A Diagnosis Algorithm for Inconsistent Constraint Sets

Alexander Felfernig [1], **Monika Schubert** [1]

[1] *Graz University of Technology, Inffeldgasse 16b, 8010 Graz, Austria*
alexander.felfernig@ist.tugraz.at
monika.schubert@ist.tugraz.at

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 constraints that have to be adapted or deleted in order to restore consistency. In this paper we introduce a divide-and-conquer based diagnosis algorithm (FastDiag) which identifies minimal sets of faulty constraints in an over-constrained 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.

1 INTRODUCTION

Constraint technologies (Tsang, 1993) are applied in different areas such as configuration (Fleischanderl *et al.*, 1998; Mittal and Frayman, 1989; Sinz and Haag, 2007), recommendation (Felfernig *et al.*, 2009), and scheduling (Castillo *et al.*, 2005). There are many scenarios where the underlying constraint sets can become over-constrained. 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 (Felfernig *et al.*, 2009; O'Sullivan *et al.*, 2007), 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 wide-spread approach to support users in the identification of minimal sets of faulty constraints is to combine conflict detection (see, e.g., (Junker, 2004)) with a corresponding hitting set algorithm (Reiter, 1987; DeKleer *et al.*, 1992). In their original form these algorithms are applied for the calculation of *minimal diagnoses* which 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). Note that the complement of such a minimal diagnosis can also be denoted as maximally successful sub-query (McSherry, 2004).

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 a logarithmic number of consistency checks. 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 & reconfiguration (Felfernig *et al.*, 2004), recommendation of products & 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 is also applicable, for example, in the context of SAT solving (Marques-Silva and Sakallah, 1996) and description logics reasoning (Friedrich and 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 cal-

culation of diagnoses. In Section 4 we introduce an algorithm (FastDiag) for calculating preferred diagnoses for a given over-constrained 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. Since 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). V = $\{v_1, v_2, \ldots, v_n\}$ represents a set of finite domain variables. D = $\{dom(v_1), dom(v_2), \ldots, dom(v_n)\}$ represents a set of variable domains $dom(v_k)$ where $dom(v_k)$ 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. C_R = $\{c_{q+1}, c_{q+2}, \ldots, c_t\}$ is a set of customer requirements 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 parc distance control functionality, *fuel* represents the fuel consumption per 100 kilometers, a *skibag* allows the 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 which are denoted as *configuration knowledge base*, C_{KB} = $\{c_1, c_2, c_3, c_4\}$. Furthermore, we assume the set of *customer requirements* C_R = $\{c_5, c_6, c_7\}$.

- V = {***type***, ***pdc***, ***fuel***, ***skibag***, ***4-wheel***}
- D = {dom(***type***)={*city*, *limo*, *combi*, *xdrive*}, dom(***pdc***)= {*yes*, *no*}, dom(***fuel***) = {*4l*, *6l*, *10l*}, dom(***skibag***)={*yes*, *no*}, dom(***4-wheel***)={*yes*, *no*}
- C_{KB} = {c_1: ***4-wheel*** = *yes* ⇒ ***type*** = *xdrive*, c_2: ***skibag*** = *yes* ⇒ ***type*** ≠ *city*, c_3: ***fuel*** = *4l* ⇒ ***type*** = *city*, c_4: ***fuel*** = *6l* ⇒ ***type*** ≠ *xdrive*}
- C_R = {c_5: ***type*** = *combi*, c_6: ***fuel*** = *4l*, c_7: ***4-wheel*** = yes}

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 = $\{v_1=ins_1, v_2=ins_2, \ldots, v_n=ins_n\}$ where $ins_k \in dom(v_k)$.

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 OVER-CONSTRAINED 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 kilometers*. Consequently, we want to identify minimal sets of constraints ($c_i \in C_R$) which have to be deleted (or adapted) 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}, i.e., 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 over-constrained problem is to resolve all minimal conflicts contained in the constraint set. The determination of such constraints is based on a conflict detection algorithm (see, e.g., (Junker, 2004)), the derivation of the corresponding diagnoses is based on the calculation of hitting sets (DeKleer and Williams, 1987; Reiter, 1987). Since 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 CS ⊆ C_R s.t. $C_{KB} \cup$ CS is inconsistent. A conflict set CS is a *minimal* if there does not exist a conflict set CS' with CS' ⊂ CS.

In our working example we can identify three minimal conflict sets which are CS_1={c_5,c_6}, CS_2={c_5,c_7}, and CS_3={c_6,c_7}.

CS_1, CS_2, CS_3 are conflict sets since $CS_1 \cup C_{KB} \vee CS_2 \cup C_{KB} \vee CS_3 \cup C_{KB}$ is inconsistent. The minimality property is fulfilled since there does not exist a conflict set CS_4 with $CS_4 \subset CS_1$ or $CS_4 \subset CS_2$ or $CS_4 \subset CS_3$. 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 (or adapted) in order to make the remaining constraints consistent with C_{KB}. Since 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 (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 $\Delta \subseteq C_R$, s.t., $C_{KB} \cup (C_R - \Delta)$ is consistent. Δ is *minimal* if there does not exist a diagnosis $\Delta' \subset \Delta$ s.t. $C_{KB} \cup$

(C_R - Δ') 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}. Due to 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} since 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 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 Δ_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 Δ_2 = {c_5, c_7} since 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 – since {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 Δ_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. Since users are often interested in a reduced subset of all the potential diagnoses, alternative algorithms are needed that are capable of identifying preferred diagnoses (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.

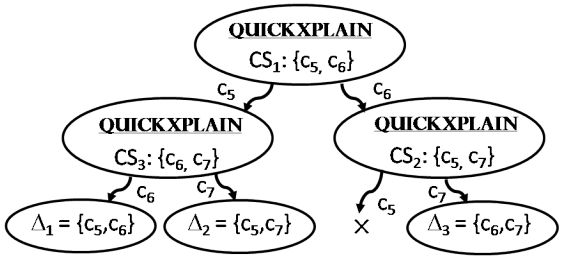

Figure 1: HSDAG (Hitting Set Directed Acyclic Graph) for the CR diagnosis problem (C_R={c_5, c_6, c_7}, C_{KB}={c_1, c_2, c_3, c_4}). The sets {c_5, c_6}, {c_6, c_7}, and {c_5, c_7} are the minimal diagnoses - the conflict sets CS_1, CS_2, and CS_3 are determined on the basis of QuickXplain .

4 CALCULATING PREFERRED DIAGNOSES WITH FASTDIAG

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 *multi-attribute utility theory* (Ardissono *et al.*, 2003; Winterfeldt and Edwards, 1986) 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 (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 as follows:

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

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

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

Based on this definition of a lexicographical ordering, we can now introduce 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}) iff there does not exist a minimal diagnosis Δ' with $\Delta' >_{lex} \Delta$.

In our working example we assumed the lexicographical ordering ($c_5 < c_6 < c_7$), i.e., 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 $\{c_6, c_7\}$ since 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.

4.1 FastDiag Approach

For the following discussions we introduce the set AC = $C_{KB} \cup C_R$ which represents the union of customer requirements (C_R) and the configuration knowledge base (C_{KB}). The basic idea of the FastDiag algorithm (Algorithm 1) is the following.[1] In our working example, the set of customer requirements $C_R = \{c_5, c_6, c_7\}$ includes at least one minimal diagnosis since C_{KB} is consistent and $C_{KB} \cup C_R$ is inconsistent. If C_R itself represents the minimal diagnosis this means that *all* constraints in C_R are part of the diagnosis. 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 since 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 which means that we do not have to further take into account $\{c_7\}$ for determining the diagnosis. Since $\{c_5\}$ 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, i.e., $\Delta = \{c_5, c_6\}$ 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 example is shown in Figure 2.

Algorithm 1 – FastDiag

```
1  func FastDiag(C ⊆ AC, AC = {c_1..c_t}) : Δ
2  if isEmpty(C) or inconsistent(AC − C) return ∅
3  else return FD(∅, C, AC);

4  func FD(D, C = {c_1..c_q}, AC) : diagnosis Δ
5  if D ≠ 0 and consistent(AC) return ∅;
6  if singleton(C) return C;
7  k = q/2;
8  C_1 = {c_1..c_k}; C_2 = {c_{k+1}..c_q};
9  D_1 = FD(C_1, C_2, AC − C_1);
10 D_2 = FD(D_1, C_1, AC − D_1);
11 return(D_1 ∪ D_2);
```

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

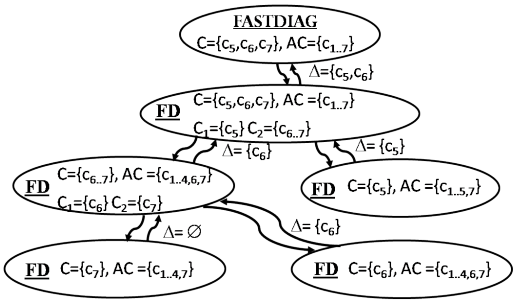

Figure 2: FastDiag execution trace for the CR diagnosis problem (C_R=$\{c_5, c_6, c_7\}$, C_{KB}=$\{c_1, c_2, c_3, c_4\}$).

4.2 Calculating n>1 Diagnoses

In order to be able to calculate n>1 diagnoses[2] with FastDiag we have to adopt the HSDAG construction introduced in (Reiter, 1987) by substituting the resolution of conflicts (see Figure 1) with the deletion of elements c_i from C_R (C) (see Figure 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 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 for finding another minimal diagnosis for the CR diagnosis problem ($\{c_6, c_7\}$, C_{KB}). Since AC - $\{c_5\}$ is inconsistent, we can conclude that $C_R = \{c_6, c_7\}$ includes another minimal diagnosis ($\Delta_2 = \{c_6, c_7\}$) which is determined by FastDiag for the CR diagnosis problem (C_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, i.e., we have identified the last minimal diagnosis which 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 ($\{c_5, c_6, c_7\}$-$\{c_5\}$, $\{c_1, ..., c_7\}$) which is Δ_2 in our example case, i.e., $\Delta_1 >_{lex} \Delta_2$. Consequently, diagnoses part of one path in the search tree (such as Δ_1 and Δ_2 in Figure 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.

4.3 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, i.e., whether AC - C is consistent – the function assumes

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

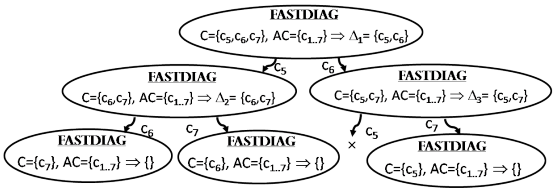

Figure 3: FastDiag: calculating the complete set of *minimal diagnoses*.

that it is activated in the case that AC is inconsistent. If AC - C is inconsistent or C = ∅, FastDiag returns the empty set as result (no solution can be found). If at least one diagnosis is contained in the set of constraints C, FastDiag activates the FD function which is in charge of retrieving a preferred diagnosis. FastDiag follows a divide-and-conquer strategy where the recursive function FD divides the set of constraints (in our case the elements of C_R) into two different subsets (C_1 and C_2) and tries to figure out whether C_1 already contains a diagnosis. 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. 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 can not be identified by FastDiag, this would mean that there exists at least one constraint c_a in C which 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 ∅ statement in FD and ∅ 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 non-minimal diagnosis Δ calculated by FastDiag, this would mean that there exists at least one constraint c_a with Δ - $\{c_a\}$ 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 FD) then c_a is the only element that can be deleted from AC, i.e., c_a must be part of the diagnosis.

5 EVALUATION

5.1 Performance of FastDiag

In this subsection we 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 $2d \cdot \log_2(\frac{n}{d})+2d$, where d is the minimal diagnoses set size and n is the number of constraints (in C). The best case complexity is $\log_2(\frac{n}{d})+2d$. In the worst case each element of the diagnosis is contained in a different path of the search tree: $\log_2(\frac{n}{d})$ 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 $2k \cdot \log_2(\frac{n}{k})+2k$ 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 $\log_2(\frac{n}{k})+2k$ (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} FD 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 Figures 4–7. The basis for these evaluations were generated constraint sets (t = 100 constraints with a randomized lexicographical ordering and n = 100 variables) with a varying number of conflict sets (of cardinality 1–4) and corresponding diagnoses (#diagnoses between 3 – 22). The constraint solver used for consistency checking was CHOCO (*choco.emn.fr)* and the tests have been executed on a standard desktop computer (*Intel(R) Core(TM)2 Quad CPU QD9400* CPU with *2.66Ghz* and *2GB* RAM). Each experiment (data point in the graph) has been conducted 10 times with a differing constraint ordering. 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, i.e., FastDiag outperforms the two HSDAG versions. Our evaluations show that FastDiag is very efficient in calculating preferred minimal diagnoses. In contrast to the HSDAG-based best-first search mode FastDiag has a performance that makes it an excellent choice for interactive settings.

5.2 Prediction Quality of FastDiag

On the basis of interaction data collected with a computer configuration environment we also analyzed the prediction quality of FastDiag. We measured the quality of the algorithm in terms of precision (see Formula 1): *how often a diagnosis that leads to a selected configuration (selected by the user) is among the topmost-*

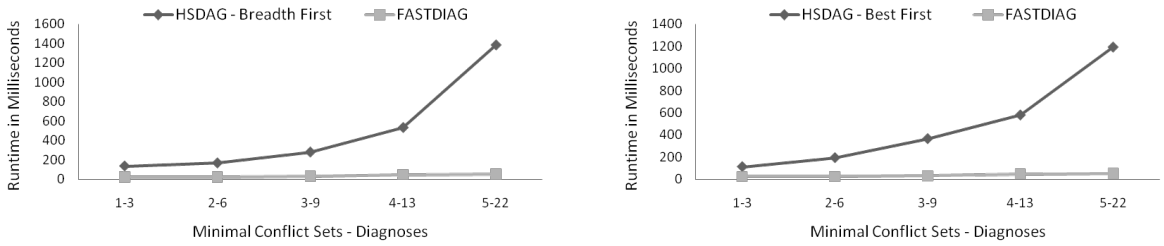

Figure 4: Calculating the *first min. diagnosis* with FastDiag vs. hitting set based diagnosis with QuickXplain.

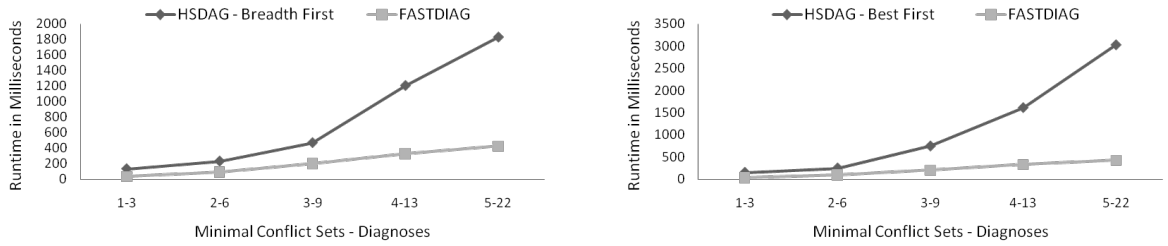

Figure 5: Calculating *topmost-5 min. diagnoses* with FastDiag vs. hitting set based diagnosis with QuickXplain.

n diagnoses ranked (predicted) by FastDiag.

$$precision = \frac{|correctly\ predicted\ diagnoses|}{|predicted\ diagnoses|} \quad (1)$$

The major result of this study was that we could not detect any signficant difference in prediction quality between FastDiag and best-first search based HSDAG. Due to space limitations a detailed characterization of the study results will be provided in an extended version of this paper.

5.3 A Simple Extension: FlexDiag

We now introduce a simple modification of FastDiag (FlexDiag) which allows us to further reduce the number of consistency checks in the case we want to calculate exactly one (preferred) diagnosis. This can be achieved with the tradeoff of loosing the minimality property (see Definition 5). Note that further evaluations and extensions of FlexDiag are within the scope of future work.

A reduction of the number of consistency checks can be achieved by changing the *singleton(C)* statement in line 6 of Algorithm 1 to *size*(C) $\leq m$. If we check the current set of constraints C for the containment of $\leq m$ elements (for $m>1$), the modified algorithm (FlexDiag) determines a constraint set which includes the minimal diagnosis (but itself is not necessarily minimal). Consequently, there is a tradeoff: if we increase *m*, we decrease the number of consistency checks and vice-versa.

FlexDiag calculates a set of constraints C_D where $C_D = \Delta \cup C_{offset}$. Δ is the minimal (and preferred) diagnosis calculated by Algorithm 1 and C_{offset} is a set of constraints which are not part of Δ. We evaluated this approach w.r.t. the relevance (see Formula 2) of C_D. We define relevance as the share of relevant constraints (all constraints in Δ) w.r.t. the constraints in C_D (see Formula 2):

$$relevance(C_D) = \frac{|\Delta|}{|C_D|} \quad (2)$$

We calculated the average relevance of diagnoses determined by FlexDiag for our dataset (see Section 5.1). The relevance for *m*=2 is around 60% which means that 40% of the constraints are not part of the minimal (and preferred) diagnosis, on an average (calculated from 100 runs with randomized constraint order). Further increasing the value of *m* (for *m* = 3, 4, and 5) significantly decreases the relevance of the corresponding diagnoses C_D (see Figure 8). Lower relevance values correlate with decreasing runtimes (see Table 1).

m=2	m=3	m=4	m=5
16.6%	33.8%	35.5%	41.1%

Table 1: Runtime improvements achieved with FlexDiag, for example, with *m*=2 we can reduce the average runtime for calculating *one diagnosis* by 16.6%.

6 RELATED WORK

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; DeKleer *et al.*, 1992) for the calculation of minimal diagnoses. In this context, conflicts are induced by test cases (examples) that, for example, are stemming from previous

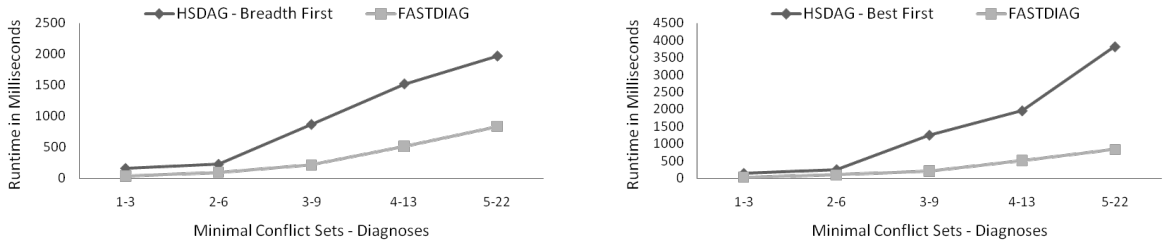

Figure 6: Calculating *topmost-10 min. diagnoses* with FastDiag vs. hitting set based diagnosis with QuickXplain.

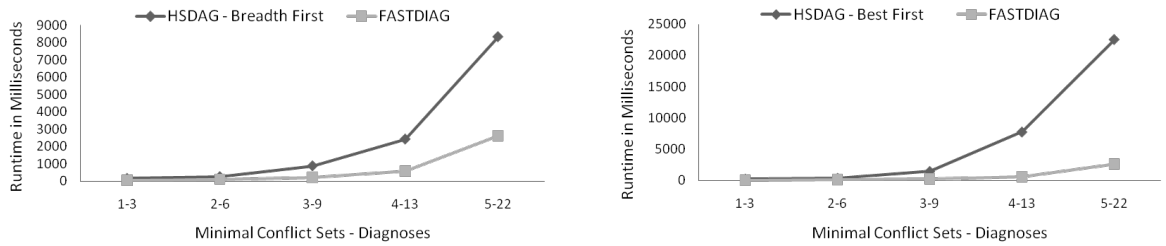

Figure 7: Calculating *all min. diagnoses* with FastDiag vs. hitting set based diagnosis with QuickXplain.

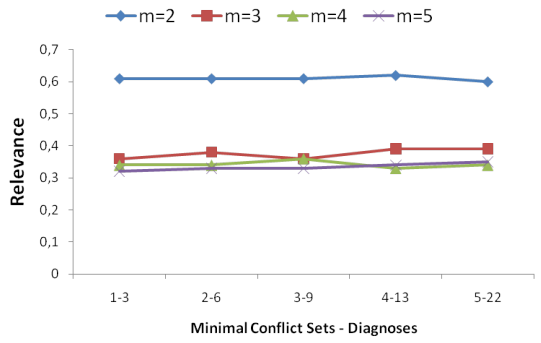

Figure 8: Relevance of diagnoses C_D (see Formula 2) calculated by FlexDiag.

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.*, 2007; 2004; 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 and Suen, 2003).

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 QuickXplain (Junker, 2004) 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.

The authors of (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 which correspond to the concept of minimal diagnoses as defined in (Reiter, 1987; DeKleer *et al.*, 1992). As mentioned, the major focus of (O'Sullivan *et al.*, 2007) are interactive 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 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.

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. A first step towards 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. We see our work as a major contribution in this context since 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.

REFERENCES

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

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

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

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

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

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

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

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

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

(Felfernig *et al.*, 2008) A. Felfernig, G. Friedrich, E. Teppan, and K. Isak. Intelligent debugging and repair of utility constraint sets in knowledge-based recommender applications. In *IUI'08*, pages 218–226, 2008.

(Felfernig *et al.*, 2009) A. Felfernig, G. Friedrich, M. Schubert, M. Mandl, M. Mairitsch, and E. Teppan. Plausible repairs for inconsistent requirements. In *IJCAI'09*, pages 791–796, 2009.

(Fleischanderl *et al.*, 1998) G. Fleischanderl, G. Friedrich, A. Haselboeck, H. Schreiner, and M. Stumptner. Configuring large systems using generative constraint satisfaction. *IEEE Intell. Syst.*, 13(4):59–68, 1998.

(Friedrich and Shchekotykhin, 2005) G. Friedrich and K. Shchekotykhin. A general diagnosis method for ontologies. In *4th International Semantic Web Conference (ISWC'05)*, pages 232–246, 2005.

(Junker, 2004) U. Junker. Quickxplain: Preferred explanations and relaxations for over-constrained problems. In *AAAI'04*, pages 167–172, 2004.

(Marques-Silva and Sakallah, 1996) J. Marques-Silva and K. Sakallah. Grasp: A new search algorithm for satisfiability. In *International Conference on Computer-Aided Design*, pages 220–227, 1996.

(McSherry, 2004) D. McSherry. Maximally successful relaxations of unsuccessful queries. In *15th Conference on Artificial Intelligence and Cognitive Science*, pages 127–136, 2004.

(Mittal and Frayman, 1989) S. Mittal and F. Frayman. Towards a generic model of coniguration tasks. In *IJCAI'89*, pages 1395–1401, 1989.

(O'Sullivan *et al.*, 2007) Barry O'Sullivan, A. Papdopoulos, B. Faltings, and P. Pu. Representative explanations for over-constrained problems. In *AAAI'07*, pages 323–328, 2007.

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

(Sinz and Haag, 2007) C. Sinz and A. Haag. Configuration. *IEEE Intell. Syst.*, 22(1):78–90, 2007.

(Tsang, 1993) E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

(Winterfeldt and Edwards, 1986) D. Winterfeldt and W. Edwards. Decision analysis and behavioral research. *Cambridge University Press*, 1986.