A Two-Step Hierarchical Algorithm for Model-Based Diagnosis

Alexander Feldman and **Arjan van Gemund**
Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Mekelweg 4, 2628 CD, Delft, The Netherlands
Tel.: +31 15 2781935, Fax: +31 15 2786632, e-mail: {a.b.feldman,a.j.c.vangemund}@tudelft.nl

Abstract

For many large systems the computational complexity of complete model-based diagnosis is prohibitive. In this paper we investigate the speedup of the diagnosis process by exploiting the hierarchy/locality as is typically present in well-engineered systems. The approach comprises a compile-time and a run-time step. In the first step, a hierarchical CNF representation of the system is compiled to hierarchical DNF of adjustable hierarchical depth. In the second step, the diagnoses are computed from the hierarchical DNF and the actual observations. Our hierarchical algorithm, while sound and complete, allows large models to be diagnosed, where compile-time investment directly translates to run-time speedup. The benefits of our approach are illustrated by using weak-fault models of real-world systems, including the ISCAS-85 combinatorial circuits. Even for these non-optimally partitioned problems the speedup compared to traditional approaches ranges in the hundreds.

Introduction

Fault diagnosis is a computationally very demanding problem. In this paper we study mechanisms for exploiting hierarchy in a divide-and-conquer approach to significantly lower the computational cost. The potential of the hierarchical approach is to reduce the complexity of the diagnosis computation to that of the biggest subsystem in a model (e.g., in a system comprising non-connected subsystems the time for computing a global diagnosis is the sum of the times for diagnosing each subsystem separately). For a class of decomposable systems this may lead to substantial savings in the diagnosis complexity. In this paper we present a hierarchical approach where we demonstrate that real-world systems may have sufficient hierarchy and subsystem independence for such significant gains to be made.

Exploiting hierarchy has been the subject of much work. In (Stumptner & Wotawa 2003), a diagnosis problem is represented as a Constraint Satisfaction Problem (CSP). In this framework the issues of model decomposition and hierarchical diagnosis are discussed. The problem of discovering hierarchies is also treated in (Provan 2001), while (Mozetič 1991) discusses methods for hierarchical abstraction. Implicit system structure to speedup diagnosis is used in (Darwiche 1998). Darwiche introduces the tractable decomposable negation normal form (DNNF) which is a negation normal form with conjuncts not sharing literals. This representation can be compared to our *hierarchical DNF* (disjunctive normal form) although the latter does not impose the restriction on sharing variables at the price of some extra computational time. Our technique differs from the above work in the fact that it allows for a configurable time/space trade-off and formulates the basis for finding an optimal model compilation.

A CSP-based algorithm (Fattah & Dechter 1995) uses a method known as tree-clustering for circuit decomposition. The algorithm qualifies as a strict compilation technique as the resulting representation can be used for diagnosis in time polynomial to its size. Similar to the other strict-compilation approaches this method does not provide an adjustable size of the compiled representation. The algorithm is especially suitable for circuits having nearly-acyclic constraint networks.

Our approach comprises a compile-time and a run-time step, both of which exploit the hierarchy of the models. The input model is assumed to be represented in terms of a *hierarchical CNF* (conjunctive normal form). In the first step the model is compiled to hierarchical DNF where significant speedup is obtained, compared to traditional CNF to DNF compilation (e.g., in (Alexander Feldman & Bos 2005), is achieved a speedup of 549 even for a 2-bit adder). The conversion to DNF may still take exponential time, but this one-only investment is amortized over many diagnoses (observations).

A special feature of the hierarchical CNF compilation is that it can also compile to hierarchical DNF where the depth of the hierarchy can be varied between fully hierarchical and fully expanded ("flat", i.e., ordinary DNF). This feature additionally improves the diagnostic speed at run-time.

In the second step the compiled model, together with the observations is solved, generating the diagnoses. The diagnosis approach taken in this paper is based on an informed search that exploits the hierarchy as far as still present in the pre-processed model. In particular, we have chosen A* using an admissible heuristics based on a-priori health state probabilities such as in Conflict-Directed A* (CDA* (Williams & Ragno 2004)), but other informed strategies are possible. Although the choice for CDA* would further improve performance, in our aim to assess the potential of hierarchy, at present we have focused on

adapting the more simple A* algorithm to our hierarchical framework.

We have compared the performance of the hierarchical algorithm with the performance of a traditional A* diagnosis search. Depending on the structure of the problem we observe speedups up to 270. In the worst case the hierarchical approach performs similar to traditional A*, while speedup of $20 - 90$ is measured for well-decomposed problems.

The rest of the paper is organized as follows. In the second section we introduce concepts and terminology in traditional model-based diagnosis. In the third section we present our compile-time and run-time diagnosis algorithms. In the fourth section we present the performance results. Finally, conclusions and notes for future work are presented.

Non-Hierarchical Diagnosis

We will base the hierarchical diagnosis approach on the well-known model-based diagnosis formalisms introduced by (de Kleer & Williams 1987). These classical approaches in general use one level of hierarchy, i.e., the set of components of which the overall system is built up.

Definition 1 (System). A diagnostic problem *DP* is defined as the ordered triple $DP = \langle SD, COMPS, OBS \rangle$, where *SD* is a set of propositional sentences describing the behavior of the system, *COMPS* is a set of components, contained in the system, and *OBS* is a term stating an observation over some set of "measurable" variables in *SD*.

For brevity, we refer to the classical diagnosis approach as "flat", i.e., non-hierarchical. In the latter technique for each component $c \in COMPS$ there is a corresponding propositional variable h_c representing its health state. We will call these variables h_c *health variables* and every instantiation of $\bigwedge_{c \in COMPS} h_c$ a *health state*.

Definition 2 (Diagnosis). A *diagnosis*[1] for the system $DP = \langle SD, COMPS, OBS \rangle$ is a set $D \subseteq COMPS$ such that $SD \wedge OBS \wedge \left[\bigwedge_{c \in D} \neg h_c\right] \wedge \left[\bigwedge_{c \in (COMPS \setminus D)} h_c\right] \not\models \bot$.

A diagnosis D is a *minimal* if no other diagnosis D', such that $D' \subset D$, exists. A *partial diagnosis* P is such a conjunction of health literals h_c or $\neg h_c$, $c \in COMPS$, that for every other conjunction ϕ which contains P it follows that $SD \wedge OBS \wedge \phi \not\models \bot$. Similarly, a *kernel diagnosis* is a partial diagnosis which is not contained in any other partial diagnosis (de Kleer, Mackworth, & Reiter 1992).

An informed search, such as the non-hierarchical A* algorithm used by us, computes the diagnoses in best-first order, starting with a minimal diagnosis (note that the diagnoses produced subsequently are not necessarily minimal). Thus, not all minimal diagnoses need be computed (the number of all minimal diagnoses can be still exponential in the number of components (Vatan 2002)). The hierarchical search discussed in this paper, however, computers partial diagnoses.

The diagnosis can be split in two phases: a compilation step and a run-time step. The compilation step constitutes a conversion of the system description into (preferably irreducible) DNF. From DNF diagnosis is straightforward as described by Proposition 1.

Proposition 1. Let $DP = \langle SD, COMPS, OBS \rangle$, be a system. Then D is a *partial diagnosis* of *DP* iff the conjunction of all the health variables of the elements in D is an *implicant* of $SD \wedge OBS$.

From Proposition 1 it follows that to perform diagnosis, it is enough to convert *SD* to any DNF and to check each term in the DNF formula for consistency with *OBS*. Furthermore, in order to generate all the minimal diagnoses, it is necessary to compute a minimal cover of the prime implicants of $SD \wedge OBS$ (that is compute an irreducible DNF equivalent to *SD*).

In this paper we assume that all the components are described in CNF. The non-hierarchical approach is illustrated by diagnosing a small circuit, shown in Figure 1.

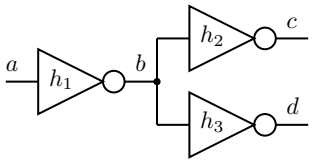

Figure 1: A circuit consisting of three inverters.

For a weak-fault model the corresponding propositional system is given by:

$$SD = \begin{cases} (\neg h_1 \vee a \vee b) \wedge (\neg h_1 \vee \neg a \vee \neg b) \\ (\neg h_2 \vee b \vee c) \wedge (\neg h_2 \vee \neg b \vee \neg c) \\ (\neg h_3 \vee b \vee d) \wedge (\neg h_3 \vee \neg b \vee \neg d) \end{cases} \quad (1)$$

In (1), the health status of the component set $COMPS = \{I_1, I_2, I_3\}$ is given by the health variables h_1, h_2 and h_3. Converting *SD* to an irreducible DNF results in: $\phi = (\neg h_1 \wedge \neg h_2 \wedge \neg h_3) \vee (\neg a \wedge b \wedge \neg h_2 \wedge \neg h_3) \vee (a \wedge \neg b \wedge \neg h_2 \wedge \neg h_3) \vee (\neg b \wedge c \wedge \neg h_1 \wedge \neg h_3) \vee (b \wedge \neg c \wedge \neg h_1 \wedge \neg h_3) \vee (\neg b \wedge d \wedge \neg h_1 \wedge \neg h_2) \vee (b \wedge \neg d \wedge \neg h_1 \wedge \neg h_2) \vee (\neg b \wedge c \wedge d \wedge \neg h_1) \vee (b \wedge \neg c \wedge \neg d \wedge \neg h_1) \vee (\neg a \wedge b \wedge \neg d \wedge \neg h_2) \vee (a \wedge \neg b \wedge d \wedge \neg h_2) \vee (\neg a \wedge b \wedge \neg c \wedge \neg h_3) \vee (a \wedge \neg b \wedge c \wedge \neg h_3) \vee (\neg a \wedge b \wedge \neg c \wedge \neg d) \vee (a \wedge \neg b \wedge c \wedge d)$.

Consider $OBS = a \wedge \neg c \wedge d$ over the observable variables a, c and d. Instantiating ϕ with *OBS* gives us: $\phi \wedge OBS \models (\neg h_1 \wedge \neg h_2 \wedge \neg h_3) \vee (\neg b \wedge \neg h_2 \wedge \neg h_3) \vee (b \wedge \neg h_1 \wedge \neg h_3) \vee (\neg b \wedge \neg h_1 \wedge \neg h_2) \vee (\neg b \wedge \neg h_2)$. Two implicants of $\phi \wedge OBS$ form minimal diagnoses; these are $D_1 = \{\neg h_2\}$ and $D_2 = \{\neg h_1, \neg h_3\}$ (and are the first ones to be generated by A* for equal a priori health probabilities). Note, that in this particular case $D_3 = \{\neg h_1, \neg h_2, \neg h_3\}$ is also a diagnosis but it is not minimal.

The conversion to DNF in the above example can be accomplished off-line, thus demonstrating a non-strict compilation approach.

Hierarchical Diagnosis

In order to present the two steps of our hybrid hierarchical algorithm we introduce the notion of a hierarchical system in terms of a tree-like data structure which includes ordinary "flat" systems in its nodes.

Definition 3 (Hierarchical System). A hierarchical system is a rooted, edge-labeled, acyclic multidigraph $H =$

[1] Throughout this paper we consider consistency-based diagnosis as opposed to abductive diagnosis.

$\langle V, \rho, E\rangle$, where every node V_i, $V_i \in V$, contains a knowledge base SD_i and a set of components $COMPS_i$. The multidigraph is such that $COMPS_1 \cap COMPS_2 \cap \ldots \cap COMPS_n = \emptyset$. The root node is marked by ρ and the labels of the edges in E are maps $f : SD_i \rightarrow SD_j$ between the literals in the knowledge bases represented by the nodes V_i and V_j.

Furthermore, we define hierarchical CNF and hierarchical DNF as hierarchical representation systems with the propositional knowledge base of each node $V_i \in V$ in CNF or DNF respectively. A *hierarchical diagnosis problem* is an ordered pair $HP = \langle H, OBS\rangle$ where *OBS* is a term over some observable variables in H. The size of a hierarchy can be defined in terms of the size of the knowledge-bases in the nodes of the hierarchy: $|H| = \sum_{e\in E} |SD_e|$, where $|SD_e|$ is the size (e.g., the number of terms if *SD* is DNF) of the knowledge-base in the node in which edge e terminates.

In our approach we also consider the depth of a hierarchy d, which is one of its fundamental parameters.

Definition 4. The depth d of a hierarchy $H = \langle V, \rho, E\rangle$ is the number of nodes in the longest path from ρ to any node $v \in V$ such that v is of outdegree 0.

Hierarchical CNF to DNF Compilation

In this section we present the first part of our hybrid technique. Algorithm 1 modifies its input (a hierarchical CNF), producing a hierarchical DNF of adjustable size. The investment of processing time in this algorithm translates to increased compilation size and faster run-time diagnosis in the second part of our technique.

The nested subroutine FLATTENNODE converts a CNF hierarchy to a flat DNF by converting each node to DNF and then multiplying the nodes alongside the multidigraph edges. The function COMPILEDNF transform a node's CNF to DNF. This function is implemented using a slightly modified satisfiability checker (Davis & Putnam 1960).

We use Algorithm 1 in three configurations, depending on the parameter t. Let d be the initial depth of the hierarchy H. If $t = d$, then FLATTENNODE will never descend, and after FLATTEN finishes, H will be the original hierarchy with each of its node converted from CNF to DNF. The result is hierarchical DNF which we denote as DNF/H. On the other extreme we may invoke FLATTEN with parameter $t = 1$. In this case FLATTEN will *fully flatten* H, i.e., the result will have one node only. The result of FLATTEN in this configuration we denote as DNF/F. FLATTEN is mostly used with $1 < t < d$. The result of FLATTEN, then, will be a *partially flattened* DNF (DNF/P) as the depth of the DNF/P is t.

To demonstrate Algorithm 1 we consider a small CNF hierarchy H consisting of three nodes (V_1, V_2 and V_3) and two edges ($e_1 = \langle V_1, V_2\rangle$ and $e_2 = \langle V_1, V_3\rangle$). Let V_1, V_2 and V_3 contain respectively $SD_1 = (\neg h_1 \vee z) \wedge (x \vee z)$, $SD_2 = \neg h_2 \wedge \neg x$ and $SD_3 = (\neg h_3 \vee z) \wedge y$. In this example we will produce DNF/F, hence we invoke FLATTEN with $t = 1$. In this case COMPILEDNF will be immediately invoked on ρ and FLATTEN will never recursively descend. The workings of FLATTENNODE on ρ are described next. The formula SD_1 in node V_1 needs to be converted to DNF, and at the first invocation of FLATTENNODE we get $P = \{\neg h_1 \wedge x, z\}$. The function is invoked recursively for e_1 and as in the second node SD_2 is already in DNF the result is: $Q = \{\neg h_2 \wedge \neg x\}$. Multiplying P and Q results in a set with one consistent term: $R = \{\neg h_2 \wedge \neg x \wedge z\}$. At the second recursive call of FLATTENNODE we convert $SD_3 = (\neg h_3 \vee z) \wedge y$ to DNF which results in $Q = \{\neg h_3 \wedge y, y \wedge z\}$. Multiplying the partial result R by P for the second time results in the final set of terms $R = \{\neg h_2 \wedge \neg h_3 \wedge \neg x \wedge y \wedge z, \neg h_2 \wedge \neg x \wedge y \wedge z\}$.

The first and the second multiplications in the above example are performed in two steps each. The whole transformation takes four steps, which is obviously an improvement over a brute-force enumeration of all possible instantiations over the variables h_1, h_2, h_3, x, y, and z in the flat CNF formula produced from the hierarchy H.

Algorithm 1 Hierarchical model compilation.

```
procedure FLATTEN(H, N, t, r)
    inputs: H = ⟨V, ρ, E⟩, hierarchy
            N, the current node, N ∈ V, initially ρ
            t, integer, maximal depth
            r, integer, current depth, initially 1
    local variables: SD, the CNF in N
    function FLATTENNODE(H, K) returns a DNF
        inputs: H = ⟨V, ρ, E⟩, hierarchy
                K, the current node, K ∈ V
        local variables: SD_1, the CNF in K
                         P, Q, R, term sets, initially ∅
        P ← COMPILEDNF(SD_1)
        for all {e ∈ E : e = V → L} do
5:          Q ← FLATTENNODE(H, L)
            for all {p ∈ P, q ∈ Q : p ∧ q ⊭ ⊥} do
                R ← R ∪ {p ∧ q}
            end for
        end for
10:     return R
    end function
    if t = r then
        SD ← FLATTENNODE(H, N)
        E ← E \ {e ∈ E : e = V → M}
15: else
        SD ← COMPILEDNF(SD_1)
    end if
    for all {e ∈ E : e = V → M} do
        FLATTEN(H, M, t, r + 1)
20: end for
end procedure
```

Complexity of the Hierarchical DNF Compilation

Next we discuss the complexity of the compilation step in relation to two other well-known compilation forms – DNNF (Darwiche 1998) and OBDD (Brace, Rudell, & Bryant 1990). Transforming CNF to DNF is worst-time exponential of the number of variables, hence both the time and space complexity of the compilation phase are $O(2^{|SD_{\max}|})$, where $|SD_{\max}|$ is the node with a knowledge-base having the highest number of variables. A full-flattening ($t = 1$) is equivalent to the worst-case in the traditional compila-

tion approaches, hence the time speed-up in the compilation phase can be exponential to $|SD|/|SD_{\max}|$, where $|SD|$ is the total number of variables in H.

The DNNF and OBDD compilations can, in the worst-case, produce a representation with the number of nodes exponential of the input size. The major advantage of these strict-compilation approaches is that they support queries (e.g., minimal cardinality, model-counting, etc.) in time polynomial to the compiled size, allowing for accurate time-bounds on the online reasoning phase.

Hierarchical DNF is a restricted form of NNF, i.e., it is a conjunction of disjunction of conjunctions. For these hierarchical DNF formulae, decomposability (as defined for DNNF) does not hold, and the performance of the algorithms described here will depend on the number of shared literals between any two hierarchical DNF nodes. It can be shown that hierarchical DNF is more succinct than DNNF, however we will omit a formal proof for a lack of space. This succinctness results in that hierarchical DNF does not support the queries which DNNF supports in polynomial time, hence not qualifying as a strict compilation language.

To analyze the effect of the compilation depth t on the compilation complexity we assume that each node in the original hierarchy H is transformed to DNF in constant time (e.g., by an oracle). In this case, the partial flattening process will increase the size of the nodes exponentially to the number of nodes, counted from the nodes having no incoming edges, that is the worst-case space complexity is $O(2^{d-t})$. In practice, for non-tightly connected models t can assume only values close to the original depth d. For more tightly connected models (e.g., strong-fault models), pruning can be done to a smaller depth, increasing the node-size until a predetermined bound on the number of terms M per node has been reached.

A* Search in a Hierarchical DNF

Next, we proceed with the run-time part based on A*. We assume that components failures are independent and use the *a priori* probability of a fault term to guide a heuristic search for the most likely diagnosis. We assign the same small probability to all the components (de Kleer 1990) as the reasoning technique is not probability driven and it is possible to use other heuristics with similar results (e.g., the cardinality of a fault-mode).

The heuristic function for getting to a node containing a term σ in a tree constructed from a hierarchy with an equivalent DNF, ϕ, is $f(\sigma) = P_1(h_1)P_2(h_2)\dots P_n(h_n)$, where $h_1, h_2, \dots, h_n$ are all the health variables in ϕ. In this case the probability of h_i being true if h_i is in σ is $P(h_i)$ and $P_i(h_i) = 1 - P(h_i)$ if $\neg h_i$ is in σ. If neither h_i nor $\neg h_i$ are present in σ, $P_i(h_i) = \max(P(h_i), 1 - P(h_i))$. The A* algorithm shown below is guided by $f(\sigma)$ when traversing the hierarchy.

In its main loop, Algorithm 2 selects such a term c from the hierarchical node that the heuristic estimate $f(c)$ is maximized. When a consistent conjunction of terms is chosen from all the nodes in the hierarchy, OUTPUTDIAGNOSIS is invoked to send the result to the user.

Algorithm 2 A* search in a hierarchical DNF dictionary.

```
    procedure HIERARCHICALDIAGNOSE(H)
        inputs: H, hierarchical node
        local variables: Q, priority queue
                         s, c, terms
        PUSH(Q, INITIALSTATE(H))
        while (c ←POP(Q)) ≠ ∅ do
            ENQUEUESIBLINGS(Q, c)
 5:         if DIAGNOSIS(c) then
                OUTPUTDIAGNOSIS(c)
            else
                if (s ←NEXTBESTSTATE(H, c)) ⊭⊥ then
                    PUSH(Q, s)
10:             end if
            end if
        end while
    end procedure
```

The auxiliary functions PUSH and POP perform the respective priority queue manipulation on Q (POP returns $\emptyset$ if the queue is empty). The initial state in the search tree, returned by INITIALSTATE, is the empty term. The selection of the next candidate states to be added to the search queue is done by the functions NEXTBESTSTATE and ENQUEUESIBLINGS. The former chooses the child state of the current state c and uses this term s from it, which again maximizes the utility function $f(s)$. To allow for a complete search, ENQUEUESIBLINGS is used to enqueue the siblings of the current node and all its ancestors. Only consistent terms are added to the queue, and OUTPUTDIAGNOSIS should keep track and prevent duplicate diagnoses. A practical approach is to keep the diagnoses in a trie (Forbus & de Kleer 1993).

Let us illustrate the workings of Algorithm 2 on the hierarchical problem $HP = \langle H, OBS \rangle$, with a hierarchy H consisting of three nodes V_1, V_2, and V_3, two edges $e_1 = \{V_1, V_2\}$, $e_2 = \{V_1, V_3\}$ and a root node V_1. The DNF expressions in V_1, V_2, and V_3 are $\phi_1 = \{\}$, $\phi_2 = (h_1 \wedge \neg a \wedge b) \vee (h_1 \wedge a \wedge \neg b) \vee (\neg h_1)$ and $\phi_3 = (h_2 \wedge \neg b \wedge c) \vee (h_2 \wedge b \wedge \neg c) \vee (\neg h_2)$, respectively. Let $OBS = c$ and $P_i(h_i) = 0.95$ for $i = 1, 2$.

After instantiating the terms in V_1, V_2, and V_3 with *OBS* and removing the inconsistent terms we get the search tree in Figure 2.

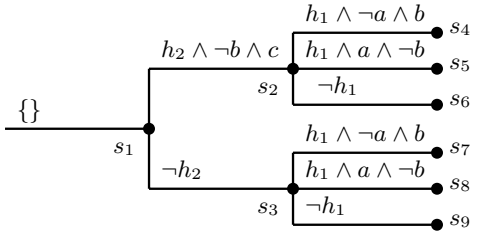

Figure 2: The search tree constructed by Algorithm 2 applied on an example hierarchical diagnosis problem.

Initially, the state s_1 is pushed on the queue. Its best child is s_2 as it has higher probability than s_3: $f(s_2) = 0.95$, while $f(s_3) = 0.05$. When s_2 is popped next, its first child s_4 is skipped as it leads to inconsistency (in the b literal) and s_5

is pushed on the Q. Next s_5 is retrieved (it is the only state on the queue) and it is a leaf node, hence its health variables form a diagnosis. After showing the diagnosis, Algorithm 2 pushes the best consistent siblings of all the predecessors of s_5 on the queue. These are s_6 and s_3. Now s_3 has the the highest probability in the queue, hence it is popped and its best child s_7 is pushed. In the next step s_7 is popped, which is a diagnosis. The process continues until, finally, we pop s_9 from the queue; a state which has the lowest probability diagnosis: $D = \{\neg h_1, \neg h_2\}$.

Typically, the performance of Algorithm 2 is not sensitive to the choice of the a-priori health probabilities for the components. Algorithm 2 works faster for well-decomposed trees (i.e., having a small number of shared variables). In practice, however, this is not always the case and subsystems sharing more variables appear at the low levels of the hierarchy, constraining the number of solutions. Such subsystems are likely to be flattened by the pre-processing part of our hybrid algorithm thus leading to a fast overall diagnosis for systems well decomposed at the top level and constrained at the nodes, appearing close to the leaves of the tree.

Algorithm 2 differs from the traditional A* by the way it constructs its search tree. The depth of the search tree equals the number of subsystems and the order in which they are traversed depends on the hierarchy of the system (i.e., terms of top-level systems show near the root of the search-tree).

Experimental Results

We have derived a diagnosis benchmark from the ISCAS-85 set of combinatorial circuits (Brglez & Fujiwara 1985). Unfortunately, the hierarchy in the original ISCAS-85 circuits has been flattened-out and they have been distributed in a simple netlist format. As the *original high-level design* is a good starting point for model partitioning, we have used the reverse engineered ISCAS-85 circuits (Hansen, Yalcin, & Hayes 1999) instead of the original flat representation. The flattened and the hierarchical ISCAS-85 circuits are, of course, logically equivalent.

Name	Gates	$\|SD\|$	$\|\Delta\|$	$\|OBS\|$	$\|V\|$	$\|E\|$	$\|T\|$
74182	19	47	75	14	9	19	20
74283	40	89	130	14	16	43	44
74L85	41	93	134	14	12	39	46
74181	62	138	216	22	18	66	67
c432	146	328	486	43	13	151	152
c499	202	445	714	73	9	204	205
c880	383	826	1 112	86	10	383	384
c1355	514	1 069	1 546	73	10	208	621
c1908	252	541	911	58	25	237	268
c2670	983	2 153	2 856	226	47	314	1 174
c3540	1 297	2 685	3 861	72	59	382	1 550
c5315	2 202	4 796	6 983	295	66	294	2 738
c6288	2 416	4 864	7 216	64	4	2 416	2 417
c7552	3 024	6 390	9 085	325	71	359	3 806

Table 1: Basic characteristics of the weak-fault diagnosis models based on the 74XXX and ISCAS-85 combinatorial circuits.

The basic characteristics of the high-level ISCAS-85 models are shown in Table 1. We have counted the number of variables $|SD|$ and the number of clauses in the CNF $|\Delta|$. The number of observable variables is denoted as $|OBS|$. Next to the CNF characteristics are shown the fundamental DNF/H metrics $|V|$ and $|E|$ measuring the number of nodes and edges respectively.

The high-level ISCAS-85 circuits are available in the Verilog HDL format and as our algorithms are implemented in the LYDIA[2] framework, we have implemented a Verilog to LYDIA translator. In addition to the original ISCAS-85 models we have added four circuits from the 74X family. The latter are smaller than the ISCAS-85 models and provide additional reference points for the fault-diagnosis experiments allowing some exhaustive searches.

The ISCAS-85 specification does not provide for a fault-modeling, hence we use our own standard logic component libraries allowing every gate-level component to fail. Throughout this paper we have used weak-fault models of the components.

The original high-level combinatorial circuits provide a top-level module whose *only function* is to re-assign all the input and output pins. As this pin-assignment does not change the solutions of the circuit and it is trivial to extend the algorithms for explicitly handling such assignment lists by moving them away in the beginning and renaming the variables in the solutions according to the stored assignment list, we have removed these top-level modules from all the circuits.

All the experiments described in this paper are performed on a 1.86 GHz Pentium M CPU.

Compile-Time Flattening Trade-Offs

The effect of the pre-processing step (Algorithm 1) on the representation size and the time for compilation is shown in Table 2. The three representations on which we perform diagnostic search are constructed as follows. The DNF/H models are the original high-level ISCAS-85 models converted to hierarchical DNF (each leaf-node pushed down to the same depth d). By partially flattening to a depth $d-1$ we obtain DNF/P_1 and by flattening to $d-2$ we obtain DNF/P_2.

In Table 2 we can see the compilation times for converting hierarchical CNF to hierarchical DNF and the time necessary for partial flattening. This time is denoted as T_c. The sum of the terms in each of the nodes of the hierarchical DNF is denoted as $|\phi_t|$. Note, that for DNF/H we don't have a partial flattening step, hence the compilation time is only the time for converting the hierarchical CNF to hierarchical DNF. Finally, T_d measures the time for finding a leading single-fault diagnosis using hierarchical A*.

By partial flattening to DNF/P_1 we gain speedup by a factor varying from 2 to 5.6 and with the deeper flattening to DNF/P_2 the speed improvement varies from 2.9 to 12.3 (cf. Table 2). For this improvement in speed we pay the price of increasing the representation size in comparison to the original DNF/H. This increase in size in the DNF/P_1 compilations is 1.7 – 2.6 times and in the DNF/P_2 representation is a factor of 5.9 – 49.7.

[2]The LYDIA package for model-based fault diagnosis as well as the ISCAS-85 benchmark circuits can be downloaded from `http://www.fdir.org/lydia/`.

	DNF/H			DNF/P_1			DNF/P_2		
	T_c [ms]	$\|\phi_t\|$	T_d [ms]	T_c [ms]	$\|\phi_t\|$	T_d [ms]	T_c [ms]	$\|\phi_t\|$	T_d [ms]
74182	0.43	141	0.56	4.75	323	0.28	662.51	2 431	0.15
74283	0.94	216	1.41	4.74	442	0.4	244.81	3 332	0.3
74L85	0.63	257	1.28	4.73	554	0.65	688.71	6 678	0.45
74181	0.75	364	17.97	8.37	803	3.43	1 391.56	8 326	1.46
c432	1.13	670	61.27	17.16	1 396	10.93	1 484.81	10 579	10
c499	1.31	1 079	12.22	41.92	2 828	3.55	69 673.39	53 653	3.84
c1908	2.31	1 407	20.61	59.44	3 225	7.17	28 881.22	28 544	4.75
c880	1.4	1 826	38.44	39.49	4 055	12.43	2 282.52	29 305	4.51
c1355	1.26	2 327	102.34	74.03	4 167	40.04	68 565.88	43 677	16.36
c2670	3.84	5 027	981.73	101.99	9 572	333.2	3 425.12	36 817	111.47
c3540	4.21	7 021	871.21	153.16	14 070	413.96	7 272.73	82 808	187.32
c5315	54.48	12 534	23 172.46	268.11	22 656	6 786.24	6 129.04	73 576	2 796.49
c6288	4.72	10 145	1 453.99	210.85	17 301	450.89	4 067.29	74 014	123.38
c7552	10.95	16 569	4 264.19	427.74	33 758	1 533.95	93 107.63	213 382	746.95

Table 2: Compilation time, resulting H-DNF size and first single fault diagnosis search time for the benchmark suite models.

The speed of the run-time search is determined by the depth and the quality of partial flattening, which depends on the original hierarchy. The DNF/H representations of the ISCAS-85 circuits, however, are obtained by sequential splitting of large nodes to prevent explosion in the size of the partially flattened DNF. This naïve partitioning is the reason for the mostly linear speedup in T_d.

While model partitioning is a topic on its own, even these preliminary results suggest the existence of an optimal space/time trade-off which we intend to exploit in subsequent research. Experiments with hand-prepared hierarchies (Alexander Feldman & Bos 2005) show speedup growing faster than the model size and in the range of $10^2 - 10^5$.

State of the art compilation techniques like DNNF (Darwiche 2004) report experimental results on model-counting for ISCAS-85. Using the DNNF compiler discussed there we could not compile circuits bigger than c1908 in time less than 15 minutes. The circuits we have used, however, are weak-fault models of ISCAS-85 which makes them bigger than the original ones. In addition to that our approach does not support queries for model-counting which makes the comparison of the two techniques difficult.

Diagnostic Performance Related to the Model Size

We have compared the performance of the traditional A* and hierarchical A* diagnostic engines as a function of the model size by using the circuits in Table 1. Figure 3 shows the performance of the two methods with random observations consistent with zero- and single-fault health. These observations were chosen arbitrarily and do not guarantee worst-case or average performance of the run-time part of the algorithm but are indicative for its potential.

The speedup of the hierarchical approach for nominal diagnoses (Figure 3) is in the range of 0.6 – 7.9. Note, that checking for nominal behavior with the traditional A* algorithm is very efficient since the consistency check is invoked only once. However, in these experiments, due to the fact that all inputs and outputs are assigned, the DPLL check finishes in polynomial time by doing *unit-propagation only*. As a result, our experimental results are conservative in terms of the hierarchical speedup (the latter would increase with

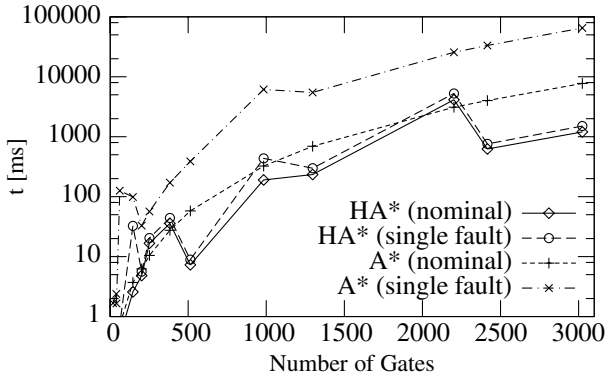

Figure 3: Time for finding a leading diagnosis in the ISCAS-85 circuits by the A* and the hierarchical A* diagnostic engines.

shorter observations). The gain in the hierarchical algorithm comes from the shallower search tree which, in the former case, has a depth equal to the number of nodes $|T|$ in the hierarchical representation (cf. Table 1).

The advantage of the hierarchical algorithm for finding a leading single-fault diagnosis is bigger and varies from 2.8 for c1908 to 170.1 for 74181, being 43.9 and 43.7 for the two biggest models (c6288 and c7552 respectively). In all the cases, the traditional A* algorithm finds a leading fault after a relatively small number of consistency checks (the biggest number of DPLL calls is 41 for the 74181 circuit but is typically between 2 and 4 for the rest of them). The hierarchical technique shows better scalability as the diagnosis time decreases by, e.g., a factor of 1.5 between c2670 and c3540 (while the number of gates increases by a factor of 1.3). Contrary to that behavior, the flat method shows increase in the amount of time for computing diagnosis.

Performance of Multiple-Fault Diagnosis

From the above it is to be expected that the speedup increases with the number of faults diagnosed. We define the *cardinality* of a fault $|D|$ to be the number of elements in a diagnosis D. The leading or minimal fault cardinality (denoted as *MC*) is the cardinality of the fault with the highest

probability estimator P as computed in Algorithm 2.

For the next experiment we have chosen the 74283 circuit and 6 possible sets of observations (consistent with $MC = 0, 1, \ldots 5$). The highest *MC* is the highest possible cardinality of a minimal diagnosis for this model as it contains 5 outputs only and lacks redundancy. Finding those observations that are exclusive for maximal *MC* is computationally demanding and we have succeeded to do this only for the three smallest circuits. Next, we compare the performance of the "traditional" and hierarchical A* implementations in finding these leading diagnoses of variable *MC*. The result is shown in Figure 4.

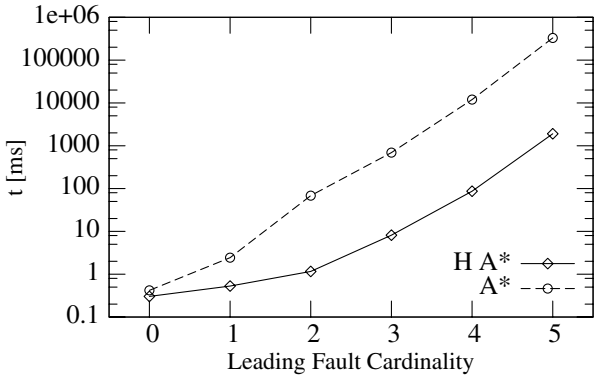

Figure 4: Time for finding a leading-diagnosis of different minimal cardinality with the A* and hierarchical A* algorithms.

The time for finding a multiple-fault diagnosis with the traditional A* and the a-priori health probability heuristics increases exponentially with the cardinality of the diagnosis, which is visible in the performance graph. The worst-case complexity of the hierarchical A* algorithm is the same, but with certain hierarchies, multiple faults can reside in the same partition leading to faster detection. This results in speedups ranging from 1.4 to 173.3 in our 74283 experiment. To detect a minimal fault of cardinality 5, the "flat" A* algorithm takes 786 366 times the time for determining a nominal health. In the hierarchical version, the factor is 6 276, which demonstrates significantly better scalability.

Conclusion

We have described a two-step hierarchical method for computing diagnoses. The first preprocessing step, transforms a hierarchical CNF model of the system to hierarchical DNF of adjustable depth. In the second step, this hierarchical DNF is input to a hierarchical A* search for states consistent with the observation. The heuristic used for the hierarchical A* search is the a-priori probability of a state.

The improved performance of the hierarchical approach over the traditional CNF to DNF conversion and non-hierarchical A* are empirically demonstrated. Experiments, including diagnosis of models based on the ISCAS-85 benchmarks demonstrate a speedup of 2 - 270, typically around 90 for finding a leading single-fault diagnosis. Furthermore, the hierarchical approach scales better in finding multiple-faults as we have shown a speedup of 170 in finding a leading diagnosis of cardinality 5.

Future work includes three main directions. First, we aim at generalizing our technique for allowing a wider range of reasoning queries and comparing the results with state of the art compilation techniques like DNNF. Second, extending our algorithms with state of the art conflict-learning and probabilistic driven reasoning would give us better insight into the merits of the hierarchical approach. Finally, we are interested in optimal model partitioning, and extending the pre-processing step to work on a wider variety of hierarchical representations. In particular, we aim at determining performance bounds based on the connectivity of the input hierarchy for determining optimal hierarchical representation.

Acknowledgments

We extend our gratitude to the anonymous reviewers for their insightful feedback.

References

Alexander Feldman, A. v. G., and Bos, A. 2005. A hybrid approach to hierarchical fault diagnosis. In *Proc. DX'05*, 101–106.

Brace, K.; Rudell, R.; and Bryant, R. 1990. Efficient implementation of a bdd package. In *Proc. DAC'90*, 40–45.

Brglez, F., and Fujiwara, H. 1985. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proc. ISCAS'85*, 695–698.

Darwiche, A. 1998. Model-based diagnosis using structured system descriptions. *JAIR* 8:165–222.

Darwiche, A. 2004. New advances in compiling CNF into DNNF. In *Proc. ECAI'04*, 328–332.

Davis, M., and Putnam, H. 1960. A computing procedure for quantification theory. *JACM* 7(3):201–215.

de Kleer, J., and Williams, B. 1987. Diagnosing multiple faults. *JAI* 32(1):97–130.

de Kleer, J.; Mackworth, A.; and Reiter, R. 1992. Characterizing diagnoses and systems. *Artificial Intelligence* 56(2–3):197–222.

de Kleer, J. 1990. Using crude probability estimates to guide diagnosis. *AI* 45(3):381–291.

Fattah, Y. E., and Dechter, R. 1995. Diagnosing tree-decomposable circuits. In *IJCAI'95*, 1742–1749.

Forbus, K., and de Kleer, J. 1993. *Building Problem Solvers*. MIT Press.

Hansen, M.; Yalcin, H.; and Hayes, J. 1999. Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering. *IEEE Design & Test* 16(3):72–80.

Mozetič, I. 1991. Hierarchical model-based diagnosis. *JMMS* 35(3):329–362.

Provan, G. 2001. Hierarchical model-based diagnosis. In *Proc. DX'01*.

Stumptner, M., and Wotawa, F. 2003. Coupling CSP decomposition methods and diagnosis algorithms for tree structured systems. In *Proc. DX'03*.

Vatan, F. 2002. The complexity of the diagnosis problem. Technical Report NPO-30315, Jet Propulsion Laboratory, California Institute of Technology.

Williams, B., and Ragno, R. 2004. Conflict-directed A* and its role in model-based embedded systems. *JDAM*.