

FLEXDIAG: AnyTime Diagnosis for Reconfiguration

Alexander Felfernig¹ and Rouven Walter² and Stefan Reiterer¹

Abstract. Anytime diagnosis is able to determine solutions within predefined time limits. This is especially useful in realtime scenarios such as production scheduling, robot control, and communication networks management where diagnosis and corresponding reconfiguration capabilities play a major role. Anytime diagnosis in many cases comes along with a tradeoff between diagnosis quality and the efficiency of diagnostic reasoning. In this paper we introduce and analyze FLEXDIAG which is an anytime variant of existing direct diagnosis approaches. We evaluate the algorithm with regard to performance and diagnosis quality using a configuration benchmark.

Keywords: Anytime Diagnosis, Reconfiguration.

1 Introduction

Knowledge-based configuration is one of the most successful applications of Artificial Intelligence [7, 24]. There are many different applications of configuration technologies ranging from *telecommunication infrastructures* [11], *railway interlocking systems* [5], the *automotive domain* [22, 26, 28] to the *configuration of services* [27]. Configuration technologies must be able to deal with inconsistencies which can occur in different contexts. First, a configuration knowledge base can be inconsistent, i.e., no solution can be determined. In this context, the task of knowledge engineers is to figure out which constraints are responsible for the unintended behavior of the knowledge base. Bakker et al. [1] show the application of model-based diagnosis [19] to determine minimal sets of constraints in a knowledge base that are responsible for a given inconsistency. A variant thereof is documented in Felfernig et al. [6] where an approach to the automated debugging of knowledge bases with test cases is introduced. Felfernig et al. [6] also show how to diagnose customer requirements that are inconsistent with a configuration knowledge base. The underlying assumption is that the configuration knowledge base itself is consistent but combined with a set of requirements is inconsistent.

All diagnosis approaches mentioned so far are based on conflict-directed hitting set determination [15, 19]. These approaches typically determine diagnoses in a breadth-first search manner which allows the identification of minimal cardinality diagnoses. The major disadvantage of applying these approaches is the need of pre-determining minimal conflicts which is inefficient especially in cases where only the leading diagnoses (the most relevant ones) are sought.

Anytime diagnosis algorithms are useful in scenarios where diagnoses have to be provided in real-time, i.e., within given time limits. If diagnosis is applied in *interactive configuration*, for example, to determine repairs for inconsistent customer requirements, response times should be below one second [2]. Efficient diagnosis and reconfiguration of *communication networks* is crucial to retain the quality of service [18, 25]. In today's production scenarios which are characterized by small batch sizes and high product variability, it is increasingly important to develop algorithms that support the efficient *reconfiguration of schedules*. Such functionalities support the paradigm of *smart production*, i.e., the flexible and efficient production of highly variant products. Further applications are the diagnosis and repair of *robot control software* [23], the *reconfiguration of cars* [29], and the *reconfiguration of buildings* [12].

Algorithmic approaches to provide efficient solutions for diagnosis problems are manifold. Some approaches focus on improvements of Reiter's original hitting set directed acyclic graph (HSDAG) [19] in terms of a personalized computation of leading diagnoses [3] or other extensions that make the basic approach [19] more efficient [31]. Wang et al. [30] introduce an approach to derive binary decision diagrams (BDDs) on the basis of a pre-determined set of conflicts – diagnoses can then be determined by solving the BDD. A pre-defined set of conflicts can also be compiled into a corresponding linear optimization problem [10]; diagnoses can then be determined by solving the given problem. In knowledge-based recommendation scenarios, diagnoses for user requirements can be pre-compiled in such a way that for a given set of customer requirements, the diagnosis search task can be reduced to querying a relational table (see, for example, [14, 20]). All of the mentioned approaches either extend the approach of Reiter [19] or improve efficiency by exploiting pre-generated information about conflicts or diagnoses.

An alternative to conflict-directed diagnosis [19] are *direct diagnosis* algorithms that determine minimal diagnoses without the need of pre-determining minimal conflict sets [9, 17, 21]. The FASTDIAG algorithm [9] is a divide-and-conquer based algorithm that supports the determination of diagnoses without a preceding conflict detection. In this paper we show how this algorithm can be converted into an anytime diagnosis algorithm (FLEXDIAG) that is able to improve performance by disregarding the aspect of minimality, i.e., the algorithm allows for tradeoffs between diagnosis quality (e.g., minimality) and performance of diagnostic search. In this paper we focus on *reconfiguration scenarios*, i.e., we show how FLEXDIAG can be applied in situations where a given configuration (solution) has to be adapted conform to a changed set of customer requirements.

Our contributions in this paper are the following. First, we show how to solve reconfiguration tasks with direct diagnosis. Second, we make direct diagnosis anytime-aware by including a parametrization

¹ Applied Software Engineering Group, Institute for Software Technology, TU Graz, Austria, email: {alexander.felfernig, stefan.reiterer}@ist.tugraz.at

² Symbolic Computation Group, WSI Informatics, Universität Tübingen, Germany, email: rouven.walter@uni-tuebingen.de

that helps to systematically reduce the number of consistency checks. Finally, we report the results of a FLEXDIAG-related evaluation conducted on the basis of a configuration benchmark.

The remainder of this paper is organized as follows. In Section 2 we introduce an example configuration knowledge base from the domain of resource allocation. This knowledge base will serve as a working example throughout the paper. Thereafter (Section 3) we introduce a definition of a reconfiguration task. In Section 4 we discuss basic principles of direct diagnosis on the basis of FLEXDIAG and show how this algorithm can be applied in reconfiguration scenarios. In Section 5 we present the results of a performance analysis. A simple example of the application of FLEXDIAG in production environments is given in Section 6. In Section 7 we discuss major issues for future work. With Section 8 we conclude the paper.

2 Example Configuration Knowledge Base

A configuration system determines configurations (solutions) on the basis of a given set of customer requirements [13]. In many cases, constraint satisfaction problem (CSP) representations [16] are used for the definition of a configuration task. A configuration task and a corresponding configuration (solution) can be defined as follows.

Definition 1 (Configuration Task and Configuration). A configuration task can be defined as a CSP (V, D, C) where $V = \{v_1, v_2, \dots, v_n\}$ is a set of variables, $D = \cup \text{dom}(v_i)$ represents domain definitions, and $C = \{c_1, c_2, \dots, c_m\}$ is a set of constraints. Additionally, user requirements are represented by a set of constraints $R = \{r_1, r_2, \dots, r_k\}$. A configuration (solution) for a configuration task is a set of assignments (constraints) $S = \{s_1 : v_1 = a_1, s_2 : v_2 = a_2, \dots, s_n : v_n = a_n\}$ where $a_i \in \text{dom}(v_i)$ which is consistent with $C \cup R$.

An example of a configuration task represented as a constraint satisfaction problem is the following.

Example (Configuration Task). In this resource allocation problem example, items (a barrel of fuel, a stack of paper, a pallet of fireworks, a pallet of personal computers, a pallet of computer games, a barrel of oil, a palette of roof tiles, and a palette of rain pipes) have to be assigned to three different containers. There are a couple of constraints (c_i) to be taken into account, for example, fireworks must not be combined with fuel (c_1). Furthermore, there is one requirement (r_1) which indicates that the palette of fireworks has to be assigned to container 1. On the basis of this configuration task definition, a configurator can determine a configuration S .

- $V = \{\text{fuel}, \text{paper}, \text{fireworks}, \text{pc}, \text{games}, \text{oil}, \text{roof}, \text{pipes}\}$
- $\text{dom}(\text{fuel}) = \text{dom}(\text{paper}) = \text{dom}(\text{fireworks}) = \text{dom}(\text{pc}) = \text{dom}(\text{games}) = \text{dom}(\text{oil}) = \text{dom}(\text{roof}) = \text{dom}(\text{pipes}) = \{1, 2, 3\}$
- $C = \{c_1 : \text{fireworks} \neq \text{fuel}, c_2 : \text{fireworks} \neq \text{paper}, c_3 : \text{fireworks} \neq \text{oil}, c_4 : \text{pipes} = \text{roof}, c_5 : \text{paper} \neq \text{fuel}\}$
- $R = \{r_1 : \text{fireworks} = 1\}$
- $S = \{s_1 : \text{pc} = 3, s_2 : \text{games} = 1, s_3 : \text{paper} = 2, s_4 : \text{fuel} = 3, s_5 : \text{fireworks} = 1, s_6 : \text{oil} = 2, s_7 : \text{roof} = 1, s_8 : \text{pipes} = 1\}$

On the basis of the given definition of a configuration task, we now introduce the concept of reconfiguration (see also [12, 18, 25, 28]).

3 Reconfiguration Task

It can be the case that an existing configuration S has to be adapted due to a change or extension of the given set of customer require-

ments. Examples thereof are changing requirements that have to be taken into account in production schedules, failing components or overloaded network infrastructures in a mobile phone network, and changes in the internal model of the environment of a robot. In the following we assume that the *palette of paper* should be reassigned to container 3 and the *personal computer* and *games palettes* should be assigned to the same container. Formally, the set of new requirements is represented by $R_\rho : \{r'_1 : \text{pc} = \text{games}, r'_2 : \text{paper} = 3\}$. In order to determine reconfigurations, we have to calculate a corresponding diagnosis Δ (see Definition 2).

Definition 2 (Diagnosis). A diagnosis Δ (correction subset) is a subset of $S = \{s_1 : v_1 = a_1, s_2 : v_2 = a_2, \dots, s_n : v_n = a_n\}$ such that $S - \Delta \cup C \cup R_\rho$ is consistent. Δ is minimal if there does not exist a diagnosis Δ' with $\Delta' \subset \Delta$.

On the basis of the definition of a minimal diagnosis, we can introduce a formal definition of a reconfiguration task.

Definition 3 (Reconfiguration Task and Reconfiguration). A reconfiguration task can be defined as a CSP (V, D, C, S, R_ρ) where V is a set of variables, D represents variable domain definitions, C is a set of constraints, S represents an existing configuration, and $R_\rho = \{r'_1, r'_2, \dots, r'_k\}$ (R_ρ consistent with C) represents a set of reconfiguration requirements. A reconfiguration is a variable assignment $S_\Delta = \{s_1 : v_1 = a'_1, s_2 : v_2 = a'_2, \dots, s_l : v_l = a'_l\}$ where $s_i \in \Delta$, $a'_i \neq a_i$, and $S - \Delta \cup S_\Delta \cup C \cup R_\rho$ is consistent.

If R_ρ is inconsistent with C , the new requirements have to be analyzed and changed before a corresponding reconfiguration task can be triggered [4, 8]. An example of a reconfiguration task in the context of our configuration knowledge base is the following.

Example (Reconfiguration Task). In the resource allocation problem, the original customer requirements R are substituted by the requirements $R_\rho = \{r'_1 : \text{pc} = \text{games}, r'_2 : \text{paper} = 3\}$. The resulting reconfiguration task instance is the following.

- $V = \{\text{fuel}, \text{paper}, \text{fireworks}, \text{pc}, \text{games}, \text{oil}, \text{roof}, \text{pipes}\}$
- $\text{dom}(\text{fuel}) = \text{dom}(\text{paper}) = \text{dom}(\text{fireworks}) = \text{dom}(\text{pc}) = \text{dom}(\text{games}) = \text{dom}(\text{oil}) = \text{dom}(\text{roof}) = \text{dom}(\text{pipes}) = \{1, 2, 3\}$
- $C = \{c_1 : \text{fireworks} \neq \text{fuel}, c_2 : \text{fireworks} \neq \text{paper}, c_3 : \text{fireworks} \neq \text{oil}, c_4 : \text{pipes} = \text{roof}, c_5 : \text{paper} \neq \text{fuel}\}$
- $S = \{s_1 : \text{pc} = 3, s_2 : \text{games} = 1, s_3 : \text{paper} = 2, s_4 : \text{fuel} = 3, s_5 : \text{fireworks} = 1, s_6 : \text{oil} = 2, s_7 : \text{roof} = 1, s_8 : \text{pipes} = 1\}$
- $R_\rho = \{r'_1 : \text{pc} = \text{games}, r'_2 : \text{paper} = 3\}$

To solve a reconfiguration task (see Definition 3), conflict-directed diagnosis approaches [19] would determine a set of minimal conflicts and then determine a hitting set that resolves each of the identified conflicts. In this context, a minimal conflict set $CS \subseteq S$ is a minimal set of variable assignments that trigger an inconsistency with $C \cup R_\rho$, i.e., $CS \cup C \cup R_\rho$ is inconsistent and there does not exist a conflict set CS' with $CS' \subset CS$. In our working example, the minimal conflict sets are $CS_1 : \{s_1 : \text{pc} = 3, s_2 : \text{games} = 1\}$, $CS_2 : \{s_3 : \text{paper} = 2\}$, and $CS_3 : \{s_4 : \text{fuel} = 3\}$. The corresponding minimal diagnoses are $\Delta_1 : \{s_1, s_3, s_4\}$ and $\Delta_2 : \{s_2, s_3, s_4\}$. The elements in a diagnosis indicate which variable assignments have to be adapted such that a reconfiguration can be determined that takes into account the new requirements in R_ρ . If we choose Δ_1 , the reconfigurations (reassignments) for the variable assignments in Δ_1 can be determined by a CSP solver call $C \cup R_\rho \cup (S - \Delta_1)$. The resulting configuration S' can be $\{s_1 : \text{pc} = 1, s_2 : \text{games} = 1, s_3 : \text{paper} = 3, s_4 : \text{fuel} = 2, s_5 : \text{fireworks} = 1, s_6 : \text{oil} = 2, s_7 : \text{roof} = 1, s_8 : \text{pipes} = 1\}$. For

a detailed discussion of conflict-based diagnosis we refer to Reiter [19]. In the following we introduce an approach to the determination of minimal reconfigurations which is based on a *direct diagnosis* algorithm, i.e., diagnoses are determined without the need of determining related minimal conflict sets.

4 Reconfiguration with FLEXDIAG

In the following discussions, the set $AC = C \cup R_\rho \cup S$ represents the union of all constraints that restrict the set of possible solutions for a given reconfiguration task. Furthermore, S represents a set of constraints that are considered as candidates for being included in a diagnosis Δ . The idea of FLEXDIAG (Algorithm 1) is to systematically filter out the constraints that become part of a minimal diagnosis using a divide-and-conquer based approach.

Algorithm 1 – FLEXDIAG.

```

1 func FLEXDIAG( $S, AC = C \cup R_\rho \cup S$ ) :  $\Delta$ 
2 if  $isEmpty(S)$  or  $inconsistent(AC - S)$  return  $\emptyset$ 
3 else return FLEXD( $\emptyset, S, AC$ );

4 func FLEXD( $D, S = \{s_1..s_q\}, AC$ ) :  $\Delta$ 
5 if  $D \neq 0$  and  $consistent(AC - S_1)$  return  $\emptyset$ ;
6 if  $size(S) \leq m$  return  $S$ ;
7  $k = \frac{q}{2}$ ;
8  $S_1 = \{s_1..s_k\}; S_2 = \{s_{k+1}..s_q\}$ ;
9  $D_1 = FLEXD(S_1, S_2, AC - S_1)$ ;
10  $D_2 = FLEXD(D_1, S_1, AC - D_1)$ ;
11 return  $(D_1 \cup D_2)$ ;
```

In our example reconfiguration task, the original configuration $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$ and the new set of customer requirements is $R_\rho = \{r'_1, r'_2\}$. Since $S \cup R_\rho \cup C$ is inconsistent, we are in the need of a minimal diagnosis Δ and a reconfiguration S_Δ such that $S - \Delta \cup S_\Delta \cup R_\rho \cup C$ is consistent. In the following we will show how the FLEXDIAG (Algorithm 1) can be applied to determine a minimal diagnosis Δ .

The FLEXDIAG algorithm is assumed to be activated under the assumption that AC is inconsistent, i.e., the consistency of AC is not checked by the algorithm. If AC is inconsistent but $AC - S$ is also inconsistent, FLEXDIAG will not be able to identify a diagnosis in S ; therefore \emptyset is returned. Otherwise, a recursive function FLEXD is activated which is in charge of determining one minimal diagnosis Δ . In each recursive step, the constraints in S are divided into two different subsets (S_1 and S_2) in order to figure out if already one of these subsets includes a diagnosis. If this is the case, the second set must not be inspected for diagnosis elements anymore.

FLEXDIAG is based on the concepts of FASTDIAG [9], i.e., it returns one diagnosis (Δ) at a time and is complete in the sense that if a diagnosis is contained in S , then the algorithm will find it. A corresponding reconfiguration can be determined by a solver call $C \cup R_\rho \cup (S - \Delta)$. The determination of multiple diagnoses at a time can be realized on the basis of the construction of a HSDAG [19]. If $m = 1$ (see Algorithm 1), the number of consistency checks needed for determining one minimal diagnosis is $2\delta \times \log_2(\frac{n}{\delta}) + 2\delta$ in the worst case [9]. In this context, δ represents the set size of the minimal diagnosis Δ and n represents the number of constraints in solution S .

If $m > 1$, the number of needed consistency checks can be systematically reduced if we accept the tradeoff of possibly loosing the property of diagnosis minimality (see Definition 2). If we allow settings with $m > 1$, we can reduce the upper bound of the number of consistency checks to $2\delta \times \log_2(\frac{2n}{\delta \times m})$ in the worst case. These upper bounds regarding the number of needed consistency checks allow to estimate the worst case runtime performance of the diagnosis algorithm which is extremely important for realtime scenarios. Consequently, if we are able to estimate the upper limit of the runtime needed for completing one consistency check (e.g., on the basis of simulations with an underlying constraint solver), we are also able to figure out lower bounds for m that must be chosen in order to guarantee a FLEXDIAG runtime within predefined time limits.

Table 1 depicts an overview of consistency checks needed depending on the setting of the parameter m and the diagnosis size δ for $|S| = 16$. For example, if $m = 2$ and the size of a minimal diagnosis is $\delta = 4$, then the upper bound for the number of needed consistency checks is 16. If the size of δ increases further, the number of corresponding consistency checks does not increase anymore. Figures 1 and 2 depict FLEXDIAG search trees depending on the setting of granularity parameter m .

δ	m=1	m=2	m=4	m=8
1	10	8	6	4
2	16	12	8	4
4	24	16	8	4
8	32	16	16	16

Table 1. Worst-case estimates for the number of needed consistency checks depending on the granularity parameter m and the diagnosis size δ for $|S| = 16$.

FLEXDIAG determines one diagnosis at a time which indicates variable assignments of the original configuration that have to be changed such that a reconfiguration conform to the new requirements (R_ρ) is possible. The algorithm supports the determination of *leading diagnoses*, i.e., diagnoses that are preferred with regard to given user preferences [9]. FLEXDIAG is based on a strict lexicographical ordering of the constraints in S : the lower the importance of a constraint $s_i \in S$ the lower the index of the constraint in S . For example, $s_1 : pc = 3$ has the lowest ranking. The lower the ranking, the higher the probability that the constraint will be part of a reconfiguration S_Δ . Since s_1 has the lowest priority and it is part of a conflict, it is element of the diagnosis returned by FLEXDIAG. For a discussion of the properties of lexicographical orderings we refer to [9, 15].

5 Evaluation

In order to evaluate FLEXDIAG, we analyzed the two major aspects of (1) *algorithm performance* and (2) *diagnosis quality* in terms of *minimality* and *accuracy*. We analyzed both aspects by varying the value of parameter m . Our hypothesis in this context was that the higher the value of m , the lower the number of needed consistency checks (the higher the efficiency of diagnosis search) and the lower diagnosis quality in terms of the share of diagnosis-relevant constraints returned by FLEXDIAG. Diagnosis quality can, for example, be measured by the degree of minimality of the constraints contained in a diagnosis Δ returned by FLEXDIAG (see Formula 1).

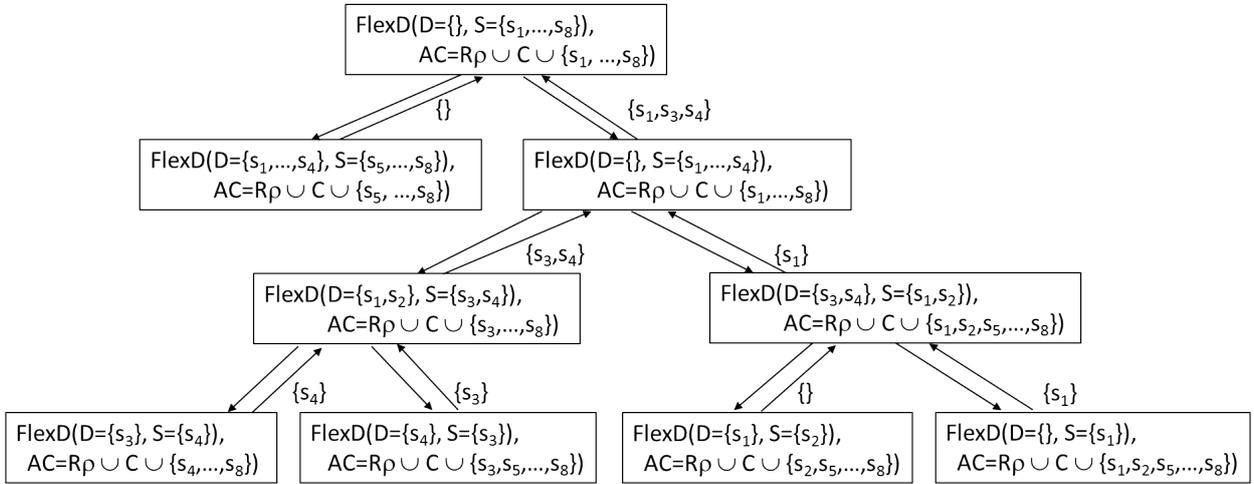


Figure 1. FLEXDIAG: determining one minimal diagnosis with $m = 1$ ($\Delta = \{s_1, s_3, s_4\}$).

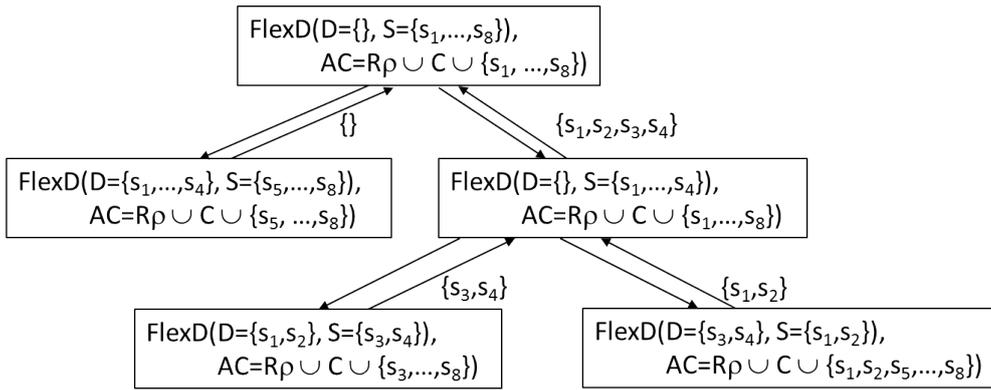


Figure 2. FLEXDIAG: determining a minimal diagnosis with $m = 2$ ($\Delta = \{s_1, s_2, s_3, s_4\}$).

id	V	C	R _ρ	Δ _{min}	avg. runtime (ms)					minimality (accuracy)				
					m=1	m=2	m=4	m=6	m=10	m=1	m=2	m=4	m=6	m=10
1	47	285	5	4	772	647	561	429	350	1.0(1.0)	0.56(1.0)	0.45(1.0)	0.21(1.0)	0.15(1.0)
2	55	73	10	7	359	273	203	180	85	1.0(1.0)	0.55(0.91)	0.31(0.91)	0.30(0.83)	0.41(0.58)
3	73	150	10	5	733	593	343	304	195	1.0(1.0)	0.64(0.91)	0.83(0.55)	0.83(0.55)	1.07(0.30)
4	158	242	20	24	2176	1864	1451	1419	819	1.0(1.0)	0.66(0.89)	0.43(0.82)	0.41(0.67)	0.33(0.65)

Table 2. Avg. runtime and minimality of FLEXDIAG evaluated with feature models from www.splot-research.org (calculation of the first diagnosis: 1: Dell laptops, 2: Smarthomes, 3: Cars, 4: Xerox printers). |R_ρ| is the number of changed requirements and |Δ_{min}| the average cardinality of minimal diagnoses.

$$\text{minimality}(\Delta) = \frac{|\Delta_{\min}|}{|\Delta|} \quad (1)$$

If $m > 1$, there is no guarantee that the diagnosis Δ determined for S is a superset of the diagnosis Δ_{\min} determined for S in the case $m = 1$. Besides minimality, we introduce accuracy as an additional quality indicator (see Formula 2). The higher the share of elements of Δ_{\min} in Δ , the higher the corresponding accuracy (the algorithm is able to reproduce the elements of the minimal diagnosis for $m = 1$).

$$\text{accuracy}(\Delta) = \frac{|\Delta \cap \Delta_{\min}|}{|\Delta_{\min}|} \quad (2)$$

In order to evaluate FLEXDIAG with regard to both aspects we applied the algorithm to the configuration benchmark from www.splot-research.org - the configuration models are feature models which include requirement constraints, compatibility constraints, and different types of structural constraints such as mandatory relationships and alternatives. The feature models were represented as CSP on the basis of the Java-based Choco library.³ For each setting (see Table 2) in the benchmark, we randomly generated |R_ρ| new requirements that were inconsistent with an already determined configuration (10 iterations per setting). The average cardinality of a minimal diagnosis for $m = 1$ is |Δ_{min}|. Related average runtimes (in milliseconds)⁴ and degrees of minimality and accuracy (see Formula 1) are depicted in Table 2. As can be seen in Table 2, increasing the value of m leads to an improved runtime performance in our example cases. Minimality and accuracy depend on the configuration domain and are not necessarily monotonous. For example, since a diagnosis determined by FLEXDIAG is not necessarily a superset of a diagnosis determined with $m = 1$, it can be the case that the *minimality* of a diagnosis determined with $m > 1$ is greater than 1 (if FLEXDIAG determines a diagnosis with lower cardinality than the minimal diagnosis determined with $m = 1$).

Note that in this paper we did not compare FLEXDIAG with more traditional diagnosis approaches – for related evaluations we refer the reader to [9] where detailed related analyses can be found. The outcome of these analyses is that direct diagnosis approaches such as FLEXDIAG clearly outperform standard diagnosis approaches based on the resolution of minimal conflicts [19].

6 Reconfiguration in Production

The following simplified reconfiguration task is related to *scheduling in production* where it is often the case that, for example, schedules and corresponding production equipment has to be reconfigured. In

³ choco-solver.org.

⁴ Test platform: Windows 7 Professional 64 Bit, Intel(R) Core(TM) i5-2320 3.00 GHz CPU with 8.00 GB of memory.

this example setting we do not take into account configurable production equipment (configurable machines) and limit the reconfiguration to the assignment of orders to corresponding machines. The assignment of an order o_i to a certain machine m_j is represented by the corresponding variable $o_i m_j$. The domain of each such variable represents the different possible slots in which an order can be processed, for example, $o_1 m_1 = 1$ denotes the fact that the processing of order o_1 on machine m_1 is performed during and finished after time slot 1.

Further constraints restrict the way in which orders are allowed to be assigned to machines, for example, $o_1 m_1 < o_1 m_2$ denotes the fact that order o_1 must be completed on machine m_1 before a further processing is started on machine m_2 . Furthermore, no two orders must be assigned to the same machine during the same time slot, for example, $o_1 m_1 \neq o_2 m_1$ denotes the fact that order o_1 and o_2 must not be processed on the same machine in the same time slot (slots 1..3). Finally, the definition of our reconfiguration task is completed with an already determined schedule S and a corresponding reconfiguration request represented by the reconfiguration requirement $R_\rho = \{r'_1 : o_3 m_3 < 5\}$, i.e., order o_3 should be completed within less than 5 time units.

- $V = \{o_1 m_1, o_1 m_2, o_1 m_3, o_2 m_1, o_2 m_2, o_2 m_3, o_3 m_1, o_3 m_2, o_3 m_3\}$
- $\text{dom}(o_1 m_1) = \text{dom}(o_2 m_1) = \text{dom}(o_3 m_1) = \{1, 2, 3\}$.
 $\text{dom}(o_1 m_2) = \text{dom}(o_2 m_2) = \text{dom}(o_3 m_2) = \{2, 3, 4\}$.
 $\text{dom}(o_1 m_3) = \text{dom}(o_2 m_3) = \text{dom}(o_3 m_3) = \{3, 4, 5\}$.
- $C = \{c_1 : o_1 m_1 < o_1 m_2, c_2 : o_1 m_2 < o_1 m_3,$
 $c_3 : o_2 m_1 < o_2 m_2, c_4 : o_2 m_2 < o_2 m_3, c_5 : o_3 m_1 < o_3 m_2,$
 $c_6 : o_3 m_2 < o_3 m_3, c_7 : o_1 m_1 \neq o_2 m_1,$
 $c_8 : o_1 m_1 \neq o_3 m_1, c_9 : o_2 m_1 \neq o_3 m_1,$
 $c_{10} : o_1 m_2 \neq o_2 m_2, c_{11} : o_1 m_2 \neq o_3 m_2,$
 $c_{12} : o_2 m_2 \neq o_3 m_2, c_{13} : o_1 m_3 \neq o_2 m_3,$
 $c_{14} : o_1 m_3 \neq o_3 m_3, c_{15} : o_2 m_3 \neq o_3 m_3\}$
- $S = \{s_1 : o_1 m_1 = 1, s_2 : o_1 m_2 = 2, s_3 : o_1 m_3 = 3,$
 $s_4 : o_2 m_1 = 2, s_5 : o_2 m_2 = 3, s_6 : o_2 m_3 = 4,$
 $s_7 : o_3 m_1 = 3, s_8 : o_3 m_2 = 4, s_9 : o_3 m_3 = 5\}$
- $R_\rho = \{r'_1 : o_3 m_3 < 5\}$

This reconfiguration task can be solved using FLEXDIAG. If we keep the ordering of the constraints as defined in S , FLEXDIAG (with $m = 1$) returns the diagnosis $\Delta : \{s_4, s_5, s_6, s_7, s_8\}$ which can be used to determine the new solution $S = \{s_1 : o_1 m_1 = 1, s_2 : o_1 m_2 = 2, s_3 : o_1 m_3 = 3, s_4 : o_2 m_1 = 3, s_5 : o_2 m_2 = 4, s_6 : o_2 m_3 = 5, s_7 : o_3 m_1 = 2, s_8 : o_3 m_2 = 3, s_9 : o_3 m_3 = 4\}$. Possible ordering criteria for constraints in such rescheduling scenarios can be, for example, customer value (changes related to orders of important customers should occur with a lower probability) and the importance of individual orders. If some orders in a schedule should not be changed, this can be achieved by simply defining such requests as requirements (R_ρ), i.e., change requests as well as stability

requests can be included as constraints r_i^j in R_ρ .

7 Ongoing And Future Work

We are currently evaluating FLEXDIAG with a more complex (industrial) benchmark from three different German car manufacturers on the level of type series. In this context we include further evaluation metrics that help to better estimate the quality of diagnoses (reconfigurations) – for example, the currently applied accuracy metric does not take into account the importance of the different constraints contained in a diagnosis. Furthermore, we will extend the FLEXDIAG algorithm in order to make it applicable in scenarios where knowledge bases are tested [6]. Our goal in this context is to improve the performance of existing automated debugging approaches and to investigate to which extent diagnoses resulting from $m > 1$ are considered as relevant by knowledge engineers. Finally, we will compare FLEXDIAG with local search approaches such as genetic algorithms.

8 Conclusions

Efficient reconfiguration functionalities are needed in various scenarios such as the reconfiguration of production schedules, the reconfiguration of the settings in mobile phone networks, and the reconfiguration of robot context information. We analyzed the FLEXDIAG algorithm with regard to potentials of improving existing direct diagnosis algorithms. When using FLEXDIAG, there is a clear tradeoff between performance of diagnosis calculation and diagnosis quality (measured, for example, in terms of minimality and accuracy).

REFERENCES

- [1] R. Bakker, F. Dikker, F. Tempelman, and P. Wogmim, 'Diagnosing and solving over-determined constraint satisfaction problems', in *13th International Joint Conference on Artificial Intelligence*, pp. 276–281, Chambery, France, (1993).
- [2] S.K. Card, G.G. Robertson, and J.D. Mackinlay, 'The information visualizer, an information workspace', in *CHI '91 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 181–188, New Orleans, Louisiana, USA, (1991). ACM.
- [3] J. DeKleer, 'Using crude probability estimates to guide diagnosis', *AI Journal*, **45**(3), 381–391, (1990).
- [4] A. Falkner, A. Felfernig, and A. Haag, 'Recommendation Technologies for Configurable Products', *AI Magazine*, **32**(3), 99–108, (2011).
- [5] A. Falkner and H. Schreiner, 'SIEMENS: Configuration and Reconfiguration in Industry', in *Knowledge-based Configuration – From Research to Business Cases*, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, chapter 16, 251–264, Morgan Kaufmann Publishers, (2013).
- [6] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner, 'Consistency-based diagnosis of configuration knowledge bases', *Artificial Intelligence*, **152**(2), 213–234, (2004).
- [7] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*, Elsevier/Morgan Kaufmann Publishers, 1st edn., 2014.
- [8] A. Felfernig, M. Schubert, G. Friedrich, M. Mandl, M. Mairitsch, and E. Teppan, 'Plausible repairs for inconsistent requirements', in *21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pp. 791–796, Pasadena, CA, USA, (2009).
- [9] A. Felfernig, M. Schubert, and C. Zehentner, 'An efficient diagnosis algorithm for inconsistent constraint sets', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM)*, **26**(1), 53–62, (2012).
- [10] A. Fijany and F. Vatan, 'New approaches for efficient solution of hitting set problem', in *Winter International Symposium on Information and Communication Technologies*, pp. 1–6, Cancun, Mexico, (2004). Trinity College Dublin.
- [11] Gerhard Fleischanderl, Gerhard E. Friedrich, Alois Haselböck, Herwig Schreiner, and Markus Stumptner, 'Configuring large systems using generative constraint satisfaction', *IEEE Intelligent Systems*, **13**(4), 59–68, (1998).
- [12] G. Friedrich, A. Ryabokon, A. Falkner, A. Haselböck, G. Schenner, and H. Schreiner, '(Re)configuration using Answer Set Programming', in *IJCAI 2011 Workshop on Configuration*, pp. 17–24, (2011).
- [13] L. Hotz, A. Felfernig, M. Stumptner, A. Ryabokon, C. Bagley, and K. Wolter, 'Configuration Knowledge Representation & Reasoning', in *Knowledge-based Configuration – From Research to Business Cases*, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, chapter 6, 59–96, Morgan Kaufmann Publishers, (2013).
- [14] D. Jannach, 'Finding preferred query relaxations in content-based recommenders', in *3rd Intl. IEEE Conference on Intelligent Systems*, pp. 355–360, London, UK, (2006).
- [15] Ulrich Junker, 'QUICKPLAIN: preferred explanations and relaxations for over-constrained problems', in *19th Intl. Conference on Artificial Intelligence (AAAI'04)*, eds., Deborah L. McGuinness and George Ferguson, pp. 167–172. AAAI Press, (2004).
- [16] A. Mackworth, 'Consistency in Networks of Relations', *Artificial Intelligence*, **8**(1), 99–118, (1977).
- [17] J. Marques-Silva, F. Heras, M. Janota, A. Previt, and A. Belov, 'On computing minimal correction subsets', in *IJCAI 2013*, pp. 615–622, Peking, China, (2013).
- [18] I. Nica, F. Wotawa, R. Ochenbauer, C. Schober, H. Hofbauer, and S. Boltek, 'Kapsch: Reconfiguration of Mobile Phone Networks', in *Knowledge-based Configuration – From Research to Business Cases*, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, chapter 19, 287–300, Morgan Kaufmann Publishers, (2013).
- [19] R. Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **32**(1), 57–95, (1987).
- [20] M. Schubert and A. Felfernig, 'BFX: Diagnosing Conflicting Requirements in Constraint-based Recommendation', *International Journal on Artificial Intelligence Tools*, **20**(2), 297–312, (2011).
- [21] I. Shah, 'Direct algorithms for finding minimal unsatisfiable subsets in over-constrained csp's', *International Journal on Artificial Intelligence Tools*, **20**(1), 53–91, (2011).
- [22] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin, 'Formal methods for the validation of automotive product configuration data', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **17**(1), 75–97, (2003).
- [23] G. Steinbauer, M. Mörth, and F. Wotawa, 'Real-Time Diagnosis and Repair of Faults of Robot Control Software', in *RoboCup 2005*, LNAI, pp. 13–23. Springer, (2005).
- [24] M. Stumptner, 'An Overview of Knowledge-based Configuration', *AI Communications*, **10**(2), 111–126, (1997).
- [25] M. Stumptner and F. Wotawa, 'Reconfiguration using model-based diagnosis', in *10th International Workshop on Principles of Diagnosis (DX-99)*, pp. 266–271, (1999).
- [26] J. Tiihonen and A. Anderson, 'VariSales', in *Knowledge-based Configuration – From Research to Business Cases*, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, chapter 26, 377–388, Morgan Kaufmann Publishers, (2013).
- [27] J. Tiihonen, W. Mayer, M. Stumptner, and M. Heiskala, 'Configuring Services and Processes', in *Knowledge-based Configuration – From Research to Business Cases*, eds., A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, chapter 21, 313–324, Morgan Kaufmann Publishers, (2013).
- [28] R. Walter and W. Küchlin, 'ReMax - A MaxSAT aided Product (Re-) Configurator', in *Workshop on Configuration 2014*, pp. 55–66, (2014).
- [29] R. Walter, C. Zengler, and W. Küchlin, 'Applications of maxsat in automotive configuration', in *Workshop on Configuration 2013*, pp. 21–28, (2013).
- [30] K. Wang, Z. Li, Y. Ai, and Y. Zhang, 'Computing Minimal Diagnosis with Binary Decision Diagrams Algorithm', in *6th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD'2009)*, pp. 145–149, (2009).
- [31] F. Wotawa, 'A variant of reiter's hitting-set algorithm', *Information Processing Letters*, **79**(1), 45–51, (2001).