Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2012), *26*, 53–62.

doi:10.1017/S0890060411000011

An efficient diagnosis algorithm for inconsistent constraint sets

A. FELFERNIG, M. SCHUBERT, AND C. ZEHENTNER
Institute for Software Technology, Graz University of Technology, Graz, Austria

(RECEIVED May 26, 2010; ACCEPTED October 4, 2010)

Abstract

Constraint sets can become inconsistent in different contexts. For example, during a configuration session the set of customer requirements can become inconsistent with the configuration knowledge base. Another example is the engineering phase of a configuration knowledge base where the underlying constraints can become inconsistent with a set of test cases. In such situations we are in the need of techniques that support the identification of minimal sets of faulty constraints that have to be deleted in order to restore consistency. In this paper we introduce a divide and conquer-based diagnosis algorithm (FASTDIAG) that identifies minimal sets of faulty constraints in an overconstrained problem. This algorithm is specifically applicable in scenarios where the efficient identification of leading (preferred) diagnoses is crucial. We compare the performance of FASTDIAG with the conflict-directed calculation of hitting sets and present an in-depth performance analysis that shows the advantages of our approach.

Keywords: Inconsistent Constraint Sets; Interactive Configuration; Model-Based Diagnosis; Preferred Diagnosis

1. INTRODUCTION

Constraint technologies (Tsang, 1993) are applied in different areas such as configuration (Mittal & Frayman, 1989; Fleischanderl et al., 1998; Sinz & Haag, 2007), recommendation (Felfernig et al., 2009), and scheduling (Castillo et al., 2005). There are many scenarios where the underlying constraint sets can become overconstrained. For example, when implementing a configuration knowledge base, constraints can become inconsistent with a set of test cases (Felfernig et al., 2004). Alternatively, when interacting with a configurator application (O'Sullivan et al., 2007; Felfernig et al., 2009), the given set of customer requirements (represented as constraints) can become inconsistent with the configuration knowledge base. In both situations there is a need of an intelligent assistance that actively supports users of a constraint-based application (end users or knowledge engineers). A widespread approach to support users in the identification of minimal sets of faulty constraints is to combine conflict detection (e.g., see Junker, 2004) with a corresponding hitting set algorithm (DeKleer & Williams, 1987; Reiter, 1987; DeKleer et al., 1992). In their original form these algorithms are applied for the calculation of *minimal* (*cardinality*) *diagnoses* that are typically determined with breadth-first search. Further diagnosis algorithms have been developed that follow a best-first search regime where the expansion of the hitting set search tree is guided by failure probabilities of components (DeKleer, 1990). Another example for such an approach is presented in (Felfernig et al., 2009), where similarity metrics are used to guide the (best-first) search for a preferred (plausible) minimal diagnosis (including repairs).

Both simple breadth-first search and best-first search diagnosis approaches are predominantly relying on the calculation of conflict sets (Junker, 2004). In this context, the determination of a minimal diagnosis of cardinality n requires the identification of at least n minimal conflict sets. In this paper we introduce a diagnosis algorithm (FASTDIAG) that allows to determine *one minimal diagnosis at a time* with the same computational effort related to the calculation of *one conflict set at a time*. The algorithm supports the identification of preferred diagnoses given predefined preferences regarding a set of decision alternatives. FASTDIAG is boosting the applicability of diagnosis methods in scenarios such as online configuration and reconfiguration (Felfernig et al., 2004), recommendation of products and services (Felfernig et al., 2009), and (more generally) in scenarios where the efficient calculation of preferred (leading) diagnoses is crucial (DeKleer, 1990). FASTDIAG is not restricted to constraint-based systems but it

Reprint requests to: A. Felfernig, Institute for Software Technology, Graz University of Technology, Inffeldgasse 16b, A-8010 Graz, Austria. E-mail: alexander.felfernig@ist.tugraz.at

is also applicable, for example, in the context of SAT solving (Marques-Silva & Sakallah, 1996) and description logics reasoning (Friedrich & Shchekotykhin, 2005).

The remainder of this paper is organized as follows. In Section 2 we introduce a simple example configuration task from the automotive domain. In Section 3 we discuss the basic hitting set-based approach to the calculation of diagnoses. In Section 4 we introduce an algorithm (FastDiag) for calculating preferred diagnoses for a given overconstrained problem. In Section 5 we present a detailed evaluation of FastDiag, which clearly outperforms standard hitting set-based algorithms in the calculation of the *topmost-n* preferred diagnoses. With Section 6 we provide an overview of related work in the field. The paper is concluded with Section 7.

2. EXAMPLE DOMAIN: CAR CONFIGURATION

Car configuration will serve as a working example throughout this paper. Because we exploit configuration problems for the discussion of our diagnosis algorithm, we first introduce a formal definition of a configuration task. This definition is based on Felfernig et al. (2004) but is given in the context of a constraint satisfaction problem (CSP; Tsang, 1993).

Definition 1 (configuration task). A configuration task can be defined as a CSP (V, D, C). Here, $V = \{v_1, v_2, \ldots, v_n\}$ represents a set of finite domain variables; $D = \{\text{dom}(v_1), \text{dom}(v_2), \ldots, \text{dom}(v_n)\}$ represents a set of variable domains $\text{dom}(v_k)$, where $\text{dom}(v_k)$ represents the domain of variable v_k; $C = C_{KB} \cup C_R$, where $C_{KB} = \{c_1, c_2, \ldots, c_q\}$ is a set of domain specific constraints (the configuration knowledge base) that restrict the possible combinations of values assigned to the variables in V; and $C_R = \{c_{q+1}, c_{q+2}, \ldots, c_t\}$ is a set of customer requirements that is also represented as constraints. ∎

A simplified example of a configuration task in the automotive domain is the following. In this example, *type* represents the car type, *pdc* is the park distance control functionality, *fuel* represents the fuel consumption per 100 km, a *skibag* allows ski stowage inside the car, and *4-wheel* represents the corresponding actuation type. These variables describe the potential set of requirements that can be specified by the user (customer). The possible combinations of these requirements are defined by a set of constraints that are denoted as C_{KB}, which is defined as $C_{KB} = \{c_1, c_2, c_3, c_4\}$ in our example. Furthermore, we assume the set of *customer requirements* $C_R = \{c_5, c_6, c_7\}$.[1]

- $V = \{\textbf{\textit{type}}, \textbf{\textit{pdc}}, \textbf{\textit{fuel}}, \textbf{\textit{skibag}}, \textbf{\textit{4-wheel}}\}$
- $D = \{\text{dom}(\textbf{\textit{type}}) = \{city, limo, combi, xdrive\}, \text{dom}(\textbf{\textit{pdc}}) = \{yes, no\}, \text{dom}(\textbf{\textit{fuel}}) = \{4l, 6l, 10l\}, \text{dom}(\textbf{\textit{skibag}}) = \{yes, no\}, \text{dom}(\textbf{\textit{4-wheel}}) = \{yes, no\}\}$
- $C_{KB} = \{c_1\text{: } \textbf{\textit{4-wheel}} = yes \Rightarrow \textbf{\textit{type}} = xdrive, c_2\text{: } \textbf{\textit{skibag}} = yes \Rightarrow \textbf{\textit{type}} \neq city, c_3\text{: } \textbf{\textit{fuel}} = 4l \Rightarrow \textbf{\textit{type}} = city, c_4\text{: } \textbf{\textit{fuel}} = 6l \Rightarrow \textbf{\textit{type}} \neq xdrive\}$
- $C_R = \{c_5\text{: } \textbf{\textit{type}} = combi, c_6\text{: } \textbf{\textit{fuel}} = 4l, c_7\text{: } \textbf{\textit{4-wheel}} = \text{yes}\}$

[1] Note that constraints are not necessarily *unary* or *binary* (we tried to keep the example simple). They can also be *n-ary*.

On the basis of this configuration task definition, we can now introduce the definition of a concrete *configuration* (solution for a configuration task).

Definition 2 (configuration). A configuration for a given configuration task (V, D, C) is an instantiation $I = \{v_1 = \text{ins}_1, v_2 = \text{ins}_2, \ldots, v_n = \text{ins}_n\}$, where $\text{ins}_k \in \text{dom}(v_k)$. ∎

A configuration is *consistent* if the assignments in I are consistent with the $c_i \in C$. Furthermore, a configuration is *complete* if all variables in V are instantiated. Finally, a configuration is *valid* if it is consistent and complete.

3. DIAGNOSING OVERCONSTRAINED PROBLEMS

For the configuration task introduced in Section 2 we are *not* able to find a solution, for example, a *combi*-type car does not support a fuel consumption of *4l per 100 km*. Consequently, we want to identify minimal sets of constraints ($c_i \in C_R$) which have to be deleted in order to be able to identify a solution (restore the consistency). In the example of Section 2 the set of constraints $C_R = \{c_5, c_6, c_7\}$ is inconsistent with the constraints $C_{KB} = \{c_1, c_2, c_3, c_4\}$, that is, no solution can be found for the underlying configuration task. A standard approach to determine a minimal set of constraints that have to be deleted from an overconstrained problem is to resolve all minimal conflicts contained in the constraint set. The determination of such constraints is based on a conflict detection algorithm (e.g., see Junker, 2004), the derivation of the corresponding diagnoses is based on the calculation of hitting sets (Reiter, 1987). Because both the notion of a (*minimal*) *conflict* and the notion of a (*minimal*) *diagnosis* will be used in the following sections, we provide the corresponding definitions here.

Definition 3 (conflict set). A conflict set is a set $\text{CS} \subseteq C_R$ such that $C_{KB} \cup \text{CS}$ is inconsistent. CS is a *minimal* if there does not exist a conflict set CS′ with $\text{CS}' \subset \text{CS}$. ∎

In our working example we can identify three minimal conflict sets, which are $\text{CS}_1 = \{c_5, c_6\}$, $\text{CS}_2 = \{c_5, c_7\}$, and $\text{CS}_3 = \{c_6, c_7\}$.

CS_1, CS_2, and CS_3 are conflict sets because $\text{CS}_1 \cup C_{KB} \vee \text{CS}_2 \cup C_{KB} \vee \text{CS}_3 \cup C_{KB}$ is inconsistent. The minimality property is fulfilled because a conflict set CS_4 does not exist with $\text{CS}_4 \subset \text{CS}_1$ or $\text{CS}_4 \subset \text{CS}_2$ or $\text{CS}_4 \subset \text{CS}_3$. The standard approach to resolve the given conflicts is the construction of a corresponding *hitting set directed acyclic graph* (*HSDAG*; Reiter, 1987) where the resolution of all minimal conflict sets automatically corresponds to the identification of a minimal diagnosis. A minimal diagnosis in our application context is a minimal set of customer requirements contained in the set of car features (C_R) that has to be deleted from C_R

https://doi.org/10.1017/S0890060411000011 Published online by Cambridge University Press

in order to make the remaining constraints consistent with C_{KB}. Because we are dealing with the diagnosis of customer requirements, we introduce the definition of a *customer requirements diagnosis problem* (Definition 4). This definition is based on the definition given in Felfernig et al. (2004).

DEFINITION 4 (CR diagnosis problem). A customer requirements diagnosis (CR diagnosis) problem is defined as a tuple (C_{KB}, C_R), where C_R is the set of given customer requirements and C_{KB} represents the constraints part of the configuration knowledge base. ■

The definition of a *CR diagnosis* that corresponds to a given CR diagnosis problem is the following (see Definition 5).

DEFINITION 5 (CR diagnosis). A CR diagnosis for a CR diagnosis problem (C_{KB}, C_R) is a set $\Delta \subseteq C_R$, such that $C_{KB} \cup (C_R - \Delta)$ is consistent. Δ is *minimal* if there does not exist a diagnosis $\Delta' \subset \Delta$, such that $C_{KB} \cup (C_R - \Delta')$ is consistent. ■

The HSDAG algorithm for determining minimal diagnoses is discussed in detail in Reiter (1987). The concept of this algorithm will be explained on the basis of our working example. It relies on a conflict detection algorithm that is responsible for detecting minimal conflicts in a given set of constraints (in our case, in the given customer requirements). One conflict detection algorithm is QUICKXPLAIN (Junker, 2004), which is based on an efficient divide and conquer search strategy. For the purposes of our working example let us assume that the first minimal conflict set determined by QUICKXPLAIN is the set $CS_1 = \{c_5, c_6\}$. Because of the minimality property, we are able to resolve each conflict by simply deleting one element from the set, for example, in the case of CS_1 we have to either delete c_5 or c_6. Each variant to resolve a conflict set is represented by a specific path in the corresponding HSDAG. The HSDAG for our working example is depicted in Figure 1. The deletion of c_5 from CS_1 triggers the calculation of another conflict set $CS_3 = \{c_6, c_7\}$ because $C_R - \{c_5\} \cup C_{KB}$ is inconsistent. If we decide to delete c_6 from CS_1, $C_R - \{c_6\} \cup C_{KB}$ remains inconsistent, which means that QUICKXPLAIN returns another minimal conflict set, which is $CS_2 = \{c_5, c_7\}$.

The original HSDAG algorithm (Reiter, 1987) follows a strict breadth-first search regime. Following this strategy, the next node to be expanded in our working example is the minimal conflict set CS_3, which has been returned by

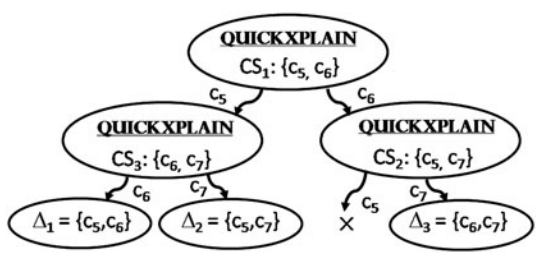

Fig. 1. Hitting set directed acyclic graph (HSDAG; Reiter, 1987) for the CR diagnosis problem ($C_R = \{c_5, c_6, c_7\}$, $C_{KB} = \{c_1, c_2, c_3, c_4\}$). The sets $\{c_5, c_6\}$, $\{c_6, c_7\}$, and $\{c_5, c_7\}$ are the minimal diagnoses; the conflict sets CS_1, CS_2, and CS_3 are determined on the basis of QUICKXPLAIN (Junker, 2004).

QUICKXPLAIN for $C_R - \{c_5\} \cup C_{KB}$. In this context, the first option to resolve CS_3 is to delete c_6. This option is a valid one and $\Delta_1 = \{c_5, c_6\}$ is the resulting minimal diagnosis. The second option for resolving CS_3 is to delete the constraint c_7. In this case, we have identified the next minimal diagnosis $\Delta_2 = \{c_5, c_7\}$ because $C_R - \{c_5, c_7\} \cup C_{KB}$ is consistent. This way we are able to identify all minimal sets of constraints Δ_i that, if deleted from C_R, help to restore the consistency with C_{KB}. If we want to calculate the complete set of diagnoses for our working example, we still have to resolve the conflict set CS_2. The first option to resolve CS_2 is to delete c_5, because $\{c_5, c_6\}$ has already been identified as a minimal diagnosis, we can close this node in the HSDAG. The second option to resolve CS_2 is to delete c_7. In this case we have determined the third minimal diagnosis, which is $\Delta_3 = \{c_6, c_7\}$.

In our working example we are able to enumerate all possible diagnoses that help to restore consistency. However, the calculation of all minimal diagnoses is expensive and thus in many cases not practicable for interactive settings. Because users are often interested in a reduced subset of all the potential diagnoses, alternative algorithms are needed that are capable of identifying preferred diagnoses (Reiter, 1987; DeKleer, 1990; Felfernig et al., 2009). Such approaches have already been developed (DeKleer, 1990; Felfernig et al., 2009); however, they are still based on the resolution of conflict sets, which is computationally expensive (see Section 5). Our idea presented in this paper is a diagnosis algorithm that helps to determine preferred diagnoses without the need of calculating the corresponding conflict sets. The basic properties of FASTDIAG will be discussed in Section 4.

4. CALCULATING PREFERRED DIAGNOSES WITH FASTDIAG

4.1. Preferred diagnoses

Users typically prefer to keep the important requirements and to change or delete (if needed) the less important ones (Junker, 2004). The major goal of (model-based) diagnosis tasks is to identify the preferred (leading) diagnoses, which are not necessarily minimal cardinality ones (DeKleer, 1990). For the characterization of a preferred diagnosis we will rely on the definition of a total ordering of the given set of constraints in C (respectively C_R). Such a total ordering can be achieved, for example, by *directly asking* the customer regarding the preferences, by applying *multiattribute utility theory* (Winterfeldt & Edwards, 1986; Ardissono et al., 2003) where the determined interest dimensions correspond with the attributes of C_R or by applying the rankings determined by *conjoint analysis* (Belanger, 2005). The following definition of a *lexicographical ordering* (Definition 6) is based on total orderings for constraints that has been applied in (Junker, 2004) for the determination of *preferred conflict sets*.

DEFINITION 6 (total lexicographical ordering). Given a total order $<$ on C, we enumerate the constraints in C in increasing < order $c_1..c_n$, starting with the *least important constraints* (i.e., $c_i < c_j \Rightarrow i < j$). We compare two subsets X and Y of C lexicographically:

$$X >_{\text{lex}} Y \quad \text{iff}$$

$$\exists k\text{: } c_k \in Y - X \quad \text{and}$$

$$X \bigcap \{c_{k+1}, \ldots, c_t\} = Y \bigcap \{c_{k+1}, \ldots, c_t\}.$$

Based on this definition of a lexicographical ordering, we can now introduce the definition of a *preferred diagnosis*. ■

Definition 7 (preferred diagnosis). A minimal diagnosis Δ for a given CR diagnosis problem (C_R, C_{KB}) is a preferred diagnosis for (C_R, C_{KB}) if and only if another minimal diagnosis Δ' with $\Delta' >_{\text{lex}} \Delta$. ■

In our working example we assumed the lexicographical ordering ($c_5 < c_6 < c_7$), that is, the most important customer requirement is c_7 (the 4-wheel functionality). If we assume that $X = \{c_5, c_7\}$ and $Y = \{c_6, c_7\}$ then $Y - X = \{c_6\}$ and $X \cap \{c_7\} = Y \cap \{c_7\}$. Intuitively, $\{c_5, c_7\}$ is a preferred diagnosis compared to $\{c_6, c_7\}$ because both diagnoses include c_7 but c_5 is less important than c_6. If we change the ordering to ($c_7 < c_6 < c_5$), FastDiag would then determine $\{c_6, c_7\}$ as the preferred minimal diagnosis.

4.2. FastDiag approach

For the following discussions we introduce the set AC, which is initially set to $C_{KB} \cup C_R$ [the union of customer requirements (C_R) and the configuration knowledge base (C_{KB})] and subsequently changed when the algorithm runs. The basic idea of the FastDiag algorithm (Algorithm 1) is the following.[2] In our example, the set of customer requirements $C_R = \{c_5, c_6, c_7\}$ includes at least one minimal diagnosis because C_{KB} is consistent and $C_{KB} \cup C_R$ is inconsistent. In the extreme case C_R itself represents the minimal diagnosis, which then means that *all* constraints in C_R are part of the diagnosis, that is, each $c_i \in C_R$ represents a singleton conflict. In our case C_R obviously does not represent a minimal diagnosis—the set of diagnoses in our working example is $\{\Delta_1 = \{c_5, c_6\}, \Delta_2 = \{c_5, c_7\}, \Delta_3 = \{c_6, c_7\}\}$ (see Section 3). The next step in Algorithm 1 is to divide the set of customer requirements $C_R = \{c_5, c_6, c_7\}$ into the two sets $C_1 = \{c_5\}$ and $C_2 = \{c_6, c_7\}$ and to check whether AC $-$ C_1 is already consistent. If this is the case, we can omit the set C_2 because at least one minimal diagnosis can already be identified in C_1. In our case, AC $-$ $\{c_5\}$ is inconsistent, which means that we have to consider further elements from C_2. Therefore, $C_2 = \{c_6, c_7\}$ is divided into the sets $\{c_6\}$ and $\{c_7\}$. In the next step we can check whether AC – ($C_1 \cup \{c_6\}$) is consistent—this is the case that means that we do not have to further take into account $\{c_7\}$ for determining the diagnosis. Because

[2] In Algorithm 1 we use the set C instead of C_R because the application of the algorithm is not restricted to inconsistent sets of customer requirements.

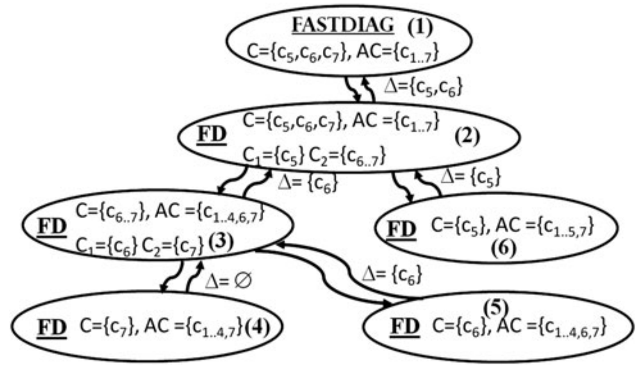

Fig. 2. FastDiag execution trace for the C_R diagnosis problem ($C_R = \{c_5, c_6, c_7\}$, $C_{KB} = \{c_1, c_2, c_3, c_4\}$). Enumerations 1–6 show the order in which the different incarnations of the FD function are activated.

$\{c_5\}$ does not include a diagnosis but $\{c_5\} \cup \{c_6\}$ includes a diagnosis, we can deduce that $\{c_6\}$ must be part of the diagnosis. The final step is to check whether AC – $\{c_6\}$ leads to a diagnosis without including $\{c_5\}$. We see that AC – $\{c_6\}$ is inconsistent, that is, $\Delta = \{c_5, c_6\}$ is a minimal diagnosis for the CR diagnosis problem ($C_R = \{c_5, c_6, c_7\}$, $C_{KB} = \{c_1, \ldots, c_4\}$). An execution trace of the FastDiag algorithm in the context of our working example is shown in Figure 2.

Algorithm 1: FastDiag

func FastDiag($C \subseteq AC$, $AC = \{c_1..c_t\}$) : *diagnosis* Δ
if *isEmpty*(C) *or inconsistent*(AC – C) *return* $\emptyset$
else *return* FD($\emptyset$, C, AC);
func FD(D, $C = \{c_1..c_q\}$, AC) : *diagnosis* Δ
if $D \neq \emptyset$ *and consistent*(AC) *return* $\emptyset$;
if *singleton*(C) *return* C;
$k = q/2$;
$C_1 = \{c_1..c_k\}$; $C_2 = \{c_{k+1}..c_q\}$;
$D_1 = FD(C_1, C_2, AC - C_1)$;
$D_2 = FD(D_1, C_1, AC - D_1)$;
return($D_1 \cup D_2$);

4.3. Calculating *n* > 1 diagnoses

In order to be able to calculate $n > 1$ diagnoses[3] with FastDiag we have to adopt the HSDAG construction introduced in Reiter (1987) by substituting the resolution of conflicts (see Fig. 1) with the deletion of elements c_i from C_R (C) (see Fig. 3). In this case, a path in the HSDAG is closed if no further diagnoses can be identified for this path or the elements of the current path are a superset of an already closed path (containment check). Conform to the HSDAG approach presented in Reiter (1987), we expand the search tree in a *breadth-first* manner. In our working example, we can delete $\{c_5\}$ (one element of the first diagnosis $\Delta_1 = \{c_5, c_6\}$) from the set C_R of diagnosable elements and restart the algorithm

[3] Typically a CR diagnosis problem has more than one related diagnosis.

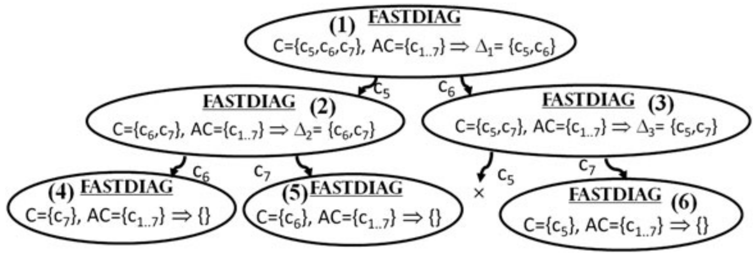

Fig. 3. FastDiag: calculating the complete set of *minimal diagnoses*. Enumerations 1–6 show the order in which the different incarnations of the FastDiag algorithm are activated.

for finding another minimal diagnosis for the CR diagnosis problem ($\{c_6, c_7\}$, C_{KB}). Because AC $-$ $\{c_5\}$ is inconsistent, we can conclude that $C_\text{R} = \{c_6, c_7\}$ includes another minimal diagnosis ($\Delta_2 = \{c_6, c_7\}$), which is determined by FastDiag for the CR diagnosis problem ($C_\text{R} - \{c_5\}$, C_{KB}). Finally, we have to check whether the CR diagnosis problem ($\{c_5, c_7\}$, C_{KB}) leads to another minimal diagnosis. This is the case, that is, we have identified the last minimal diagnosis that is $\Delta_3 = \{c_5, c_7\}$. The calculation of all diagnoses in our working example on the basis of FastDiag is depicted in Figure 3.

Note that for a given set of constraints (C) FastDiag always calculates the preferred diagnosis in terms of Definition 7. If Δ_1 is the diagnosis returned by FastDiag and we delete one element from Δ_1 (e.g., c_5), then FastDiag returns the preferred diagnosis for the CR diagnosis problem ($\{c_5, c_6, c_7\} - \{c_5\}$, $\{c_1, \ldots, c_7\}$), which is Δ_2 in our example case, that is, $\Delta_1 >_{\text{lex}} \Delta_2$. Consequently, diagnoses part of one path in the search tree (such as Δ_1 and Δ_2 in Fig. 3) are in a strict preference ordering. However, there is only a *partial order* between individual diagnoses in the search tree in the sense that a diagnosis at level k is not necessarily preferable to a diagnosis at level $k + 1$.

4.4. FastDiag properties

A detailed listing of the basic operations of FastDiag is shown in Algorithm 1. First, the algorithm checks whether the constraints in C contain a diagnosis, that is, whether AC $-$ C is consistent—the function assumes that it is activated in the case that AC is inconsistent. If AC $-$ C is inconsistent or C $= \emptyset$, FastDiag returns the empty set as result (no solution can be found, line 2 of the algorithm). If at least one diagnosis is contained in the set of constraints C, FastDiag activates the FastDiag function that is in charge of retrieving a preferred diagnosis (line 3 of the algorithm). FastDiag follows a divide and conquer strategy where the recursive function FastDiag divides the set of constraints (in our case the elements of C_R) into two different subsets (C_1 and C_2; line 8 of the algorithm) and tries to figure out whether C_1 already contains a diagnosis (line 5 of the algorithm). If this is the case, FastDiag does not further take into account the constraints in C_2. If only one element is remaining in the current set of constraints C and the current set of constraints in AC is still inconsistent, then the element in C is part of a minimal diagnosis (line 6 of the algorithm). FastDiag is *complete* in the sense that if C contains exactly one minimal diagnosis then FD will find it. If there are multiple minimal diagnoses then one of them (the preferred one, see Definition 7) is returned. The recursive function FD is triggered if AC $-$ C is consistent and C consists of at least one constraint. In such a situation a corresponding minimal diagnosis can be identified. If we assume the existence of a minimal diagnosis Δ that cannot be identified by FastDiag, this would mean that there exists at least one constraint c_a in C that is part of the diagnosis but not returned by FD. The only way in which elements can be deleted from C (i.e., not included in a diagnosis) is by the return $\emptyset$ statement in FD and $\emptyset$ is only returned in the case that AC is consistent, which means that the elements of C_2 (C_1) from the previous FD incarnation are not part of the preferred diagnosis. Consequently, it is not possible to delete elements from C, which are part of the diagnosis. FastDiag computes only *minimal diagnoses* in the sense of Definition 5. If we assume the existence of a nonminimal diagnosis Δ calculated by FastDiag, this would mean that there exists at least one constraint c_a with $\Delta - \{c_a\}$ is still a diagnosis. The only situation in which elements of C are added to a diagnosis Δ is if C itself contains exactly one element. If C contains only one element (let us assume c_a) and AC is inconsistent (in the function FastDiag) then c_a is the only element that can be deleted from AC, that is, c_a must be part of the diagnosis.

5. EVALUATION

5.1. Performance of FastDiag

In this section we will compare the performance of FastDiag with the performance of the hitting set algorithm (Reiter, 1987) in combination with the QuickXplain conflict detection algorithm introduced in (Junker, 2004).

The worst case complexity of FastDiag in terms of the number of consistency checks needed for calculating one minimal diagnosis is $2d \times \log_2(n/d) + 2d$, where d is the minimal diagnoses set size and n is the number of constraints (in C). The best case complexity is $\log_2(n/d) + 2d$. In the worst case each element of the diagnosis is contained in a different path of the search tree: $\log_2(n/d)$ is the depth of the path, $2d$ represents the branching factor and the number of leaf-node consistency checks. In the best case all elements of the diagnosis are contained in one path of the search tree.

The worst case complexity of QuickXplain in terms of consistency checks needed for calculating one minimal conflict set is $2k \cdot \log_2(n/k) + 2k$ where k is the minimal conflicts set size and n is again the number of constraints (in C; Junker, 2004). The best case complexity of QuickXplain in terms of the number of consistency checks needed is $\log_2(n/k) + 2k$ (Junker, 2004). Consequently, the number of consistency checks per conflict set (QuickXplain) and the number of consistency checks per diagnosis (FastDiag) fall into a logarithmic complexity class.

Let n_{cs} be the number of minimal conflict sets in a constraint set and n_{diag} be the number of minimal diagnoses, then we need n_{diag} FastDiag calls (see Algorithm 1) plus n_{cs} additional consistency checks and n_{cs} activations of QuickXplain with n_{diag} additional consistency checks for determining *all* diagnoses. The results of a performance evaluation of FastDiag are depicted in the Figure 4, Figure 5, Figure 6, and Figure 7. The basis for these evaluations was the bicycle configuration knowledge base taken from the CLib[4] (www.itu.dk/research/cla/externals/clib/) configuration benchmarks library (34 variables and about 65 constraints). For this example knowledge base we randomly generated different sets of requirements (of cardinality 5, 7, 10, and 15 requirements) and measured the performance of calculating corresponding diagnosis sets (the first diagnosis, first 5 diagnoses, first 10 diagnoses, and all diagnoses). The run time performance of the different diagnosis algorithms and the needed amount of TP calls is shown in the Figures 4–7. As solver we used the CLib-based decision diagram representation that allows for backtracking-free solution search. The tests have been executed on a standard desktop computer (Intel Core 2 Quad CPU QD9400 2.66-GHz CPU with 2 GB of RAM). Note that we have evaluated the performance of FastDiag with different other benchmark configuration knowledge bases on the CLib Web page with basically the same result. FastDiag shows to be a valuable alternative for determining diagnoses in interactive settings especially for calculating the preferred first-n solutions.

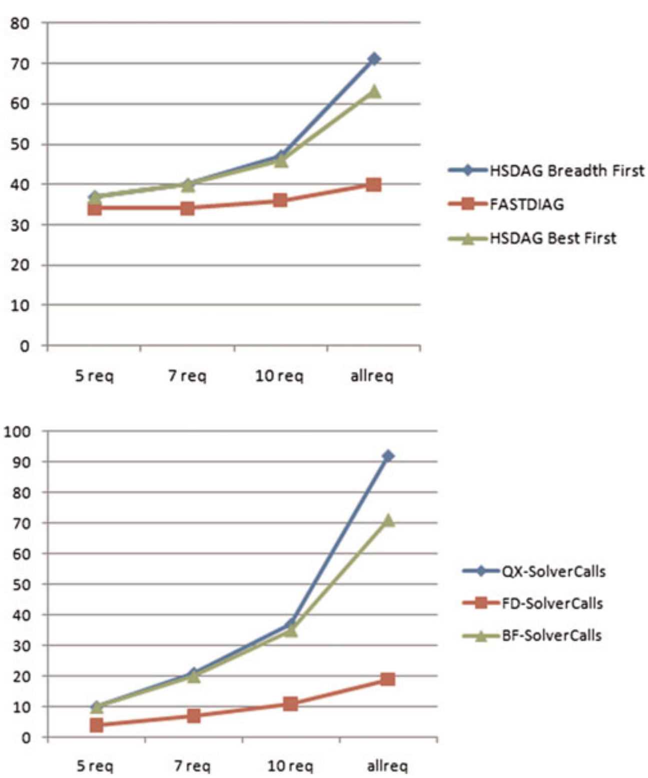

Fig. 4. Calculating the *first minimal diagnosis* with FastDiag versus hitting set-based diagnosis on the basis of QuickXplain for 5, 7, 10, and 15 user requirements (*req*): performance in *milliseconds* on the *top* and *number of needed TP calls* on the *bottom*. [A color version of this figure can be viewed online at journals.cambridge.org/aie]

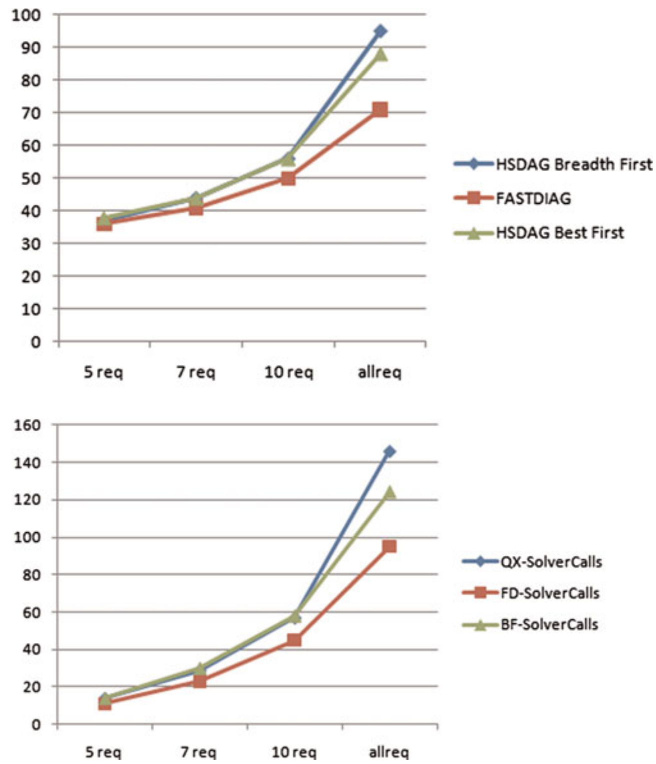

Fig. 5. Calculating the *topmost-5 minimal diagnoses* with FastDiag versus hitting set-based diagnosis on the basis of QuickXplain for 5, 7, 10, and 15 user requirements (*req*): performance in *milliseconds* on the *top* and *number of needed TP calls* on the *bottom*. [A color version of this figure can be viewed online at journals.cambridge.org/aie]

Figure 4 shows a comparison between the hitting set based diagnosis approach (denoted as HSDAG) and the FastDiag algorithm (denoted as FastDiag) in the case that only *one diagnosis* is calculated. FastDiag clearly outperforms the HSDAG approach independent of the way in which diagnoses are calculated (breadth-first or best-first). Figure 5 shows the performance evaluation for calculating the *topmost-5 minimal diagnoses*. The result is similar to the one for calculating the first diagnosis, that is, FastDiag outperforms the two HSDAG versions. Our evaluations show that FastDiag is very efficient in calculating preferred minimal diagnoses.

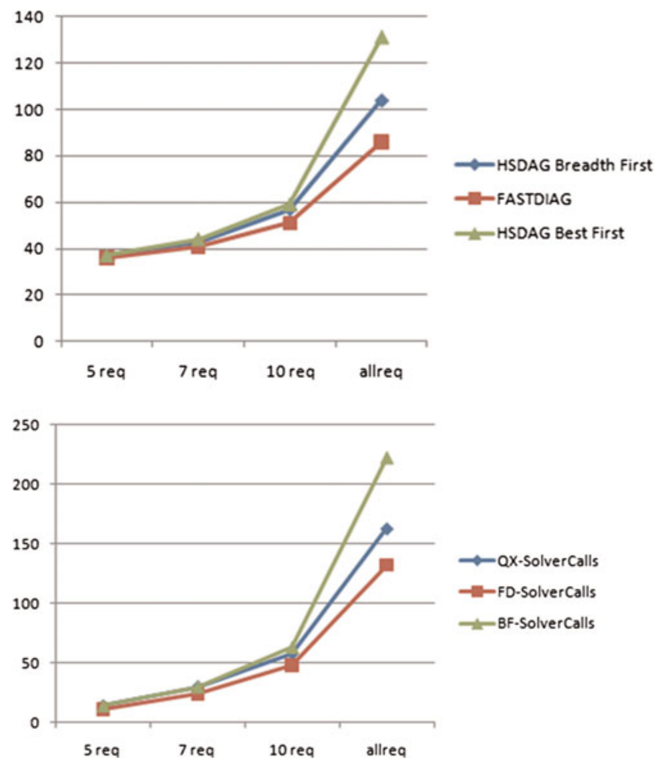

Fig. 6. Calculating the *topmost-10 minimal diagnoses* with FastDiag versus hitting set-based diagnosis on the basis of QuickXplain for 5, 7, 10, and 15 user requirements (*req*): performance in *milliseconds* on the *top* and *number of needed TP calls* on the *bottom*. [A color version of this figure can be viewed online at journals.cambridge.org/aie]

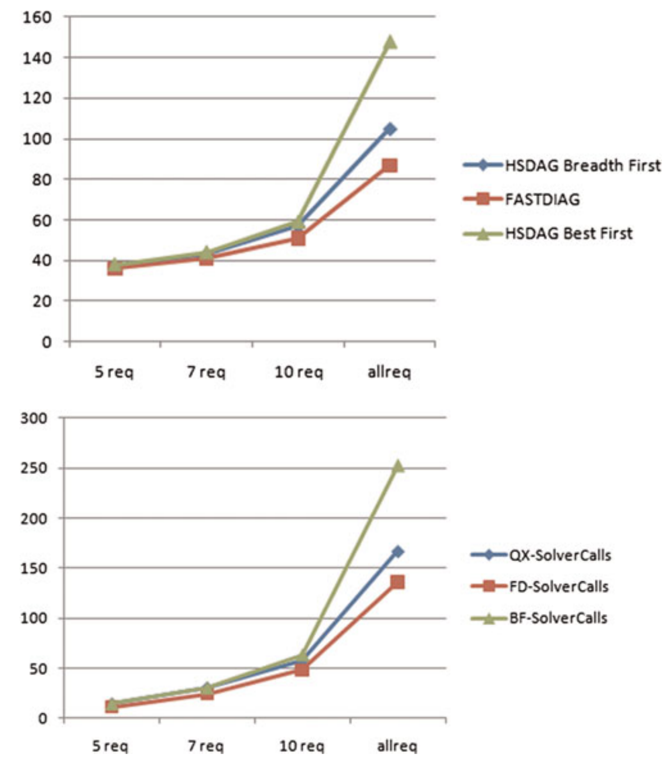

Fig. 7. Calculating *all minimal diagnoses* with FastDiag versus hitting set-based diagnosis on the basis of QuickXplain for 5, 7, 10, and 15 user requirements (*req*): performance in *milliseconds* on the *top* and *number of needed TP calls* on the *bottom*. [A color version of this figure can be viewed online at journals.cambridge.org/aie]

5.2. Empirical evaluation

Based on a computer configuration dataset of the Graz University of Technology ($N = 415$ configurations) we evaluated the three presented approaches with respect to their capability of predicting diagnoses that are acceptable for the user (diagnoses leading to selected configurations). Each entry of the dataset consists of a set of initial *user requirements* C_R inconsistent with the configuration knowledge base C_{KB} and the *configuration* that had been finally selected by the user. Because the original requirements stored in the dataset are inconsistent with the configuration knowledge base, we could determine those diagnoses that indicated which minimal sets of requirements have to be deleted in order to be able to find a solution.

We evaluated the *prediction accuracy* of the three diagnosis approaches (*breadth-first HSDAG*, FastDiag, and *best-first HSDAG*). We first measured the distance between the *predicted position* of a diagnosis leading to a selected configuration and the *expected position of the diagnosis* (which is 1). This distance was measured in terms of the *root mean square deviation* (RMSD; see Formula 1). The results of this first analysis are the following: an important result is that FastDiag has the lowest RMSD value (0.95), best-first HSDAG has a similar prediction quality (RMSD = 0.97), and breadth-first HSDAG has the worst prediction quality (RMSD = 1.64).

$$\text{RMSD} = \sqrt{\frac{1}{n}\sum_{1}^{n}(\text{predicted position} - 1)^2}. \quad (1)$$

RMSD is an often used quality estimate but it provides only a limited view on the *precision* of a (diagnosis) prediction. Therefore, we wanted to analyze the precision of the diagnosis selection strategies discussed in this paper—a measure for the precision of a diagnosis algorithm is depicted in Formula 2. The idea behind this measure is to describe how often a diagnosis that leads to a selected configuration (selected by the user) is among the topmost-*n* ranked diagnoses. As shown in Table 1, FastDiag and *best-first HSDAG* have highest prediction accuracy in terms of precision, whereas the *breadth-first HSDAG* approach shows the worst precision.

$$precision = \frac{|correctly\ predicted\ diagnoses|}{|predicted\ diagnoses|}. \quad (2)$$

We applied a Mann–Whitney *U* test in order to statistically analyze differences between the three diagnosis approaches in terms of ranking behavior. We conducted a pairwise comparison between the diagnosis approaches on the basis of the mentioned Mann–Whitney *U* test. We could identify a signif-

https://doi.org/10.1017/S0890060411000011 Published online by Cambridge University Press

Table 1. *Precision of FastDiag versus HSDAG based approaches*

Top-n Diagnoses	Breadth-First (HSDAG)	FastDiag	Best-First (HSDAG)
$n = 1$	0.51	0.70	0.74
$n = 2$	0.75	0.88	0.89
$n = 3$	0.87	0.97	0.96

Note: HSDAG, hitting set directed acyclic graph.

icant difference between the rankings of *best-first HSDAG* and *breadth-first HSDAG* based diagnosis ($p = 6.625e^{-5}$) and also between FastDiag and *breadth-first HSDAG* based diagnosis ($p < 2.441e^{-7}$). There was no significant difference between *best-first HSDAG* and FastDiag in terms of ranking behavior ($p = 0.12$).

6. RELATED WORK

6.1. Knowledge base analysis

The authors of Felfernig et al. (2004) introduce an algorithm for the automated debugging of configuration knowledge bases. The idea is to combine a conflict detection algorithm such as QuickXplain (Junker, 2004) with the hitting set algorithm used in model-based diagnosis (MBD; Reiter, 1987) for the calculation of minimal diagnoses. In this context, conflicts are induced by test cases (examples) that, for example, stem from previous configuration sessions, have been automatically generated, or have been explicitly defined by domain experts. Further applications of MBD in constraint set debugging are introduced in Felfernig et al. (2007) where diagnosis concepts are used to identify minimal sets of faulty transition conditions in state charts and in Felfernig et al. (2008) where MBD is applied for the identification of faulty utility constraint sets in the context of knowledge-based recommendation. In contrast to Felfernig et al. (2004, 2007, 2008), our work provides an algorithm that allows to directly determine diagnoses without the need to determine corresponding conflict sets. FastDiag can be applied in knowledge engineering scenarios for calculating preferred diagnoses for faulty knowledge bases given that we are able to determine reasonable ordering for the given set of constraints; this could be achieved, for example, by the application of corresponding complexity metrics (Chen & Suen, 2003).

6.2. Conflict detection

In contrast to the algorithm presented in this paper, calculating diagnoses for inconsistent requirements typically relies on the existence of (minimal) conflict sets. A well-known algorithm with a logarithmic number of consistency checks, depending on the number of constraints in the knowledge base and the cardinality of the minimal conflicts, is QuickXplain (Junker, 2004). It has made a major contribution to more efficient interactive constraint-based applications. QuickXplain is based on a divide and conquer strategy. FastDiag relies on the same principle of divide and conquer but with a different focus, namely, the determination of minimal diagnoses. QuickXplain calculates minimal conflict sets based on the assumption of a linear preference ordering among the constraints. Similarly, if we assume a linear preference ordering of the constraints in C, FastDiag calculates preferred diagnoses.

6.3. Interactive settings

Note that in the interactive configuration scenario discussed in this paper our goal was to support open configuration that lets the user explore the configuration space where the system proactively points out inconsistent requirements, such a functionality is often provided by commercial configuration environments. O'Sullivan et al. (2007) focus on interactive settings where users of constraint-based applications are confronted with situations where no solution can be found. In this context, O'Sullivan et al. (2007) introduce the concept of minimal exclusion sets that correspond to the concept of minimal diagnoses as defined in Reiter (1987). As mentioned, the major focus of O'Sullivan et al. (2007) are settings where the proposed algorithm supports users in the identification of acceptable exclusion sets. The authors propose an algorithm (representative explanations) that helps to improve the quality of the presented exclusion set (in terms of diversity) and thus increases the probability of finding an acceptable exclusion set for the user. Our diagnosis approach calculates preferred diagnoses in terms of a predefined ordering of the constraint set. Thus, compared to the work of O'Sullivan et al. (2007), we follow a different approach in terms of focusing more on preferences than on the degree of representativeness.

6.4. Diagnosis algorithms

There are a couple of algorithms that help to improve the efficiency of diagnosis determination; they are further developments of the original algorithm introduced by Reiter (1987). These approaches focus on making the construction of hitting sets more efficient. Wotawa (2001) introduces an algorithm that reduces the number of subset checks compared to the original HSDAG approach (Reiter, 1987). Fijany and Vatan (2004) introduce an approach to represent the problem of determining minimal hitting sets as a corresponding integer programming problem. Further approaches to optimize the determination of hitting sets are discussed in Lin and Jiang (2003). All the mentioned approaches rely on (minimal) conflict sets that are the basis for calculating a set of minimal diagnoses, whereas FastDiag is a complete and minimal diagnosis algorithm without the need of conflict sets. It is important to mention that especially when calculating the first n-diagnoses (for $n > 1$, i.e., not a single diagnosis), FastDiag can also exploit the mentioned algorithms of Lin and Jiang (2003) and Wotawa (2001) for the calculation of more than one diagnosis, that is, it is not bound to the usage of the original HSDAG algorithm.

Lin and Jiang (2002) introduce an approach to determine hitting sets on the basis of genetic algorithms; a similar approach to the determination of diagnoses is presented in Feldman et al. (2008) who introduce a stochastic fault diagnosis algorithm, which is based on greedy stochastic search. Such approaches show to significantly improve search performance; however, there is no general guarantee of completeness and diagnosis minimality. Finally, there exist a couple of algorithms that are improving the algorithmic performance of diagnosis calculation due to additional knowledge about the structural properties of the diagnosis problem. For example, Jannach and Liegl (2006) show the determination of (minimal) diagnoses for the case of conjunctive queries on database tables (the set of diagnoses can be precompiled by executing the individual parts of the query on the given data set). Siddiqi and Huang (2007) show one approach to exploit structural properties of system descriptions to improve the overall performance of diagnosis determination; in this case *cones* are areas in a gate with a certain structure and a certain probability of including a diagnosis, and the search process focuses on exactly those areas. FastDiag does not exploit specific properties of the underlying constraint set; however, taking into account such properties can further improve the performance of the algorithm. Corresponding evaluations are within the scope of future work.

6.5. Personalized diagnosis

Many of the existing diagnosis approaches do not take into account the need for personalizing the set of diagnoses to be presented to a user. Identifying diagnoses of interest in an efficient manner is a clear surplus regarding the acceptance of the underlying application, for example, users of a configurator application are not necessarily interested in minimal cardinality diagnoses (Reiter, 1987) but rather in those that correspond to their current preferences. A first step toward the application of personalization concepts in the context of knowledge-based recommendation is presented in Felfernig et al. (2009). The authors introduce an approach that calculates leading diagnoses on the basis of similarity measures used for determining *n*-nearest neighbors. A general approach to the identification of preferred diagnoses is introduced in DeKleer (1990), where probability estimates are used to determine the leading diagnoses with the overall goal to minimize the number of measurements needed for identifying a malfunctioning device. Basic principles of determining diagnoses in knowledge-based recommendation scenarios are discussed in Jannach and Liegl (2006). Furthermore, Froehlich et al. (1994) introduce a logical characterization of preferences that are expressed as preference relations on single diagnoses and modal logical formulas on groups of diagnoses. In contrast to our work, Froehlich et al. (1994) do not provide an algorithm to efficiently calculate preferred diagnoses. We see our work as a major contribution in this context because FastDiag helps to identify leading diagnoses more efficiently. Further empirical studies in different application contexts are within the major focus of our future work.

7. CONCLUSION

In this paper we have introduced a new diagnosis algorithm (FastDiag), which allows the efficient calculation of one diagnosis at a time with logarithmic complexity in terms of the number of consistency checks. Thus, the computational complexity for the calculation of one minimal diagnosis is equal to the calculation of one minimal conflict set in hitting set-based diagnosis approaches. The algorithm is especially applicable in settings where the number of conflict sets is equal to or larger than the number of diagnoses, or in settings where preferred (leading) diagnoses are needed. Issues for future work are the determination of repair actions for diagnoses, the further development of FastDiag for supporting anytime diagnosis tasks, and the conduction of further empirical studies in different configurator application domains.

REFERENCES

Ardissono, L., Felfernig, A., Friedrich, G., Jannach, D., Petrone, G., Schaefer, R., & Zanker, M. (2003). A framework for the development of personalized, distributed web-based configuration systems. *AI Magazine 24(3)*, 93–108.

Belanger, F. (2005). A conjoint analysis of online consumer satisfaction. *Journal of Electronic Commerce Research 6*, 95–111.

Castillo, L., Borrajo, D., & Salido, M. (2005). *Planning, Scheduling and Constraint Satisfaction: From Theory to Practice*. Amsterdam: IOS Press.

Chen, Z., & Suen, C. (2003). Measuring the complexity of rule-based expert systems. *Expert Systems with Applications 7(4)*, 467–481.

DeKleer, J. (1990). Using crude probability estimates to guide diagnosis. *AI Journal 45(3)*, 381–391.

DeKleer, J., Mackworth, A., & Reiter, R. (1992). Characterizing diagnoses and systems. *AI Journal 56(2–3)*, 197–222.

DeKleer, J., & Williams, B. (1987). Diagnosing multiple faults. *AI Journal 32(1)*, 97–130.

Feldman, A., Provan, G., & Gemund, A. (2008). Computing minimal diagnoses by greedy stochastic search. *Proc. 23rd AAAI Conf. Artificial Intelligence (AAAI'08)*, pp. 911–918, Chicago.

Felfernig, A., Friedrich, G., Isak, K., Shchekotykhin, K., Teppan, E., & Jannach, D. (2007). Automated debugging of recommender user interface descriptions. *Journal of Applied Intelligence 31(1)*, 1–14.

Felfernig, A., Friedrich, G., Jannach, D., & Stumptner, M. (2004). Consistency-based diagnosis of configuration knowledge bases. *AI Journal 152(2)*, 213–234.

Felfernig, A., Friedrich, G., Schubert, M., Mandl, M., Mairitsch, M., & Teppan, E. (2009). Plausible repairs for inconsistent requirements. *Proc. 21st Int. Joint Conf. Artificial Intelligence (IJCAI'09)*, pp. 791–796, Pasadena, CA.

Felfernig, A., Friedrich, G., Teppan, E., & Isak, K. (2008). Intelligent debugging and repair of utility constraint sets in knowledge-based recommender applications. *Proc. 13th ACM Int. Conf. Intelligent User Interfaces (IUI'08)*, pp. 218–226, Canary Islands, Spain.

Fijany, A., & Vatan, F. (2004). New approaches for efficient solutions of hitting set problems. *Proc. Int. Symp. Information and Communication Technologies*, pp. 1–10, Cancun, Mexico.

Fleischanderl, G., Friedrich, G., Haselboeck, A., Schreiner, H., & Stumptner, M. (1998). Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems 13(4)*, 59–68.

Friedrich, G., & Shchekotykhin, K. (2005). A general diagnosis method for ontologies. *Proc. 4th Int. Semantic Web Conference (ISWC'05)*, LNCS, Vol. 3729, pp. 232–246. New York: Springer.

Fröhlich, P., Nejdl, W., & Schroeder, M. (1994). A formal semantics for preferences and strategies in model-based diagnosis. *Proc. 5th Int. Workshop on Principles of Diagnosis (DX-94)*, pp. 106–113.

Jannach, D., & Liegl, J. (2006). Conflict-directed relaxation of constraints in content-based recommender systems. *Proc. IEA/AIE 2006*, pp. 819–829, Annency, France.

Junker, U. (2004). QuickXplain: preferred explanations and relaxations for over-constrained problems. *Proc. 19th National Conf. Artificial Intelligence (AAAI'04)*, pp. 167–172, San Jose, CA.

Lin, L., & Jiang, Y. (2002). Computing minimal hitting sets with genetic algorithms. *Algorithmica 32(1)*, 95–106.

Lin, L., & Jiang, Y. (2003). The computation of hitting sets: review and new algorithm. *Information Processing Letters 86*, 177–184.

Marques-Silva, J., & Sakallah, K. (1996). Grasp: a new search algorithm for satisfiability. *Proc. Int. Conf. Computer-Aided Design*, pp. 220–227, Santa Clara, CA.

Mittal, S., & Frayman, F. (1989). Towards a generic model of configuration tasks. *Proc. 11th Int. Joint Conf. Artificial Intelligence (IJCAI'89)*, pp. 1395–1401, Detroit, MI.

O'Sullivan, B., Papdopoulos, A., Faltings, B., & Pu, P. (2007). Representative explanations for over-constrained problems. *Proc. 22nd National Conf. Artificial Intelligence (AAAI'07)*, pp. 323–328, Vancouver, Canada.

Reiter, R. (1987). A theory of diagnosis from first principles. *AI Journal 23(1)*, 57–95.

Siddiqi, S., & Huang, J. (2007). Hierarchical diagnosis of multiple faults. *Proc. 20th Int. Joint Conf. Artificial Intelligence (IJCAI'07)*, pp. 581–586, Hyderabad, India.

Sinz, C., & Haag, A. (2007). Configuration. *IEEE Intelligent Systems 22(1)*, 78–90.

Tsang, E. (1993). *Foundations of Constraint Satisfaction*. New York: Academic.

Winterfeldt, D., & Edwards, W. (1986). *Decision Analysis and Behavioral Research*. New York: Cambridge University Press.

Wotawa, F. (2001). A variant of Reiter's hitting-set algorithm. *Information Processing Letters 79*, 45–51.

Alexander Felfernig is a Professor of applied software engineering at Graz University of Technology. Alexander is also Cofounder and Director of ConfigWorks, a company focused on the development of knowledge-based recommendation technologies. Prof. Felfernig's research focuses on intelligent methods and algorithms supporting the development and maintenance of complex knowledge bases. Furthermore, he is interested in the application of AI techniques in the software engineering context, for example, the application of decision and recommendation technologies to make software requirements engineering processes more effective. In 2009, Dr. Felfernig received the Heinz–Zemanek Award from the Austrian Computer Society for his research.

Monika Schubert is a PhD student in the group of Applied Software Engineering at Graz University of Technology. She received her MS in software engineering and economy from Graz University of Technology. Her research focuses on knowledge-based systems, intelligent product configuration, model-based diagnosis, and product recommendation. Monika is also interested in user interaction with complex knowledge bases.

Christoph Zehentner is a PhD student in the Applied Software Engineering Group at Graz University of Technology. He received his MS in software engineering and economy from Graz University of Technology. His research focuses on testing and debugging of knowledge-based systems as well as recommendations in group decision environments. Christoph is also interested in multiple-agent system coordination and decision making.