

A Static Model-Based Engine for Model-Based Reasoning

Peter Fröhlich and Wolfgang Nejdl

University of Hannover

Lange Laube 3, 30159 Hannover, Germany

{froehlich, nejdl}@kbs.uni-hannover.de

Abstract

Most systems for model-based reasoning record justifications during the search for solutions. In the popular ATMS systems this can lead to an explosion of recorded information when solving complicated problems. We propose an engine for model-based reasoning which works directly on logical models, uses static precompiled information on the structure of the underlying theory and does no further recording during search. This engine (DRUM-II) solves large complicated problems with attractive time and space complexity. To demonstrate the efficiency of the engine we give a new characterization of a popular benchmark suite, solve it with our engine and compare the performance to previous results.

1 Introduction

Since Reiter's seminal paper [Reiter, 1987], several generic systems for model-based diagnosis have been developed using logical inference, assumption based truth maintenance and conflicts as their underlying principles (see [de Kleer and Williams, 1987] and many others). Efficiency problems due to administration overhead inherent to this approach have only recently been solved [Raiman *et al.*, 1993].

DRUM-II follows a different line of research which investigates reasoning through direct manipulation of logical models. Chou and Winslett [Chou and Winslett, 1994] have developed model-based algorithms for the minimal change semantics. Fröhlich and Nejdl took up the ideas of Chou and Winslett and developed algorithms for minimal model computation and query answering under circumscription [Fröhlich and Nejdl, 1996].

In this paper we present a simpler and more efficient variant of this minimal model computation algorithm and apply it to diagnosis. First we introduce the basic algorithm which performs local search to find minimal models of the system description (and thus minimal diagnoses). Although this algorithm is already efficient enough for a wide range of applications (like alarm management in telecom networks) large

complex problems require additional focusing techniques. In most systems focusing is supported by recording actual dependencies among the literals in the system description during runtime. We introduce a new approach in which possible dependencies among the literals are detected statically during the compilation of the system description and used for focusing the algorithm. To demonstrate the efficiency of the engine we give a new characterization of a popular benchmark suite and discuss why the examples are hard to solve. We run our engine on these examples and compare its performance to previous results.

2 The Model-Based Framework

2.1 Relating Diagnoses and System Models

The device under consideration is described by a logical system description SD , denoted as a set of clauses. To avoid confusion we use the term model only for logical Herbrand models represented by sets of atoms in this paper (and not for the system description). The observed behaviour is characterized by a set of observations OBS . As usual, the set of system components is called $COMP$. With each component $c \in COMP$ we associate at least two behavioral modes: a correct mode denoted by $Ok(c)$ and an abnormal mode denoted by $Ab(c) \equiv \neg Ok(c)$. Our framework is suitable for expressing a wide range of diagnostic concepts. First we reconstruct Reiter's concept of minimal diagnosis.

Definition 1 A Diagnosis of $(SD, COMP, OBS)$ is a set $\Delta \subseteq COMP$, such that $SD \cup OBS \cup \{Ab(c) | c \in \Delta\} \cup \{\neg Ab(c) | c \in COMP - \Delta\}$ is consistent. Δ is called a Minimal Diagnosis, iff it is the minimal set (wrt. $\subseteq$) with this property.

Next we characterize this diagnosis concept by the logical models of $SD \cup OBS$. Given a model M of $SD \cup OBS$, the extension of the Ab -predicate (denoted by $M|Ab|$) tells us which components are considered faulty by M . Thus, every model M of $SD \cup OBS$ corresponds to a (possibly non-minimal) diagnosis $M|Ab|$. To characterize the minimal diagnoses using models, we define an order $<_{Ab}$ on the models, based on the abnormal components they contain.

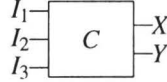
Definition 2 Given models M_1 and M_2 , we define $M_1 \leq_{Ab} M_2$, iff $M_1|Ab| \subseteq M_2|Ab|$ and $M_1 \sim_{Ab} M_2$, iff $M_1|Ab| = M_2|Ab|$. Furthermore we define $M_1 <_{Ab} M_2$, iff $M_1 \leq_{Ab} M_2$ and not $M_1 \sim_{Ab} M_2$.

The minimal diagnoses of $(SD, COMPS, OBS)$ correspond to the $\leq_{Ab}$ -minimal models of $SD \cup OBS$.

Proposition 3 A set Δ of components is a minimal diagnosis of $(SD, COMP, OBS)$, iff there exists a $\leq_{Ab}$ -minimal model M of $SD \cup OBS$, such that $\Delta = M|Ab|$.

The following example shows that there can be several system models corresponding to a single diagnosis.

Consider a system component C (an integrated digital circuit) with two input ports I_1, I_2 and one output port Y .



Suppose the system description predicts the values $X = 1$ and $Y = 0$, given the current values of I_1, I_2 and I_3 . If we observe $Y = 1$, we have to assume that C is faulty. Since C is behaving abnormally, we cannot predict the value of X . Thus we have the following two $\leq_{Ab}$ -minimal models corresponding to the minimal diagnosis $\{C\}$: $M_1 = \{Value(X, 0), Value(Y, 1), Ab(C)\}$ and $M_2 = \{Value(X, 1), Value(Y, 1), Ab(C)\}$.

M_1 and M_2 are equivalent ($M_1 \sim_{Ab} M_2$), because they correspond to the same diagnosis. DRUM-II therefore computes only one model out of every $\sim_{Ab}$ -equivalence class. Such a representative set of minimal models is called a *Transversal*.

Definition 4 Let $\mathcal{M}$ be a set of models and $\mathcal{M}/\sim_{Ab} = \{[M] | M \in \mathcal{M}\}$ be a set of equivalence classes. A set $\mathcal{M}'$ is called a *Transversal* of $\mathcal{M}/\sim_{Ab}$, iff $\mathcal{M}'$ contains exactly one member out of every equivalence class in $\mathcal{M}/\sim_{Ab}$.

The following theorem shows that computing a transversal of the minimal models is a correct implementation of model-based diagnosis.

Theorem 5 Let $\mathcal{M} := \{M | M \models SD \cup OBS \text{ and } \nexists N : (N \models SD \cup OBS \text{ and } N <_{Ab} M)\}$ be the set of all $\leq_{Ab}$ -minimal models. Let $\mathcal{M}'$ be a transversal of $\mathcal{M}/\sim_{Ab}$. Then $D := \{M|Ab| | M \in \mathcal{M}'\}$ is the set of all minimal diagnoses of $(SD, COMP, OBS)$.

While previous results on the relation of diagnosis and circumscription [Besnard and Cordier, 1994; Raiman, 1990] focus on formalizing stronger forms of explanation (than provided by consistency-based diagnosis), this theorem establishes an interesting connection between consistency-based diagnosis and circumscription, because $\mathcal{M}$ is the set of all models obtained by circumscribing $SD \cup OBS$ in Ab , while varying all other predicates.

2.2 Repairing Inconsistent Models

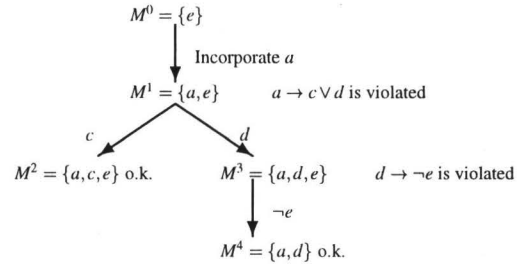
The following algorithm was motivated by Chou and Winslett, who have implemented a system (IMMORTAL) for

model-based belief revision [Chou and Winslett, 1994]. Consider a system description SD and a model M which contains no abnormals, such that $M \models SD$. Circuits, for example, can be modeled by axioms describing the function of their gates. We split the observations into observations of input and output values respectively $OBS = OBS_{in} \cup OBS_{out}$. A model M^0 of the correctly functioning circuit can be obtained by simulating the circuit's correct behavior given the input observations. More generally, monotonic first order logics can be used to construct a model of $SD \cup \{Ok(C) | C \in Comp\} \cup OBS_{in}$. We denote the model which is obtained by incorporating the observations OBS into the model M^0 by $Ins(OBS, M^0)$. It is obtained by inserting the positive literals from OBS into M^0 and deleting the negative literals in OBS from M^0 .¹

Definition 6 For a model M we define $Ins(OBS, M) := OBS \cup M \setminus \{x | \neg x \in OBS\} \setminus \{\neg x | \neg x \in OBS\}$

The changed model $M^1 = Ins(OBS, M^0)$ possibly contradicts some of the axioms in the system description. Our algorithm however uses the inconsistent model M^1 to construct a model of $SD \cup OBS$. It does so by inverting the truth values of literals in M^1 which contradict the axioms in SD .

Example 7 Consider $SD = \{a \rightarrow c \vee d, d \rightarrow \neg e\}$, in clause form $\{\{a, c, d\}, \{\neg d, \neg e\}\}$ together with the initial model $M^0 = \{e\}$ and the observations $OBS = \{a\}$.



The model M^1 contradicts the axiom $a \rightarrow c \vee d$. We can change it in two ways in order to make the axiom hold again. Either we assume c or d , because these are the other literals in the corresponding clause, which can make it true again. While assuming c creates a consistent model (M^2) of $SD \cup OBS$, assuming d violates the axiom $d \rightarrow \neg e$. Thus, we have to delete e in order to obtain the second model M^4 .

To formalize the iterative model repair algorithm motivated in the above example, we need the concept of *Committed Literals*. The positive and negative literals, which have been inverted in model M^i compared to the initial model M^0 are called committed literals $Comm(M^i)$.

Definition 8

$Comm(M^i) := \{l | \text{literal } l \text{ and } M^i \models l \text{ and } M^0 \not\models l\}$

¹ Note, that inserting OBS into M^0 has the same effect as inserting only OBS_{out} , since in our case M^0 already reflects the input observations. In general, any model without abnormals can serve as initial model, so the notation $Ins(M, OBS)$ is more general.

```

Repair_Model( $SD, M^0, OBS$ )
  Solutions :=  $\emptyset$ ;
   $M^1 := Ins(OBS, M^0)$ ;
  Models :=  $\{M^1\}$ ;
  WHILE Models  $\neq \emptyset$  DO
     $M := Select(Models)$ ;
    Models := Models  $\setminus \{M\}$ ;
    IF  $M \models SD \cup OBS$  THEN
      Solutions := Solutions  $\cup \{M\}$ ;
    ELSE
      FOR ALL Flippings  $f$  DO
        Models := Models  $\cup \{Ins(\neg f, M)\}$ ;
      END FOR;
    END IF;
  END WHILE;

```

Figure 1: Model Repair Algorithm

In our example $Comm(M^3) = \{a, d\}$. To avoid cycles in the model repair algorithm, a committed literal is not considered for inversion again. We consider a literal for inversion, if it occurs in a violated clause and neither the literal itself nor its negative counterpart are committed. Such a literal is called a *Flipping*.

Definition 9 Let M^i be a model and l a literal. l is called a *Flipping*, iff

1. There exists a clause C in SD such that $l \in C$ and $M^i \not\models C$ and
2. $l, \neg l \notin Comm(M^i)$.

Our iterative repair algorithm, which is summarized in figure 1 iteratively applies all possible flippings to the inconsistent models.²

2.3 Obtaining Minimal Models

During the repair process we maintain a set of partially repaired models (The set *Models* in figure 1). We have to select a model from this set and apply the next repair step to it. Since we want to compute minimal models, it is important to choose a model which is $<_{Ab}$ -minimal.

An earlier solution to the problem of model selection, which we presented in [Fröhlich and Nejdl, 1996] and applied successfully to reasoning about change-problems, is a best-first algorithm. The list of models is organized as a priority queue. This priority queue is sorted by the number of *Ab*-literals in the models, and so the first element of the priority queue is always $<_{Ab}$ -minimal. Although this model selection policy is sufficiently efficient to solve benchmark problems from the reasoning about action and change domain, it

²Our idea of applying local changes to models is similar to GSAT [Selman et al., 1992]. However, in contrast to GSAT, our method performs a deterministic exhaustive search for minimal models. For this task no probabilistic algorithms have been proposed so far.

proved not suitable for larger diagnostic applications. The main problem here is the management of multiple models and copying of models which is necessary after each repair step.

In DRUM-II we therefore use an iterative deepening algorithm for obtaining the minimal models. For this algorithm the model set is organized as a stack with the usual operations Push and Pop. A stack has a better locality than a priority queue, because all operations only affect its first element. When using a stack, we can implement the push and pop operations without really copying the models. However, since its top element is just some arbitrary model, we must implement other mechanisms to avoid the generation of non-minimal models. The iterative deepening algorithm first defines an initial upper bound *Cutoff* = 1 on the number of *Ab*-literals in a model. If a partially repaired model contains more *Ab*-literals than *Cutoff* it is deleted, but a flag is set to indicate the deletion. The algorithm computes all consistent models with one *Ab*-literal and then sets *Cutoff* to the smallest number of *Ab*-literals of a model which was deleted. The algorithm proceeds until either no model is deleted or the bound *Cutoff* has reached an externally defined maximum (e.g. "only search for single and double faults").

It is easy to see that all models generated by this algorithm are indeed minimal models. In the first iteration we only search for single faults. All single faults found are minimal diagnoses. Then we set the search limit to the next smallest cardinality of a diagnosis which was pruned. During the search we always compare the models to the solutions already found. Thus, no non-minimal diagnoses are generated.

2.4 Formal Properties

The algorithm developed in the previous section is a simpler and more efficient variant of the algorithms presented in [Fröhlich and Nejdl, 1996] and fits in the framework for deciding entailment under circumscription presented in that paper. In the current paper we will only give the results relevant to diagnosis.

Definition 10 Let M and N be sets. $Diff(M, N) := (M \setminus N) \cup (N \setminus M)$

Under the uniqueness of names assumption (UNA) and the domain closure assumption (DCA) (see [Lukasiewicz, 1990]) we can show that the set of models computed by ID.Repair_Model is a superset of those models of $SD \cup OBS$ which differ minimally from the initial model M^0 .

Lemma 11 Let SD be a system description without function symbols, for which we assume UNA and DCA. Let M^0 be a model such that $M^0 \models SD$ and $M^0|Ab| = \emptyset$. Let OBS be a set of literals. Let $\mathcal{M}$ be the set of models returned by ID.Repair_Model(SD, M^0, OBS). Then $\mathcal{M} \supseteq \{M' | M' \models SD \cup OBS \text{ and } Diff(M^0, M') \text{ is minimal}\}$.

Since $M^0|Ab| = \emptyset$ it is easy to show that $\mathcal{M}$ contains a model for each minimal extension of *Ab*. Thus $\mathcal{M}$ contains a transversal of the minimal models and using theorem 5 we

can conclude, that `ID_Repair_Model` is a correct implementation of diagnosis:

Proposition 12 *Let SD, OBS, M^0 and $\mathcal{M}$ be given as in lemma 11. For every diagnosis Δ there is a model $M \in \mathcal{M}$ such that $M|Ab| = \Delta$.*

3 Static Dependencies

We will now present a theory of dependencies between literals in sets of horn clauses. These dependencies (and corresponding independencies) can be used to focus the search for diagnoses. The results presented below are applicable to all system descriptions, where the correct model can be formalized by propositional horn clauses. We do not assume that the whole system description horn since some clauses are typically not horn, e.g. relations among fault models like $Ok \equiv \neg Ab$.

3.1 Exploiting Indepence of Literals

Let us consider a set of Horn Clauses $\mathcal{S}$. Let $\{a_1, \dots, a_n\}$ be the set of atoms occurring in $\mathcal{S}$. We say that a_j depends directly on a_i , if there is a clause containing a_j and $\neg a_i$.

Definition 13 a_j depends directly on a_i , denoted by $a_i \rightarrow^1 a_j$, iff there is a clause $C \in \mathcal{S}$ with $\{a_j, \neg a_i\} \subseteq C$.

We define the transitive closure of the direct dependency relation $\rightarrow^1$ and call it $\rightarrow^+$ (depends on).

Definition 14 The relation $\rightarrow^+$ is defined inductively by

1. if $a_i \rightarrow^1 a_j$, then $a_i \rightarrow^+ a_j$.
2. if $a_i \rightarrow^1 a_k$ and $a_k \rightarrow^+ a_l$, then $a_i \rightarrow^+ a_l$.

We say a_j depends on a_i , iff $a_i \rightarrow^+ a_j$.

We want to study the effect of certain atoms on the other atoms of the theory. In particular, we want to see which changes are caused by assuming different sets A, B of atoms. If we know that an atom a was entailed by SUA (i.e. $SUA \models a$), we want to know if a is still entailed if we assume the set of atoms B instead.

Proposition 15 *Let A and B be sets of atoms such that $S \cup A \not\models \perp$ and $S \cup A \not\models \perp$. Let a be an atom. If $S \cup A \models a$ and there is no $b \in \text{Diff}(A, B)$ such that $b \rightarrow^+ a$, then $S \cup B \models a$.*

Proof Sketch: The positive literal a follows from $S \cup A$, iff it holds in the minimal model of $S \cup A$. This minimal model coincides with the minimal model of $S^+ \cup A$ (S^+ consists of those clauses of $\mathcal{S}$, which contain a positive literal). Thus, $S^+ \cup A \models a$. Thus, there is a proof for a , only based on the clauses in S^+ . If we delete all unnecessary steps from this proof, only those steps remain, which derive a itself or literals on which a depends. This proof remains possible, if we replace A by B since we assume that A and B do not disagree on atoms, on which a depends. Thus, $S^+ \cup B \models a$ and, since $S \cup B$ is consistent, we conclude $S \cup B \models a$. #

The knowledge from the above proposition can be used to focus the search for diagnoses.

3.2 Application to Diagnosis

In diagnosis, the horn clauses $\mathcal{S}$ are given by the system description SD . The following proposition is a special case of proposition 15.

Proposition 16 *Let SD be a system description, $OBS = OBS_{in} \cup OBS_{out}$ an observation, a an atom and Δ a candidate (a set of components considered faulty).*

If $SD \cup OBS_{in} \models a$ and for all $c \in \Delta : Ok(c) \not\models^+ a$, then $SD \cup \{Ok(c) | c \in COMP \setminus \Delta\} \cup OBS \models a$.

To proof this proposition, we must assume, that the observations OBS_{out} are really the outputs of the device, i.e. no further atom depends on the output observations. The above proposition allows us to delete any model, which contains changes not influenced by the current candidate.

Corollary 17 *Let M be a model and $a \in M$ an atom occurring negatively in $\text{Comm}(M^i)$. Let Δ be a candidate. If for all $c \in \Delta : Ok(c) \not\models^+ a$ then $M \not\models SD \cup OBS \cup \{Ok(c) | c \in COMP \setminus \Delta\}$ and thus model M can be deleted.*

The method described above enables us to eliminate a large number of candidates, which do not influence all abnormal outputs of the device as well as a large number of models which contain changes not influenced by the current candidate. The relation $\rightarrow^+$ needs only to be computed once for each system description. We compute $\rightarrow^+$ when the system description changes and store it in a (bit-) matrix, which is loaded into memory by the DRUM-II engine. The space complexity as well as the time complexity for computing the matrix is quadratic in the number of clauses of the underlying system description. The offline computation of the matrix is rather efficient. For a large circuit, the c6288 (see section 4) our current implementation uses approximately one minute to compute $\rightarrow^+$. While we have not yet compared the various optimizations of ATMS-based systems to our approach, a few comparisons can be made already. First, while an ATMS computes actual dependencies of propositions at runtime, our approach computes possible dependencies at compile time. Second, as these possible dependencies are correct for each truth value of the input propositions, they obviously cannot exploit specific values of input propositions. An ATMS-environment is therefore more specific than our dependency set, and can exploit specific input values of the circuit. On the other hand, it seems impossible to precompile ATMS-dependencies since they depend on the actual test vector used. Third, as ATMS's usually store environments as bit vectors, the storage requirement for one environment is the same as for one of our dependency sets. Fourth, while a proposition usually can be deduced in different ways, and the ATMS therefore has to store more than one environment for these propositions, we have exactly one dependency set for each proposition, which avoids any explosion of these dependency sets. An interesting issue for further exploration would be to use our possible dependencies in an ATMS instead of

the specific justifications computed by the ATMS at run-time in order to speed up ATMS-based diagnosis systems.

4 Combinatorial Benchmark Circuits

The ISCAS-85 benchmark suite contains combinatorial circuits with 160 to 3512 components. In recent years it has been regarded as a challenge for diagnostic engines in several papers [de Kleer, 1991; Raiman *et al.*, 1993; Nejdli and Fröhlich, 1997; Williams and Nayak, 1997] due to the size of the problems and their inherent complexity. Before we discuss the performance of DRUM-II on these problems and compare it to previous results, we will give a characterization of the kind of complexity present in these problems and discuss why some of these circuits are hard to solve.

4.1 Why are these Problems so hard?

For satisfiability and constraint satisfaction benchmarks consisting of random problems are widely used. Random problems are often characterized by a set of parameters $\langle n, m, p_1, p_2 \rangle$ [Smith and Dyer, 1996], where n is the number of variables, m is the number of possible values per variable, p_1 is the constraint density (the probability that a constraint exists between two variables) and p_2 is the constraint tightness (the probability that two variables related by a constraint have incompatible values). The constraint tightness is not applicable to the circuits because the constraints (clauses) are not all binary. Other parameters for the ISCAS circuits are given in table 1. The estimated constraint density of the problems decrease when problem size increases³. This suggests that the problems are hard in a dimension not yet captured by random problems or at least by their usual numerical characterization. Figure 2 shows the graph structure of a small part of the c6288 circuit (all successors of a particular component). We can observe the phenomenon of reconvergent fanout, which was already noted by de Kleer [Raiman *et al.*, 1993]: Although a value occurs only in a small number of constraints, there are long chains of constraints, through which nearly all variables in the graph are finally related. For an algorithm this means that it is easy to find a locally consistent solution but after a long computation it will eventually turn out that the solution is not globally consistent. Consequently, the circuits are not easily solvable by brute force search alone and focusing is necessary. Moreover, there is a large number of long propagation paths making straightforward dependency recording algorithms run out of memory.

4.2 Experimental Results

In this section we include experimental results for DRUM-II on the benchmark circuits. We have run 5 test vectors for each circuit. The running time is given in table 2. The times given are elapsed times in seconds on a lightly loaded SUN Ultra-1/170. They do not include model setup time, which is,

³Due to the special structure of the system description the actual constraint density is even lower than these estimates

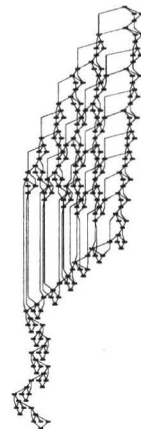


Figure 2: A Small Subgraph of the c6288 Circuit

Circuit	Gates	n	m	p_1
c499	202	894	2	0.02
c880	383	1615	2	0.008
c1355	546	2238	2	0.007
c2670	1193	4928	2	0.002
c3540	1669	6377	2	0.002
c5315	2307	9356	2	0.002
c6288	2406	9696	2	0.002
c7552	3512	13582	2	0.001

Table 1: Statistical Parameters of the Circuits

however, very short (using a domain-specific forward chainer for generating the initial model).

Circuit	Running Time (sec)				
	t1	t2	t3	t4	t5
c499	0.013	0.036	0.040	0.036	0.013
c880	0.023	0.003	0.028	0.027	0.004
c1355	0.047	0.047	0.046	1.255	0.040
c2670	0.061	0.083	0.222	0.077	0.226
c3540	0.236	0.050	0.750	0.143	1.586
c5315	0.025	0.131	0.051	0.080	0.019
c6288	89.517	0.562	23.788	160.623	0.221
c7552	0.056	2.093	1.598	0.126	2.261

Table 2: Running Times of DRUM-II for 5 test vectors

The test vectors used are available from the authors. Note, that the timing results show a large variation depending on the test vector used. This effect is striking with respect to the c6288. A reason for this is the different number of components which can explain the abnormal observation. Table 3 shows the number of propositions (possibly) influencing the abnormal observation for the different test vectors of the c6288. In the difficult cases t1 and t4 nearly the whole circuit

influences the abnormal observation. We expect a large variation in running time for ATMS-based systems, too, though no such results have been published so far.

Circuit	t1	t2	t3	t4	t5
c6288	6316	795	2545	5804	375

Table 3: Propositions influencing the Abnormal Observation

In table 4 we compare the median running time of DRUM-II to previous results from the literature. DRUM-II is always the fastest system except for the c6288 where IMPLode is faster. Since however the test vector used is not given in [Raiman *et al.*, 1993] it is not clear, whether IMPLode's critical reasoning techniques provide still better focusing on the special structure of the c6288 for all test vectors.

Circuit	DRUM-II Median	de Kleer AAAI91	IMPLode IJCAI-93	Williams DX-96
c499	0.036	7.9	0.4	4.5
c880	0.023	6.2	0.8	4.0
c1355	0.047	242	1.4	12.3
c2670	0.083	33	3	28.8
c3540	0.236	1545	6	113.3
c5315	0.051	1215	7	61.2
c6288	23.788	–	8	–
c7552	1.598	1028	14	61.5

Table 4: Comparison of Running Times

5 Conclusion and Further Work

In this paper we describe the reasoning algorithm DRUM-II for diagnosis and other non-monotonic reasoning problems, which is based on the direct manipulation of logical models. It uses static precompiled information on the structure of the underlying theory and does no further recording at runtime. Running DRUM-II on a set of circuit diagnosis benchmarks with more than 3000 components shows that its efficiency is in most cases much better than the performance of previous systems. The paper also contains a short characterization of these benchmarks and their inherent complexity.

Currently, we are extending DRUM to handle stronger definitions of diagnosis (abductive diagnosis and kernel diagnosis). For these optimizations the filtering operator defined in [Fröhlich and Nejd, 1996] is used, which decides, if the truth value of a given literal is entailed.

6 Acknowledgement

We thank Peter Baumgartner as well as an anonymous referee for their contributions to section 3

References

[Besnard and Cordier, 1994] Philippe Besnard and Marie-Odile Cordier. Explanatory diagnoses and their charac-

terization by circumscription. *Annals of Mathematics and Artificial Intelligence*, 11:75–96, 1994.

[Chou and Winslett, 1994] T. S.-C. Chou and M. Winslett. A model-based belief revision system. *Journal of Automated Reasoning*, 12:157–208, 1994.

[de Kleer and Williams, 1987] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[de Kleer, 1991] Johan de Kleer. Focusing on probable diagnoses. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 842–848, Anaheim, July 1991. Morgan Kaufmann Publishers, Inc.

[Fröhlich and Nejd, 1996] Peter Fröhlich and Wolfgang Nejd. A model-based reasoning approach to circumscription. In *Proceedings of the 12th European Conference on Artificial Intelligence*, 1996.

[Lukasiewicz, 1990] W. Lukasiewicz. *Non-monotonic reasoning: formalization of commonsense reasoning*. Ellis Horwood, 1990.

[Nejd and Fröhlich, 1997] Wolfgang Nejd and Peter Fröhlich. Minimal model semantics for diagnosis – techniques and first benchmarks. In *7th International Workshop on Principles of Diagnosis*, Val Morin, Canada, 1997.

[Raiman *et al.*, 1993] Olivier Raiman, Johan de Kleer, and Vijay Saraswat. Critical reasoning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 18–23, Chambéry, August 1993.

[Raiman, 1990] Olivier Raiman. Circumscribing diagnosis engines: Exploiting the alibi principle to avoid blind search. In *Proceedings of the International Workshop on Expert Systems in Engineering*, Vienna, September 1990. Springer-Verlag. Lecture Notes in AI.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[Selman *et al.*, 1992] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 440–446, San Jose, July 1992. Morgan Kaufmann Publishers, Inc.

[Smith and Dyer, 1996] Barbara M. Smith and Martin E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:155–181, 1996.

[Williams and Nayak, 1997] Brian C. Williams and Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *7th International Workshop on Principles of Diagnosis*, Val Morin, Canada, 1997.