Computing Multi-Scenario Diagnoses

Ingo Pill[1] and **Franz Wotawa**[1]
[1]Christian Doppler Laboratory for Quality Assurance Methodologies for Cyber-Physical Systems
Institute for Software Technology, Graz University of Technology
e-mail: {ipill, wotawa}@ist.tugraz.at

Abstract

Deriving global diagnoses for observations from multiple scenarios is computationally challenging. In this paper, we propose with MSRC-Tree a diagnosis algorithm that considers all scenarios in a single search and compare it to the naive strategy of collecting conflicts for all individual scenarios first and deriving global diagnoses from them in a second step. For ISCAS85 circuits with up to 3 faults and up to twenty failed test cases, we could achieve a speed-up of up to two orders of magnitude, where MSRC-Tree offered better performance also on average and for most scenarios.

1 Introduction

Designing and realizing a system is most often a cumbersome process. Fortunately, there is a variety of techniques that help us along the way. This includes design patterns that assist us in our development steps, formal methods like model-checking [1] that allow us to prove properties of a system's model, or testing concepts and corresponding oracles [2] that allow us to evaluate exemplary system behavior obtained via simulations or physically testing a system. If something goes wrong, model-based diagnosis (MBD) [3; 4] concepts enable us to reason about the origin(s) of encountered problems—the first step in a debugging process.

When deploying MBD, we usually focus on individual scenarios for which a system did not live up to our expectations. That is, first, we consider the behavior observed for some simulation or test and investigate whether it fits our expectations as captured in a system description. If there is an inconsistency, we then reason which subsets of a system's components could explain the malfunction—if we assume those components to be faulty. Usually this will give us more than one such diagnosis. We will then either have to investigate all these diagnoses, or we can implement a probing strategy [4; 3] that will allow us to discriminate between diagnoses via providing new measurements [5; 6]. The latter is referred to as *sequential diagnosis*.

In practice, in many situations we have access to data observed for multiple independent scenarios in the first place. For instance after executing a test suite, or when we simply observe a system operating 24/7 for some specific scenarios. Light-weight diagnosis techniques like Spectrum-based Fault Localization (SFL) [7] take advantage of this and reason with *all* these data. In this manuscript, we will thus discuss ways how to do so also in the context of MBD.

So, while we usually consider failed test cases in isolation (and then think about how to aggregate the information), our motivation is to implement a diagnosis algorithm that is able to consider all scenarios *simultaneously*. When we reasoned in [8] whether we could use a combinatorial test suite to trigger and diagnose all faults present in a system, we discussed a naive way to compute such global diagnoses. That is, due to the non-monotonic diagnosis context (see also Cor.2), we obviously can't just combine the diagnoses for the individual scenarios in order to arrive at global ones, which is a problem also for distributed diagnosis [9]. Rather we can compute them by collecting the *conflicts* for the individual scenarios, and then compute the global diagnoses as the minimal hitting sets (MHSs) of those. A disadvantage is that there is redundant work—like when we possibly compute one and the same conflict for every failed test case, or compute more conflicts than necessary (see Sec. 3).

In order to address this, we describe in this manuscript a conflict-based diagnosis algorithm where we consider multiple failing scenarios simultaneously. We extend RC-Tree [10] in order to derive our new MSRC-Tree algorithm, but the underlying concept can be easily integrated also into other conflict-driven algorithms like HS-DAG [11].

First experiments show the viability of our concept and they allowed us to identify first performance characteristics. As our experimental results suggest, there are situations where we can save a lot of expensive solver calls, resulting in a run-time advantage of up to two orders of magnitude. There are, however, also cases where the naive concept has its advantages due to the specific structure of the diagnosis problem and the chosen search parameters.

In our presentation, we will first discuss the basics of consistency-oriented model-based diagnosis in Sec. 2. After rehearsing the naive concept and proposing the details of our MSRC-Tree algorithm (and an optional alternative heurstistic) in Section 3, we will report on first experiments for ISCAS85 circuits in Sec. 4. Following a discussion of related work in Sec. 5, we conclude with final remarks.

2 Preliminaries: Of tests, conflicts, hitting sets and model-based diagnosis

As outlined in the introduction, we're aiming to reason with observations for a set of individual and independent scenarios for which a system failed. No matter whether the observed scenarios came from observing real-world behavior for sample scenarios, or during testing, in our presentation, we will refer to them as individual *tests or test cases* in order

to emphasize their independence. In this respect, please note that while we will use a combinational circuit for illustration purposes, with our algorithm we can indeed investigate also sequential behavior of systems with internal states like diagnosed in [12]. A system's input can thus certainly be also a sequence as supported by our definition of a test case.

Definition 1 (test case, test suite)**.** *A test case $t = (\delta, \rho)$ for some system S with inputs I and observable outputs O defines an input stimulus δ for S via defining desired values for all inputs $i \in I$. The system's reaction to δ is defined in ρ either via defining the expected values for all $o \in O$, or via some function implementing an automated test oracle [2]. A test suite T is a finite and non-empty set of test cases.*

As underlying diagnostic reasoning concept we rely on consistency-oriented model-based diagnosis as characterized by de Kleer and Williams in [4] and Reiter in [3] for single scenarios. There, a system description *SD* describes the behavior of the system in sentences of the form $h_i \rightarrow$ *nominal behavior of* c_i. Those sentences state that under the assumption that some component c_i works correctly (encoded in a health state variable $h_i \in H$) we know how the component shall behave.

We then have to provide such sentences for all components, and complement them with their connections and other system and background knowledge. Depending on the formalism, such background-knowledge includes, e.g., the mutual exclusiveness of values for some signal. Since we make no assumptions about how the components behave in case of a fault, the approach is considered to implement a *weak fault model*. While there exists indeed also a theory for considering fault models, it is out of this paper's scope. Given some observations *OBS* about the system's actual behavior, we can reason with *SD* whether *OBS* is consistent with the expected behavior described in *SD* under the assumption that all components work as expected (such that $SD \cup OBS \cup \{h_i | h_i \in H\}$ is consistent or satisfiable). If this is not the case, we can furthermore define and verify hypotheses $\Delta \subseteq H$ concerning faulty components [4; 3; 11; 10]—verifiable via checking $SD \cup OBS \cup \{h_i | h_i \in H \setminus \Delta\}$.

Definition 2. *A diagnosis for a diagnosis problem (SD, H, OBS) is a subset-minimal set $\Delta \subseteq H$ such that $SD \cup OBS \cup \{h_i | h_i \in H \setminus \Delta\}$ is consistent (satisfiable).*

We can compute the diagnoses for some diagnosis problem as minimal hitting sets of the set of conflicts, where it suffices to focus on the subset-minimal conflicts [3] (Cor. 1).

Definition 3. *A conflict C for a diagnosis problem (SD, H, OBS) is a subset of H such that $SD \cup OBS \cup \{h_i | h_i \in C\}$ is inconsistent (unsatisfiable). If no prober subset C' of C is a conflict, then C is a subset-minimal conflict.*

Definition 4. *A hitting set Δ for a set CS of conflicts as of Def. 3 is a subset of H such that $\Delta \cap C_i \neq \emptyset$ for all $C_i \in CS$. Δ is a minimal hitting set if and only if no proper subset of Δ is a hitting set as well.*

Corollary 1. *A diagnosis Δ for (SD, H, OBS) is a minimal hitting set for the set of all conflicts for this diagnosis problem. Since Δ will hit also all non-minimal conflicts if it hits the minimal ones, it suffices to focus on the minimal conflicts when computing the diagnoses.*

For combinational circuits, we can easily derive a CNF description for *SD* and the observations *OBS*. So for instance, we can use the following nine clauses to describe the simple circuit with two AND-gates and one OR-gate depicted in Figure 1. Please note that each gate has its corresponding health state variable h_i indicating whether it works correctly or not. If h_i is $\bot$ (False) such that the gate is faulty, the gate's clauses are satisfied by default. Otherwise the remainder of the individual clauses implement the desired function such that, e.g, an AND-gate's output is $\top$ (True) iff its two inputs are $\top$ as well.

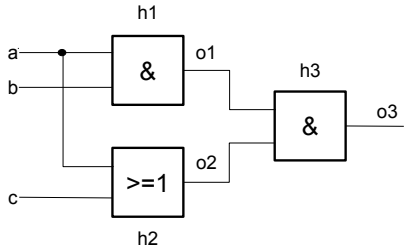

Figure 1: A simple combinational circuit.

- $\neg h_1 \vee \neg o_1 \vee a$
- $\neg h_1 \vee \neg o_1 \vee b$
- $\neg h_1 \vee \neg a \vee \neg b \vee o_1$
- $\neg h_2 \vee \neg o_2 \vee a \vee c$
- $\neg h_2 \vee o_2 \vee \neg a$
- $\neg h_2 \vee o_2 \vee \neg c$
- $\neg h_3 \vee \neg o_3 \vee o_1$
- $\neg h_3 \vee \neg o_3 \vee o_2$
- $\neg h_3 \vee \neg o_1 \vee \neg o_2 \vee o_3$

Using a CNF description of *SD* and unit clauses for the observations and health state assignment, we can exploit an off-the-shelf SAT solver for assessing whether some Δ makes the observations consistent with *SD* as of Def. 2. If this is not the case, we can ask the solver for a conflict set (a minimal unsatisfiable core in H). Please note that we will use examples from this domain for our evaluation in Sec. 4.

Some MBD algorithms like RC-Tree [10] allow us to compute the conflicts on-the-fly. This is of advantage especially if we are interested only in diagnoses up to a certain cardinality. That is, when we restrict our search by a common and intuitive assumption that we consider it unlikely that more than l components fail simultaneously. If we compute the conflicts on-the-fly, we might need to compute only a subset of all minimal conflicts, i.e., a subset that characterizes all Δ s.t. $|\Delta| \leq l$. Also in our algorithm, we support such a bound for offering scalable computations.

In contrast to sequential diagnosis where we probe for new data that allow us to discriminate between diagnoses, we are considering data from multiple independent scenarios. Due to this non-monotonic setting, we obviously can't simply compute global diagnoses (that resolve the issues in *all* scenarios) as combinations of those for the individual scenarios (see Cor. 2), and present our solution in Sec. 3.

3 Considering multiple failed test cases in a single diagnostic search

Our motivation is quite straightforward in that we would like to consider multiple failed test cases simultaneously in our diagnostic reasoning. In line with the ideas behind SFL, considering multiple failed test cases shall allow us to (a) consider more data in order to increase the preciseness of our reasoning, and (b) since not all faults will be triggered by a single test case, the reasoning then will draw on more representative data for isolating a system's faults.

It is obvious though that these two aspects are orthogonal: On one hand, we'd like to be more precise and compute less diagnoses, and on the other one, we try to trigger all the faults present in the system. To which degree some computation will achieve either goal will thus depend heavily on the test suite. Please note that while we have to take this into account when considering the experimental results, evaluating a test suite's quality is outside this paper's scope. Rather, we focus on *how to compute diagnoses for multiple failed test cases* and refer the interested reader to, e.g., [8; 13] for a discussion of a test suite's diagnostic quality.

Like mentioned in the introduction, we discussed recently a straightforward and *naive* approach of how we can derive global diagnoses for multiple failed scenarios. There, we solve in a first step the diagnosis problems for the individual failed test cases, and then compute the global diagnoses as the minimal hitting sets of *all* the *conflicts* collected from the individual scenarios (Algorithm 1 as taken from [8]).

Algorithm 1 A straightforward, naive way to compute global diagnoses for multiple scenarios

Requires: A test suite T, a system description *SD*, a cardinality limit l, as well as functions $diag(OBS, SD, l)$ and $MHS(CS, l)$. The former function computes the diagnoses and conflicts for a diagnosis problem $OBS, SD, l)$, and the latter the minimal hitting sets—up to cardinality l—for a set *CS* of conflicts.

```
1:  procedure COMBDIAG(T,SD,l)
2:      T' ← ∅
3:      CS' ← ∅
4:      for t ∈ T do                                  ▷ Stage 1
5:          (v, OBS) ← execute(t)
6:          if v = failed then
7:              T' ← T' ∪ {OBS}
8:          end if
9:      end for
10:     for OBS ∈ T' do                               ▷ Stage 2
11:         (DS, CS) ← diag(OBS, SD, l)
12:         CS' ← CS' ∪ CS
13:     end for
14:     RES ← SMHS(CS', l)                            ▷ Stage 3
15: end procedure
```

This algorithm serves as our baseline and is based on 3 stages. In Stage 1, we execute all test cases t and for each one we derive a tuple (v, OBS) with a verdict v (either *passed* or *failed*) and *OBS* as the set of observations obtained for the execution. Please note that *OBS* thus shall contain also the input stimulus δ. If $v == failed$, we add *OBS* to T'. After completing Stage 1, T' contains corresponding observation data OBS_i for each failed t (and only for those). In Stage 2, we diagnose all these failed test case executions individually using an algorithm like RC-Tree [10] or HS-DAG [11]. This algorithm shall not only return the set of diagnoses *DS*, but also the set *CS* containing all minimal conflicts derived during the computation. These conflicts are collected in *CS'*, so that upon completion of Stage 2, *CS'* contains all conflicts (and only those) needed to derive the individual diagnoses. In the final Stage 3, we compute the MHSs for *CS'* and return them via *RES*. We refer the interested reader to [8] for a more detailed discussion and corresponding proofs of soundness and exhaustiveness, but let us now rehearse a few essential observations.

Corollary 2. *(see Cor. 2 in [8]) For some diagnosis $\Delta \in DS$ for a cardinality-restricted diagnosis problem (OBS, SD, l) considered in Stage 2, RES might neither contain Δ nor some superset Δ' s.t. $\Delta \subseteq \Delta'$.*

This corollary illustrates that the global diagnoses in *RES* are not simply the possible combinations of the diagnoses for the individual problems (we are non-monotonic [3; 4]).

Corollary 3. *(see Thm. 1 in [8]) An MHS $\Delta \in RES$ defines an assignment to the health state variables such that for all $OBS \in T'$ we have that those observations are consistent with SD for this assignment. Formally, we have that $OBS \cup SD \cup \{h_i | h_i \in H \setminus \Delta\}$ is satisfiable for each $OBS \in T'$.*

Corollary 4. *(cf. Corollary 3 in [8]) For every MHS $\Delta \in RES$, and for every $OBS \in T'$ there is some $\Delta' \subseteq \Delta$ that is a diagnosis for (OBS, SD, l).*

Now, since Definition 2 of a diagnosis does not fit a multi-scenario setting, let us define the term *multi-scenario diagnosis* (in [8] we used the term multi-observation diagnosis which could be unintentionally associated also with sequential diagnosis) for describing the $\Delta \in RES$ based on Cor. 4.

Definition 5. *[multi-scenario diagnosis] (cf. Def. 5 in [8]) A multi-scenario diagnosis for some test suite T and a system description SD is a subset $\Delta \subset H$ such that:*

- *no proper subset of Δ is also a multi-observation diagnosis*
- *for every test case $t \in T$ whose execution failed, there is some $\Delta' \subseteq \Delta$ that is a diagnosis for that failed test case according to Definition 2.*

If we have a detailed look at the computation in Alg. 1, we can easily see that we might compute conflicts more than once—possibly for every test case. This stems from the fact that, in principle, we solve $|T'|$ individual diagnosis problems in Stage 2 (neglecting to exploit previously computed conflicts for solving the next diagnosis problem) and then "combine" the results via computing the MHSs in Stage 3.

In contrast, we propose with MSRC-Tree a concept where we extend RC-Tree such that we consider all scenarios simultaneously and avoid such redundant conflict computations. We compute the needed conflicts on-the-fly—switching our focus between the scenarios—and implement a *single exploration strategy* for conquering the global diagnosis search space. Consequently, we create a *single* tree for our search, rather than multiple (local) ones when using RC-Tree in Alg. 1. To this end, we consider all previously computed conflicts (no matter for which *OBS* they were computed) and keep track of which *OBS* have been previously addressed in a search branch. Please note that for *MSRC-Tree*, we assume that T' containing the OBS_i for the failed test cases (see Stage 1 of Alg. 1) is available.

Algorithm 2. *(MSRC-Tree) Let D be a growing node- and edge-labeled tree with some initial and unlabeled root node n_0. Process unlabeled nodes in D in breadth-first order as follows, where for some node n, $h(n)$ is defined to be the set of edge labels on the path in D from n_0 to n ($h(n_0) = \emptyset$). Furthermore assume the sets $\Theta(n)$ and $\Theta_C(n)$ to be subsets of $\bigcup_{C_i \in CS} C_i$, where $\Theta(n_0) = \Theta_C(n_0) = \emptyset$. Let $S(n)$ be a subset of integers in $\{0, \ldots, |T'|-1\}$ where the initial value for n_0 is $S(n_0) = \emptyset$, let $l > 0$ be an integer that defines the desired maximum diagnosis cardinality, and let $0 \leq o < |T|$ be an integer storing which $OBS_o \in T'$ was considered last*

(the initial value being $o = |T'| - 1$). Furthermore, let CS be a set containing all conflicts derived so far.

1. *(Closing) If there is a node n' s.t $h(n') \subset h(n)$ and n' is labeled with "✓" ($h(n')$ is a hitting set), then close n. Neither compute a label for n, nor generate any successor nodes for n, but proceed with the next node.*
2. *Iff for all $OBS_i \in T'$ we have that $i \in S(n)$ label n with "✓". Otherwise label n with some C_j that is the first set in CS s.t. $C_j \cap h(n) = \emptyset$. If there is no such C_j, then repeat the following steps a to c until we either find such a C_j as label for n, or until we have for all $OBS_i \in T'$ that $i \in S(n)$, so that we label n with "✓".*
   (a) *$o = (o+1)\%|T'|$*
   (b) *while $o \in S(n)$ let $o = (o+1)\%|T'|$*
   (c) *check for $OBS_o \in T'$: if $SD \cup OBS_o \cup h(n)$ is consistent, then $S(n) = S(n) \cup \{o\}$. Else compute a conflict $C_j \subseteq H$ for $SD \cup OBS_o \cup h(n)$ as label for node n*
3. *(Pruning) Iff a priorly unused set C_i was used to label node n, attempt to prune D. That is, for nodes n' labeled with some $C_j \in CS$ s.t. $C_i \subset C_j$ do as follows:*
   (a) *Relabel n' with C_i. Then, for any c_i in $C_j \setminus C_i$, the edge labeled c_i originating from n' is no longer allowed. The node connected by this edge and all of its descendants are removed.*
   Now, for all children n'' of n' update $\Theta_C(n'')$ to $\Theta_C(n'') \setminus (C_j \setminus C_i)$ and for all descendants n''' of some n'' propagate the update accordingly. Then create for all n'' and n''' all the edges that are not avoided anymore (due to the updates to their Θs), and process the new nodes in breadth-first order as usual (choosing a node with the smallest $h(n)$).
   (b) *Eliminate C_j from CS.*

   If the currently processed node n was removed, proceed with the next unlabeled node.
4. *If n was labeled with some $C_i \in CS$ and $h(n) < l$, generate for each $c_i \in C_i \setminus \Theta(n)$ a new edge e originating in n and labeled with c_i. Generate a new node n', where $\Theta(n') = \Theta_C(n') \cup \Theta(n)$ with $\Theta_C(n') = \{c_j | c_j \in C_i$ and we already created an edge labeled c_j from $n\}$ as destination for the edge e. Let the initial $S(n')$ be inherited from its parent n s.t. $S(n') = S(n)$. The new node n' will be processed (labeled and expanded) after all new nodes n_i in the same generation as n (s.t. $|h(n_i)| = |h(n)|$) have been processed.*
5. *If there is no further unlabeled node, return tree D.*

Please note that in our description, the on-the-fly computation of conflicts C_i, the conflict buffer *CS* and also the cardinality exploration limit l are explicitly outlined. Even if not mentioned in their definitions, usually these features are implemented also for algorithms like RC-Tree or HS-DAG.

MSRC-Tree is based on RC-Tree whose basic concept is to explore the diagnosis search space in a breadth-first way via resolving encountered conflicts. That is, for each identified conflict, RC-Tree generates (like HS-DAG) a new search branch for each h_i in C_j, and $h(n)$ as the collection of h_i along a branch gives us thus one h_i for each conflict label along the path from the root node n_0 to n. Via the breadth-first exploration and the closing as well as pruning steps, the algorithm ensures that no non-minimal hitting set $h(n)$ is labeled as diagnosis (with a ✓). In contrast to HS-DAG, RC-Tree does not create all outgoing edges (only those not in $\Theta(n)$) since RC-Tree creates every $h(n)$ at most once in order to avoid redundant computations.

In MSRC-Tree, we have to investigate not only one scenario's observations in Step 2 (where we decide whether $h(n)$ is a diagnosis or generate conflicts as labels on-the-fly), but those of *all* scenarios. In Step 2, we thus have to verify whether $h(n)$ provides a health state assignment that makes *all* OBS_i in T' consistent with *SD* as required by Def. 5. In the naive algorithm, this is achieved by first collecting the conflicts for all the OBS_i (via solving the individual diagnosis problems in Stage 2), but in MSRC-Tree we compute these conflicts on-the-fly. In particular, we (possibly repeatedly) select some $OBS_i \in T'$. That is, we first check whether there is already some conflict that we can use as label, and if not, we check whether $h(n)$ makes *SD* consistent with some OBS_i that was incompatible with the $h(n')$s in the path to n (i.e., subsets of $h(n)$) such that $i \notin S(n)$. We use $S(n)$ and the inheritance of the initial value in Step 4 along a branch to collect these data. From those OBS_i not in $S(n)$, we choose the next one after the previously used OBS_j (in a global context and not per branch), and store the new i in o. We thus rotate over the scenarios, where the idea is to possibly generate conflicts from multiple scenarios as early as possible. Now, only if $|S(n)| = |T'|$ (so that we can't compute a conflict for $h(n)$ for any of the scenarios), we label a node with a ✓ in order to indicate a diagnosis ($h(n)$ is an MHS of the conflicts of all scenarios). Also in MSRC-Tree, the breadth-first exploration and the closing as well as pruning steps ensure that we do not compute non-minimal diagnoses.

Please note that we could opt also for some other observation selection strategy in Step 2. In our experiments, we will thus report also on an MSRC-Tree variant where we select the observations in ascending order in each branch (coined *linear*, in contrast to the standard strategy *rotate*).

In RC-Tree (and also HS-DAG) the pruning step is not mandatory if the solver returns minimal conflicts (even for an on-the-fly computation). The same is true for Alg. 1, since there we can sort the conflicts for stage 3 according to their cardinality. In MSRC-Tree, the pruning step is important though, since even if a solver returns minimal conflicts, we have to be aware that this holds only for single scenarios. In others, the same conflict could be non-minimal (see [8] for an example). So, also if we adapted HS-DAG to mimic the exploration concept of MSRC-Tree in terms of selecting observations, the pruning step would be required.

While we can't provide full proofs for space reasons, MSRC-Tree is indeed sound and also exhaustive.

Theorem 1. *Algorithm 2 is sound and exhaustive within the bounds of cardinality limit l. That is, for every node n marked with ✓, $h(n)$ is a multi-scenario diagnosis as of Def. 5. Furthermore, if there is some multi-scenario diagnosis Δ with $|\Delta| \leq l$ for a system and a set of failed test cases T', the tree computed by Algorithm 2 will contain a node n such that $h(n) = \Delta$ and n is labeled with ✓.*

Proof. (brief sketch) Soundness can be argued via the breadth-first search and the closing rule that ensure subset-minimality on one hand, and on the other hand drawing on the fact that we mark a node with ✓ only if there is no observation such that $OBS \cup SD \cup \{h_i | h_i \in H \setminus h(n)\}$ is

inconsistent. Exhaustiveness is ensured by the exhaustive breadth-first search (see Step 4) in relation to Def. 5. □

Finally, we would like to note that while the naive algorithm needs to compute all conflicts for characterizing the individual diagnoses up to size l, MSRC-Tree computes only those necessary to describe the global ones up to size l.

4 Experiments

For our first experiments, we focused on well-known combinational circuits from the ISCAS 85 benchmarks [14]. These experiments were conducted on an Early 2015 Macbook Pro with a 2.9Ghz i5 Intel CPU, 8GB of RAM, and an SSD. MSRC-Tree in its standard variant and a variant implementing the linear strategy, as well as the naive algorithm were implemented in Python 3, where we used picomus 965 [15] as SAT solver. For creating faulty circuits, we injected 1 to 3 faults by replacing the functionality of a gate with another gate's, or implementing stuck-at-zero (SA0) or stuck-at-one (SA1) faults. Always ignoring the gate's original type, for unary gates like a buffer or inverter thus the possible faults were {BUFF, INV, SA0, SA1}, for gates of type {AND, OR, NAND, NOR, XOR, XNOR} with two inputs {AND, OR, NAND, NOR, XOR, XNOR, SA0, SA1}, and for gates of type {AND, OR, NAND, NOR} with more than two inputs the fault mode list was {AND, OR, NAND, NOR, SA0, SA1}.

For each fault cardinality we created 5 fault vectors with 20 failing test cases. Then, for each search limit l from 1 to 3, we computed diagnoses for the full test suite, two halves with 10 test cases, and four quarters with 5 test cases—amounting to 315 scenarios for each of the circuits c17, c432, and c499. The reported run-times are averages over 10 runs, so that we solved a total of 28.350 multi-scenario diagnosis problems for the results in Tables 1 and 2. In the first table, *scalls* refers to the number of satisfiable SAT solver calls, *ucalls* to the unsatisfiable ones (when we compute a conflict), and *tcalls* to the total number of solver calls. We report for each circuit the ratio between the average performance over 10 runs of MSRC-Tree against the naive algorithm and the MSRC-Tree variant implementing the linear strategy—on average over all scenarios and the best and worst ratios. Detailed run-time plots are available in Fig. 2.

In Table 2, we show for each of the three algorithms for how many of the 315 scenarios it offered the best run-time performance (average run-time over 10 runs). From that table we can see that the linear strategy can sometimes outperform the standard strategy of MSRC-Tree. Overall, MSRC-Tree outperformed the naive algorithm for 255 (81%)/305 (96.8%)/260 (82.5%) of the scenarios respectively, aside being faster on average as suggested by Table 1.

Table 1: Performance ratio of MSRC-Tree in comparison to the naive Alg. 1 and MSRC-tree with a linear strategy. The ratios were computed with the average values over 10 runs (e.g., avg time MSRC-Tree / avg. time naive).

		C17		C432		C499	
		naive	linear	naive	linear	naive	linear
time	best	0.0278	0.427	0.0106	0.159	0.00905	0.225
	avg.	0.688	0.981	0.274	0.84	0.899	0.989
	worst	1.74	1.66	3.35	1.76	11.6	2.55
scalls	best	0	0	0	0	0	0
	avg.	0.901	0.942	0.793	0.947	1.2	1.01
	worst	2.3	2	8.95	11	16.2	8
ucalls	best	0.0417	0.667	0.00733	0.125	0.037	0.333
	avg.	0.271	0.997	0.122	0.758	0.213	0.886
	worst	1	1.6	1	2.09	1	1.67
tcalls	best	0.0274	0.429	0.011	0.171	0.00587	0.3
	avg.	0.271	0.982	0.122	0.867	0.213	0.979
	worst	1.77	1.67	4.97	1.82	15.2	3

Table 2: Number and percentage of scenarios for which an algorithm offered the best average run-time over 10 runs.

algorithm	C17	C432	C499
MSRC-Tree	143 (45.4%)	215 (68.3%)	171 (54.3%)
linear var.	122 (38.7%)	92 (29.2%)	91 (28.9%)
naive	50 (15.9%)	8 (2.54%)	53 (16.8%)

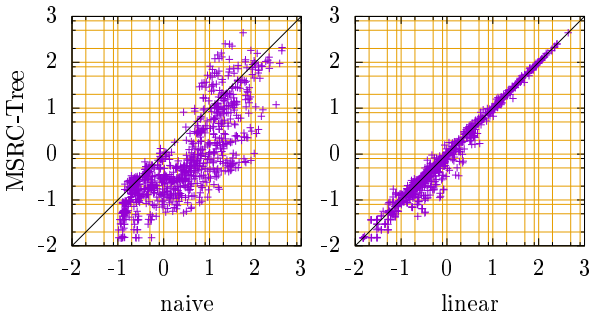

Figure 2: Scatter plots comparing MSRC-Tree's average run-time (in 10^y seconds) against those of the naive algorithm and the linear MSRC-Tree variant (C17/C432/C499).

5 Related work

From an abstract point of view, sequential diagnosis [16] where we focus on determining new "measurements" [5; 6] for being able to discriminate between diagnoses is related to our research. While also there the aim is to enrich the data used in the reasoning process, the focus is on determining *which data queries* would support such a discrimination. In our case, it is pre-defined which data we can use, and we do not construct distinguishing tests [17]. If one had the option to (dynamically) tune the generation of the test suite, sequential diagnosis research would certainly be of interest.

For such optimizations, we consider work on fault diagnosability, e.g, in the area of discrete event systems [18], to be related even closer. In relation to MSRC-Tree, such work again has a different focus, but might support us in generating and fine-tuning test suites.

Observations for multiple time steps have been used, e.g., in [19] without referring to independent scenarios or sequential behavior though. In [12] we used temporal observations for investigating temporal specifications, but did not consider multiple scenarios for a specification. In [20], the authors also consider the case of multiple observations and how to handle them. This paper is close to ours, however, there are also differences like the introduction of different algorithms relying on hitting-set computations.

6 Summary and conclusions

We proposed in this paper with RC-Tree a conflict-driven MBD algorithm that considers multiple failed scenarios in a single search. In our experiments for the ISCAS85 circuits C17,C432, and C499, MSRC-Tree needed on average only 0.674/0.307/0.562 the run-time of the naive solution and outperformed it for 81-96.8% of the scenarios. The best ratio over all circuits and scenarios was 0.0071, which means a speed-up of over two orders of magnitude. For some search configurations, MSRC-Tree was 10.1 times slower though—due to more scalls needed in the specific searches. An MSRC-Tree variant with an alternative strategy turned out to be an interesting solution as well, offering the best performance for about a third of the scenarios but being slower on average. In terms of the best algorithm for some scenario, we saw that the naive concept was the best performer for only 2.23% to 16.8% of the scenarios (depending on the circuit). In contrast, MSRC-Tree offered the best performance for 45.5% to 68.3% of the scenarios, and its linear variant for 29.2% to 38.7%.

As usual, our run-time evaluation takes single-core performance into account. Considering today's computers (even in consumer-grade hardware we sometimes have 8 cores that can deal with 16 threads), parallelization is an interesting topic though. So, while the evaluation still reports on efficiency and consumed resources then, for the naive concept we can easily parallelize the consideration of the individual scenarios in Stage 2. Experiencing an efficiency drawback of redundantly computing (possibly unnecessary) conflicts, we could still achieve shorter wall clock run-times via such a parallelization. Since MSRC-Tree's performance advantage reached more than two orders of magnitude, on consumer hardware it would outperform even a parallelized naive solution for some scenarios.

If multiple cores are available, a portfolio approach where we either run MSRC-Tree and its linear variant in parallel (or all three algorithms) could be an attractive solution. If we're targeting a single core scenario, our experimental results suggest that MSRC-Tree is the most attractive solution from an overall perspective.

In future research we will evaluate the performance for a variety of domains, including LTL specifications [12]. We will also delve into performance details—in order to identify performance indicators for the individual algorithms. Such indicators could allow us to suggest which algorithm to select for some scenario and could provide data for developing further promising strategies (vs. *rotate* and *linear*).

Acknowledgments

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

References

[1] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[2] I. Pill and F. Wotawa. Automated generation of (F)LTL oracles for testing and debugging. *Journal of Systems and Software*, 139(C):124–141, May 2018.

[3] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[4] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[5] A. Hou. A theory of measurement in diagnosis from first principles. *Art. Intell.*, 65(2):281–328, 1994.

[6] B. Han, S. Lee, and H. Yang. Comments on the theory of measurement in diagnosis from first principles. *Information Sciences*, 121(3):349 – 365, 1999.

[7] D. R. Kuhn, R. N. Kacker, and Y. Lei. Practical combinatorial testing. Technical report, 2010. `http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-142.pdf`.

[8] I. Pill and F. Wotawa. Exploiting observations from combinatorial testing for diagnostic reasoning. In *International Workshop on Principles of Diagnosis (DX)*, 2019. `https://dx-workshop.org/2019/wp-content/uploads/2019/papers/DX_2019_paper_6.pdf`.

[9] F. Wotawa and I. Pill. On classification and modeling issues in distributed model-based diagnosis. *AI Communications*, 26(1):133–143, 2013.

[10] I. Pill and T. Quaritsch. RC-Tree: A variant avoiding all the redundancy in Reiter's minimal hitting set algorithm. In *IEEE Int. Symp. on Software Reliability Engineering Workshops (ISSREW)*, pages 78–84, 2015.

[11] R. Greiner, B. A. Smith, and R. W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *Artif. Intelligence*, 41(1):79–88, 1989.

[12] I. Pill and T. Quaritsch. Behavioral diagnosis of LTL specifications at operator level. In *Int. Joint Conf. on Artificial Intelligence*, pages 1053–1059, 2013.

[13] A. Perez, R. Abreu, and A. van Deursen. A test-suite diagnosability metric for spectrum-based fault localization approaches. In *IEEE/ACM Int. Conf. on Software Engineering (ICSE)*, pages 654–664, 2017.

[14] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN. In *IEEE Int. Symp. on Circuits and Systems*, 1985.

[15] A. Biere. Picosat essentials. *J. on Satisfiability, Boolean Modeling and Comp.*, 4(2-4):75–97, 2008.

[16] P. Rodler. On active learning strategies for sequential diagnosis. In *28th International Workshop on Principles of Diagnosis (DX'17)*, volume 4 of *Kalpa Publications in Computing*, pages 264–283, 2018.

[17] R. Alur, C. Courcoubetis, and M. Yannakakis. Distinguishing tests for nondeterministic and probabilistic machines. In *27th Annual ACM Symposium on Theory of Computing*, STOC '95, pages 363–372, 1995.

[18] X. Su, M. Zanella, and A. Grastien. Diagnosability of discrete-event systems with uncertain observations. In *Int. J. Conf. on Art. Intell.*, pages 1265–1271, 2016.

[19] O. Raiman, J. de Kleer, V. A. Saraswat, and M. Shirley. Characterizing non-intermittent faults. In *National Conf. on Art. Intelligence, Vol.2*, pages 849–854, 1991.

[20] A. Ignatiev, A. Morgado, G. Weissenbacher, and J. Marques-Silva. Model-based diagnosis with multiple observations. In *Int. J. Conf. on Artificial Intelligence*, pages 1108–1115, 2019.