meta-diagnostic tool MEDITO [1]. This initial MEDITO diagnosis engine is now fully replaced by DITO. Experimental results show how this engine can be used on realistic cases based on a hierarchical view of the underlying system as in [8]. But the point of view is different and complementary. Instead of focusing on the search for specific minimal candidates (the most probable ones, the smallest ones) as in [6, 8, 7], the idea is to always provide a global overview of the system as a decision may not require the full precised solution. The first perspective of this work is obviously to automate the proposed strategy (automatic selection of the partitions) even if we believe that a user might be interested to interact with. The other challenge is to make DITO really be an anytime tool, that solves problem in a way that DITO *always* provides a *complete solution* which converge to the precise solution as time goes.

REFERENCES

[1] Nuno Belard, Yannick Pencolé, and Michel Combacau, 'Medito: a logic-based meta diagnosis tool', in *IEEE International Conference on Tools with Artificial Intelligence*, pp. 709–716, Boca Raton (FL), United States, (11 2011).

[2] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit, 'Global constraint catalogue: Past, present and future.', *Constraints*, **12**(1), 21–62, (2007).

[3] Nicolas Beldiceanu, Pierre Flener, and Xavier Lorca, 'Combining tree partitioning, precedence, and incomparability constraints', *Constraints*, **13**(4), 459–489, (2008).

[4] Iscas-85 c6288 16x16 multiplier. in http://web.eecs.umich.edu/ jhayes/iscas.restore/c6288.html.

[5] Randall Davis, 'Diagnostic reasoning based on structure and behavior', *Artificial Intelligence*, **24**, 347–410, (1984).

[6] Johan deKleer and Brian C Williams, 'Diagnosing multiple faults', *Artificial Intelligence*, **32**, 97–130, (1987).

[7] Alexander Feldman, Gregory Provan, and Arjan van Gemund, 'Approximate model-based diagnosis using greedy stochastic search', *Journal of Artificial Intelligence Research*, **38**, 371–413, (2010).

[8] Alexander Feldman and Arjan van Gemund, 'A two-step hierarchical algorithm for model-based diagnosis', in *The Twenty-First National Conference on Artificial Intelligence*, pp. 827–833, Boston, MA, United States, (2006).

[9] Karim Lunde, Rüdiger Lunde, and Burkhard Münker, 'Model-based failure analysis with rodon', in *ECAI 2006: 17th European Conference on Artificial Intelligence*, pp. 647–651, Riva del Garda, Trentino, Italy, (2006).

[10] Raymond Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **32**, 57–95, (4 1987).

[11] The Choco Team, 'choco: an open source java constraint programming library', in *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming*, Paris, France, (2010).

[12] Xiangfu Zhao and Dantong Ouyang, 'Improved algorithms for deriving all minimal conflict sets in model-based diagnosis', in *Advanced Intelligent Computing Theories and Applications. With Aspects of Theoretical and Methodological Issues. Third International Conference on Intelligent Computing*, pp. 157–166, Qingdao, China, (8 2007).