# Reuse, Reduce and Recycle: Adapting Reiter's HS-Tree to Sequential Diagnosis

**Patrick Rodler**
Alpen-Adria Universität Klagenfurt, 9020 Klagenfurt, Austria
patrick.rodler@aau.at

## Abstract

Reiter's HS-Tree is one of the most popular diagnostic search algorithms due to its desirable properties and general applicability. In sequential diagnosis, where the addressed diagnosis problem is subject to successive change through the acquisition of additional knowledge about the diagnosed system, HS-Tree is used in a stateless fashion. That is, the existing search tree is discarded when new knowledge is obtained, albeit often large parts of the tree are still relevant and have to be rebuilt in the next iteration, involving redundant operations and costly reasoner calls. As a remedy to this, we propose DynamicHS, a variant of HS-Tree that avoids these redundancy issues by maintaining state throughout sequential diagnosis while preserving all desirable properties of HS-Tree. Preliminary results of ongoing evaluations in a problem domain where HS-Tree is the state-of-the-art diagnostic method suggest significant time savings achieved by DynamicHS by reducing expensive reasoner calls.

## 1 Introduction

In model-based diagnosis, given a *diagnosis problem instance (DPI)*—consisting of the *system description*, the system *components*, and the *observations*—a *(minimal) diagnosis* is an (irreducible) set of components that, when assumed abnormal, leads to consistency between system description (predicted behavior) and observations (real behavior). In many cases, there is a substantial number of competing (minimal) diagnoses. To isolate the *actual diagnosis* (which pinpoints the *actually* faulty components), *sequential diagnosis* [2] methods collect additional system observations (called *measurements*) to gradually refine the set of diagnoses.

One of the most popular and widely used algorithm for the computation of diagnoses in model-based diagnosis is Reiter's HS-Tree [15]. It is adopted in various domains such as for the debugging of software [1; 25] or ontologies and knowledge bases [5; 16; 11; 13], or for the diagnosis of hardware [6], recommender systems [4], configuration systems [3], and circuits [15]. The reasons for its widespread adoption are that *(i) it is broadly applicable*, because all it needs is a system description in some monotonic knowledge representation language for which a sound and complete inference method exists, *(ii) it is sound and complete*, as it computes *only* minimal diagnoses and can, in principle, output *all* minimal diagnoses, and *(iii) it computes diagnoses in best-first order* according to a given preference criterion.

However, HS-Tree per se does not encompass any specific provisions for being used in an iterative way. In other words, the DPI to be solved is assumed constant throughout the execution of HS-Tree. As a consequence of that, the question we address in this work is whether HS-Tree can be optimized for adoption *in a sequential diagnosis scenario*, where the DPI to be solved is subject to successive change (information acquisition through measurements). Already Raymond Reiter, in his seminal paper [15] from 1987, asked:

> *When new diagnoses do arise as a result of system measurements, can we determine these new diagnoses in a reasonable way from the (...) HS-Tree already computed in determining the old diagnoses?*

To the best of our knowledge, no algorithm has yet been proposed that sheds light on this very question.

As a result, state-of-the-art sequential approaches which draw on HS-Tree for diagnosis computation handle the varying DPI to be solved by re-invoking HS-Tree each time a new piece of system knowledge (measurement outcome) is obtained. This amounts to a *discard-and-rebuild* usage of HS-Tree, where the search tree produced in one iteration is dropped prior to the next iteration, where a new one is built from scratch. As the new tree obtained after incorporating the information about one measurement outcome usually quite closely resembles the existing tree, this approach generally requires substantial redundant computations, which often involve a significant number of expensive reasoner calls. For instance, when debugging knowledge bases written in highly expressive logics such as OWL 2 DL [7], a single call to an inference service is already 2NEXPTIME-complete.

Motivated by that, we propose DynamicHS, a *stateful* variant of HS-Tree that pursues a *reuse-and-adapt* strategy and is able to manage the dynamicity of the DPI throughout sequential diagnosis while avoiding the mentioned redundancy issues. The idea is to maintain *one* (successively refined) tree data structure and to exploit the information it contains to enhance computations in the subsequent iteration(s), e.g., by circumventing costly reasoner invocations. The main objective of DynamicHS is to allow for a better (time-)efficiency than HS-Tree while maintaining all aforementioned advantages (generality, soundness, completeness, best-first property) of the latter. Preliminary results of ongoing evaluations on several KB debugging problems—a domain where HS-Tree is the prevalent means for diagnosis computation [5; 23; 11; 13; 9]—suggest a superiority of DynamicHS against HS-Tree.

## 2 Preliminaries

We briefly characterize technical concepts used throughout this work, based on the framework of [23; 16] which is (slightly) more general [19] than Reiter's theory of diagnosis [15].

**Diagnosis Problem Instance (DPI).** We assume that the diagnosed system, consisting of a set of components $\{c_1, \dots, c_k\}$, is described by a finite set of logical sentences $\mathcal{K} \cup \mathcal{B}$, where $\mathcal{K}$ (possibly faulty sentences) includes knowledge about the behavior of the system components, and $\mathcal{B}$ (correct background knowledge) comprises any additional available system knowledge and system observations. More precisely, there is a one-to-one relationship between sentences $ax_i \in \mathcal{K}$ and components $c_i$, where $ax_i$ describes the nominal behavior of $c_i$ (*weak fault model*). E.g., if $c_i$ is an AND-gate in a circuit, then $ax_i := out(c_i) = and(in1(c_i), in2(c_i))$; $\mathcal{B}$ in this example might encompass sentences stating, e.g., which components are connected by wires, or observed outputs of the circuit. The inclusion of a sentence $ax_i$ in $\mathcal{K}$ corresponds to the (implicit) assumption that $c_i$ is healthy. Evidence about the system behavior is captured by sets of positive ($P$) and negative ($N$) measurements [15; 2; 3]. Each measurement is a logical sentence; positive ones $p \in P$ must be true and negative ones $n \in N$ must not be true. The former can be, depending on the context, e.g., observations about the system, probes or required system properties. The latter model, e.g., properties that must not hold for the system. We call $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ a *diagnosis problem instance (DPI)*.

**Diagnoses.** Given that the system description along with the positive measurements (under the assumption $\mathcal{K}$ that all components are healthy) is inconsistent, i.e., $\mathcal{K} \cup \mathcal{B} \cup P \models \bot$, or some negative measurement is entailed, i.e., $\mathcal{K} \cup \mathcal{B} \cup P \models n$ for some $n \in N$, some assumption(s) about the healthiness of components, i.e., some sentences in $\mathcal{K}$, must be retracted. We call such a set of sentences $\mathcal{D} \subseteq \mathcal{K}$ a *diagnosis* for the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ iff $(\mathcal{K} \setminus \mathcal{D}) \cup \mathcal{B} \cup P \not\models x$ for all $x \in N \cup \{\bot\}$. We say that $\mathcal{D}$ is a *minimal diagnosis* for $dpi$ iff there is no diagnosis $\mathcal{D}' \subset \mathcal{D}$ for $dpi$. The set of minimal diagnoses is representative for all diagnoses (under the weak fault model [12]), i.e., the set of all diagnoses is exactly given by the set of all supersets of all minimal diagnoses. Therefore, diagnosis approaches usually restrict their focus to only minimal diagnoses. In the following, we denote the set of all minimal diagnoses for a DPI $dpi$ by $\mathbf{diag}(dpi)$. We furthermore denote by $\mathcal{D}^*$ the *actual diagnosis* which pinpoints the actually faulty axioms, i.e., all elements of $\mathcal{D}^*$ are in fact faulty and all elements of $\mathcal{K} \setminus \mathcal{D}^*$ are in fact correct.

**Conflicts.** Instrumental for the computation of diagnoses is the concept of a conflict [2; 15]. A conflict is a set of healthiness assumptions for components $c_i$ that cannot all hold given the current knowledge about the system. More formally, $\mathcal{C} \subseteq \mathcal{K}$ is a *conflict* for the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ iff $\mathcal{C} \cup \mathcal{B} \cup P \models x$ for some $x \in N \cup \{\bot\}$. We call $\mathcal{C}$ a *minimal conflict* for $dpi$ iff there is no conflict $\mathcal{C}' \subset \mathcal{C}$ for $dpi$. In the following, we denote the set of all minimal conflicts for a DPI $dpi$ by $\mathbf{conf}(dpi)$. A (minimal) diagnosis for $dpi$ is then a (minimal) hitting set of all minimal conflicts for $dpi$ [15], where $X$ is a *hitting set* of a collection of sets $\mathbf{S}$ iff $X \subseteq \bigcup_{S_i \in \mathbf{S}} S_i$ and $X \cap S_i \neq \emptyset$ for all $S_i \in S$.

**Algorithm 1** Sequential Diagnosis

**Input:** DPI $dpi_0 := \langle \mathcal{K}, \mathcal{B}, P, N \rangle$, probability measure $pr$, number $ld$ ($\geq 2$) of minimal diagnoses to be computed per iteration, heuristic heur for measurement selection, boolean $dynamic$ (use DynamicHS if true, HS-Tree otherwise)

**Output:** $\{\mathcal{D}\}$ where $\mathcal{D}$ is the only remaining diagnosis for the extended DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle$

$P' \leftarrow \emptyset, N' \leftarrow \emptyset$ ▷ performed measurements
$\mathbf{D}_\checkmark \leftarrow \emptyset, \mathbf{D}_\times \leftarrow \emptyset, \mathsf{state} \leftarrow \langle [\,[\,]\,], [\,], \emptyset, \emptyset \rangle$ ▷ initial state of DynamicHS
**while** $true$ **do**
  **if** $dynamic$ **then**
    $\langle \mathbf{D}, \mathsf{state} \rangle \leftarrow$ DYNAMICHS$(dpi_0, P', N', pr, ld, \mathbf{D}_\checkmark, \mathbf{D}_\times, \mathsf{state})$
  **else**
    $\mathbf{D} \leftarrow$ HS-TREE$(dpi_0, P', N', pr, ld)$
  **if** $|\mathbf{D}| = 1$ **then return** $\mathbf{D}$
  $mp \leftarrow$ COMPUTEBESTMEASPOINT$(\mathbf{D}, dpi_0, P', N', pr, \mathsf{heur})$
  $outcome \leftarrow$ PERFORMMEAS$(mp)$ ▷ oracle inquiry (user interaction)
  $\langle P', N' \rangle \leftarrow$ ADDMEAS$(mp, outcome, P', N')$
  **if** $dynamic$ **then**
    $\langle \mathbf{D}_\checkmark, \mathbf{D}_\times \rangle \leftarrow$ ASSIGNDIAGSOKNOK$(\mathbf{D}, dpi_0, P', N')$

**Sequential Diagnosis.** Given multiple minimal diagnoses for a DPI, a sequential diagnosis process can be initiated. It involves a recurring execution of four phases, *(i)* computation of a set of leading (e.g., most probable) minimal diagnoses, *(ii)* selection of the best measurement based on these, *(iii)* conduction of measurement actions, and *(iv)* exploitation of the measurement outcome to refine the system knowledge. The goal in sequential diagnosis is to achieve sufficient diagnostic certainty (e.g., a single or highly probable remaining diagnosis) with highest efficiency. At this, the overall efficiency is determined by the costs for *measurement conduction* and for *computations of the diagnosis engine*. Whereas the former—which is not the topic of this work—can be ruled by proposing appropriate (low-cost and informative) measurements [2; 14; 23; 22; 21; 20], the latter is composed of the time required for *diagnosis computation*, for *measurement selection*, as well as for the *system knowledge update*. We address the efficiency optimization problem in sequential diagnosis by suggesting new methods (DynamicHS algorithm) for the diagnosis computation and knowledge update processes.

## 3 DynamicHS Algorithm

**Inputs and Outputs.** DynamicHS (Alg. 2) accepts the following arguments: (1) an initial DPI $dpi_0 = \langle \mathcal{K}, \mathcal{B}, P, N \rangle$, (2) the already accumulated positive and negative measurements $P'$ and $N'$, (3) a probability measure $pr$ (allowing to compute the probability of diagnoses), (4) a stipulated number $ld \geq 2$ of diagnoses to be returned, (5) the set of those diagnoses returned by the previous DynamicHS run that are consistent ($\mathbf{D}_\checkmark$) and those that are inconsistent ($\mathbf{D}_\times$) with the latest added measurement, and (6) a tuple of variables $\mathsf{state}$

(cf. Alg. 2), which altogether describe DynamicHS's current state. It outputs the $ld$ (if existent) minimal diagnoses of maximal probability wrt. $pr$ for the DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle$.

**Embedding in Sequential Diagnosis Process.** Alg. 1 sketches a generic sequential diagnosis algorithm and shows how it accommodates DynamicHS (line 5) or, alternatively, Reiter's HS-Tree (line 7), as methods for iterative diagnosis computation. The algorithm reiterates a while-loop (line 3) until the solution space of minimal diagnoses includes only a single element.[1] This is the case iff a diagnoses set $\mathbf{D}$ with $|\mathbf{D}| = 1$ is output (line 8) since both DynamicHS and HS-Tree are complete and always attempt to compute at least two diagnoses ($ld \geq 2$). On the other hand, as long as $|\mathbf{D}| > 1$, the algorithm seeks to acquire additional information to rule out further elements in $\mathbf{D}$. To this end, the best next measurement point $mp$ is computed (line 9), using the current system information—$dpi_0$, $\mathbf{D}$, and acquired measurements $P'$, $N'$—as well as the given probabilistic information $pr$ and some measurement selection heuristic heur (which defines what "best" means, cf. [17]). The conduction of the measurement at $mp$ (line 10) is usually accomplished by a qualified user (*oracle*) that interacts with the sequential diagnosis system, e.g., an electrical engineer for a defective circuit, or a domain expert in case of a faulty ontology. The measurement point $mp$ along with its result $outcome$ are used to formulate a logical sentence $m$ that is either added to $P'$ if $m$ constitutes a positive measurement, and to $N'$ otherwise (line 11). Finally, if DynamicHS is adopted, the set of diagnoses $\mathbf{D}$ is partitioned into those consistent ($\mathbf{D}_{\checkmark}$) and those inconsistent ($\mathbf{D}_{\times}$) with the newly added measurement $m$ (line 13).

**Reiter's HS-Tree.** DynamicHS inherits many of its aspects from Reiter's HS-Tree. Hence, we first recapitulate HS-Tree and then focus on the differences to and idiosyncrasies of DynamicHS.

HS-Tree computes minimal diagnoses for $dpi = \langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle$ in a sound, complete[2] and best-first way. Starting from a priority queue of unlabeled nodes $\mathbf{Q}$, initially comprising only an unlabeled root node, the algorithm continues to remove and label the first ranked node from $\mathbf{Q}$ (GETANDDELETEFIRST) until all nodes are labeled ($\mathbf{Q} = [\,]$) or $ld$ minimal diagnoses have been computed. The possible node labels are minimal conflicts (for internal tree nodes) and $valid$ as well as $closed$ (for leaf nodes). All minimal conflicts that have already been computed and used as node labels are stored in the (initially empty) set $\mathbf{C}_{calc}$. Each edge in the constructed tree has a label. For ease of notation, each tree node nd is conceived of as the set of edge labels along the branch from the root node to itself. E.g., the node at location ⑫ in iteration 1 of Fig. 2 is referred to as $\{1, 2, 5\}$. Once the tree has been completed ($\mathbf{Q} = [\,]$), i.e., there are no more unlabeled nodes, it holds that $\mathbf{diag}(dpi) = \{\mathsf{nd} \mid \mathsf{nd} \text{ is labeled by } valid\}$.

To label a node nd, the algorithm calls a labeling function (LABEL) which executes the following tests in the given order and returns as soon as a label for nd has been determined:

(L1) *(non-minimality):* Check if nd is non-minimal (i.e. whether there is a node n with label $valid$ where $\mathsf{nd} \supseteq \mathsf{n}$). If so, nd is labeled by $closed$.

(L2) *(duplicate):* Check if nd is duplicate (i.e. whether $\mathsf{nd} = \mathsf{n}$ for some other n in $\mathbf{Q}$). If so, nd is labeled by $closed$.

(L3) *(reuse label):* Scan $\mathbf{C}_{calc}$ for some $\mathcal{C}$ such that $\mathsf{nd} \cap \mathcal{C} = \emptyset$. If so, nd is labeled by $\mathcal{C}$.

(L4) *(compute label):* Invoke GETMINCONFLICT, a *(sound and complete) minimal conflict searcher*, e.g., QuickXPlain [10], to get a minimal conflict for $\langle \mathcal{K} \setminus \mathsf{nd}, \mathcal{B}, P \cup P', N \cup N' \rangle$. If a minimal conflict $\mathcal{C}$ is output, nd is labeled by $\mathcal{C}$. Otherwise, if 'no conflict' is returned, then nd is labeled by $valid$.

All nodes labeled by $closed$ or $valid$ have no successors and are leaf nodes. For each node nd labeled by a minimal conflict $L$, one outgoing edge is constructed for each element $e \in L$, where this edge is labeled by $e$ and pointing to a newly created unlabeled node $\mathsf{nd} \cup \{e\}$. Each new node is added to $\mathbf{Q}$ such that $\mathbf{Q}$'s sorting is preserved (INSERTSORTED). $\mathbf{Q}$ might be, e.g., *(i)* a FIFO queue, entailing that HS-Tree computes diagnoses in minimum-cardinality-first order (*breadth-first search*), or *(ii)* sorted in descending order by $pr$, where most probable diagnoses are generated first (*uniform-cost search*; for details see [16, Sec. 4.6]).

Finally, note the *statelessness* of Reiter's HS-Tree, reflected by $\mathbf{Q}$ initially including *only an unlabeled root node*, and $\mathbf{C}_{calc}$ being initially *empty*. That is, a HS-Tree is built from scratch in each iteration, every time for different measurement sets $P', N'$.

**Dynamicity of DPI in Sequential Diagnosis.** In the course of sequential diagnosis (Alg. 1), where additional system knowledge is gathered in terms of measurements, the DPI is subject to gradual change—it is dynamic. At this, each addition of a new (informative) measurement to the DPI also effectuates a transition of the sets of (minimal) diagnoses and (minimal) conflicts. Whereas this fact is of no concern to a stateless diagnosis computation strategy, it has to be carefully taken into account when engineering a stateful approach.

**Towards Stateful Hitting Set Computation.** To understand the necessary design decisions to devise a sound and complete stateful hitting set search, we look at more specifics of the conflicts and diagnoses evolution in sequential diagnosis:[3]

**Property 1.** *Let* $dpi_j = \langle \mathcal{K}, \mathcal{B}, P, N \rangle$ *be a DPI and let* $T$ *be Reiter's HS-Tree for* $dpi_j$ *(executed until) producing the diagnoses* $\mathbf{D}$ *where* $|\mathbf{D}| \geq 2$*. Further, let* $dpi_{j+1}$ *be the DPI resulting from* $dpi_j$ *through the addition of an informative*[4] *measurement* $m$ *to either* $P$ *or* $N$*. Then:*

1. *$T$ is not a correct HS-Tree for $dpi_{j+1}$, i.e., (at least) some node labeled by valid in $T$ is incorrectly labeled.* (That is, to reuse $T$ for $dpi_{j+1}$, $T$ must be updated.)
2. *Each $\mathcal{D} \in \mathbf{diag}(dpi_{j+1})$ is either equal to or a superset of some $\mathcal{D}' \in \mathbf{diag}(dpi_j)$.*

[1]Of course, less rigorous stopping criteria are possible, e.g., when a diagnosis exceeds a predefined probability threshold [2].

[2]Unlike Reiter, we assume that only *minimal* conflicts are used as node labels. Thus, the issue pointed out by [8] does not arise.

[3]See [16, Sec. 12.4] for a more formal treatment and proofs.

[4]That is, adding $m$ to the (positive or negative) measurements of the DPI effectuates an invalidation of some diagnosis in $\mathbf{D}$.

(That is, minimal diagnoses can grow or remain unchanged, but cannot shrink. Consequently, to reuse $T$ for sound and complete minimal diagnosis computation for $dpi_{j+1}$, existing nodes must never be reduced—either a node is left as is, deleted as a whole, or (prepared to be) extended.)

3. *For all* $\mathcal{C} \in \mathbf{conf}(dpi_j)$ *there is a* $\mathcal{C}' \in \mathbf{conf}(dpi_{j+1})$ *such that* $\mathcal{C}' \subseteq \mathcal{C}$.
(That is, existing minimal conflicts can only shrink or remain unaffected, but cannot grow. Hence, priorly computed minimal conflicts (for an old DPI) might not be minimal for the current DPI. In other words, conflict node labels of $T$ can, but do not need to be, correct for $dpi_{j+1}$.)
4. *(a) There is some* $\mathcal{C} \in \mathbf{conf}(dpi_j)$ *for which there is a* $\mathcal{C}' \in \mathbf{conf}(dpi_{j+1})$ *with* $\mathcal{C}' \subset \mathcal{C}$*, and/or*
*(b) there is some* $\mathcal{C}_{new} \in \mathbf{conf}(dpi_{j+1})$ *where* $\mathcal{C}_{new} \not\subseteq \mathcal{C}$ *and* $\mathcal{C}_{new} \not\supseteq \mathcal{C}$ *for all* $\mathcal{C} \in \mathbf{conf}(dpi_j)$.
(That is, (a) some minimal conflict is reduced in size, and/or (b) some entirely new minimal conflict (not in any subset-relationship with existing ones) arises. Some existing node in $T$ which represents a minimal diagnosis for $dpi_j$ (a) can be deleted since it would not be present when using $\mathcal{C}'$ as node label in $T$ wherever $\mathcal{C}$ is used, or (b) must be extended to constitute a diagnosis for $dpi_{j+1}$, since it does not hit $\mathcal{C}_{new}$.)

**Major Modifications to Reiter's HS-Tree.** Based on Property 1, the following principal amendments to Reiter's HS-Tree are necessary to make it a properly-working stateful diagnosis computation method:

**(Mod1)** Non-minimal diagnoses (test (L1) in HS-Tree) and duplicate nodes (test (L2)) are stored in collections $\mathbf{D}_{\supset}$ and $\mathbf{Q}_{dup}$, respectively, instead of being closed and discarded.

*Justification:* Property 1.2 suggests to store also non-minimal diagnoses, as they might constitute (sub-branches of) minimal diagnoses in next iteration. Further, Property 1.4(a) suggests to record all duplicates for completeness of the diagnosis search. Because, some active node nd representing this duplicate in the current tree could become obsolete due to the shrinkage of some conflict, and the duplicate might be non-obsolete and eligible to turn active and replace nd in the tree.

**(Mod2)** Each node nd is no longer identified with the *set* of edge labels along its branch, but as an *ordered list* of these edge labels. In addition, an ordered list of the conflicts used to label internal nodes along this branch is stored in terms of nd.cs. E.g., for node nd at location ⑨ in iteration 1 of Fig. 1, nd $= [2, 5]$ and nd.cs $= [\langle 1, 2\rangle, \langle 1, 3, 5\rangle]$.

*Justification:* (Property 1.4) The reason for storing both the edge labels and the internal node labels as lists lies in the replacement of obsolete tree branches by stored duplicates. In fact, any duplicate used to replace a node must correspond to the same *set* of edge labels as the replaced node. However, in the branch of the obsolete node, some node-labeling conflict has been reduced to make the node redundant, whereas for a suitable duplicate replacing the node, no redundancy-causing changes to conflicts along its branch have occurred. By storing only sets of edge labels, we could not differentiate between the redundant and the non-redundant nodes.

**(Mod3)** Before reusing a conflict $\mathcal{C}$ (labeling test (L3) in HS-Tree), a minimality check for $\mathcal{C}$ is performed. If this leads to the identification of a conflict $X \subset \mathcal{C}$ for the current DPI, $X$ is used to prune obsolete tree branches, to replace node-labeling conflicts that are supersets of $X$, and to update $\mathbf{C}_{calc}$ in that $X$ is added and all of its supersets are deleted.

*Justification:* (Property 1.3) Conflicts in $\mathbf{C}_{calc}$ and those appearing as labels in the existing tree (elements of lists nd.cs for different nodes nd) might not be minimal for the current DPI (as they might have been computed for a prior DPI). This minimality check helps both to prune the tree (reduction of number of nodes) and to make sure that extensions to the tree use only minimal conflicts wrt. the current DPI as node labels (avoidance of the introduction of unnecessary new edges).

**(Mod4)** Execution of a tree update at start of each DynamicHS execution, where the tree produced for a previous DPI is adapted to a tree that allows to compute minimal diagnoses for the current DPI in a sound, complete and best-first way.

*Justification:* Property 1.1.

**(Mod5)** State of DynamicHS (in terms of the so-far produced tree) is stored over all its iterations executed throughout sequential diagnosis (Alg. 1) by means of the tuple state.

*Justification:* Statefulness of DynamicHS.

**DynamicHS: Algorithm Walkthrough.** Like HS-Tree, DynamicHS (Alg. 2) is processing a priority queue $\mathbf{Q}$ of nodes (while-loop). In each iteration, the top-ranked node node is removed from $\mathbf{Q}$ to be labeled (GETANDDELETEFIRST). Before calling the labeling function (DLABEL), however, the algorithm checks if node is among the known minimal diagnoses $\mathbf{D}_{\checkmark}$ from the previous iteration (line 6). If so, the node is directly labeled by $valid$ (line 7). Otherwise DLABEL is invoked to compute a label for node (line 9).

DLABEL: First, the non-minimality check is performed (lines 24–26), just as in (L1) in HS-Tree. If negative, a conflict-reuse check is carried out (lines 27–34). Note, the duplicate check ((L2) in HS-Tree) is obsolete since no duplicate nodes can ever be elements of $\mathbf{Q}$ in DynamicHS (duplicates are identified and added to $\mathbf{Q}_{dup}$ at node generation time, lines 18–19). The conflict-reuse check starts equally as in HS-Tree. However, if a conflict $\mathcal{C}$ for reuse is found in $\mathbf{C}_{calc}$ (line 28), then the minimality of $\mathcal{C}$ wrt. the *current* DPI is checked using FINDMINCONFLICT (line 29). If a conflict $X \subset \mathcal{C}$ is detected (line 32), then $X$ is used to prune the current hitting set tree (line 33; PRUNE function, see below). Finally, the found minimal conflict ($\mathcal{C}$ or $X$, depending on minimality check) is used to label node (lines 31, 34). The case where there is no conflict for reuse is handled just as in HS-Tree (lines 35–40, cf. (L4)). Finally, note that DLABEL gets and returns the tuple state (current tree state) as an argument, since the potentially performed pruning actions (line 33) might modify state.

The output of DLABEL is then processed by DynamicHS (lines 10–22) Specifically, node is assigned to $\mathbf{D}_{calc}$ if the returned label is $valid$ (line 11), and to $\mathbf{D}_{\supset}$ if the label is $nonmin$ (line 13). If the label is a minimal conflict $L$, then a child node $\mathsf{node}_e$ is created for each element $e \in L$ and assigned to either $\mathbf{Q}_{dup}$ (line 19) if there is a node in $\mathbf{Q}$ that is *set*-equal to $\mathsf{node}_e$, or to $\mathbf{Q}$ otherwise (line 21). At this, $\mathsf{node}_e$ is constructed from node via the APPEND function (lines 16 and 17), which appends the element $e$ to the list node, and the conflict $L$ to the list node.cs (cf. **(Mod2)** above).

When the hitting set tree has been completed ($\mathbf{Q} = [\,]$), or $ld$ diagnoses have been found ($|\mathbf{D}_{calc}| = ld$), DynamicHS returns $\mathbf{D}_{calc}$ along with the current tree state state.

UPDATETREE: The goal is to adapt the existing tree in a way it constitutes a basis for finding all and only minimal diagnoses in highest-probability-first order for the *current* DPI. The strategy is to search for non-minimal conflicts to be updated, and tree branches to be pruned, among the minimal diagnoses for the previous DPI that have been invalidated by the latest added measurement (the elements of $\mathbf{D}_{\times}$).

Regarding the pruning of tree branches, we call a node nd *redundant* (wrt. a DPI $dpi$) iff there is some index $j$ and a minimal conflict $X$ wrt. $dpi$ such that the conflict $\mathsf{nd.cs}[j] \supset X$ and the element $\mathsf{nd}[j] \in \mathsf{nd.cs}[j] \setminus X$. Moreover, we call $X$ a *witness of redundancy for* nd (wrt. $dpi$). Simply put, nd is redundant iff it would not exist given that the (current) minimal conflict $X$ had been used as a label instead of the (old, formerly minimal, but by now) non-minimal conflict $\mathsf{nd.cs}[j]$.

If a redundant node is detected among the elements of $\mathbf{D}_{\times}$ (function REDUNDANT), then the PRUNE function (see below) is called given the witness of redundancy of the redundant node as an argument (lines 42–44). After each node in $\mathbf{D}_{\times}$ has been processed, the remaining nodes in $\mathbf{D}_{\times}$ (those that are non-redundant and thus have not been pruned) are re-added to $\mathbf{Q}$ in prioritized order (INSERTSORTED) according to $pr$ (lines 45–47). Likewise, all non-pruned nodes in $\mathbf{D}_{\supset}$ (note that pruning always considers all node collections $\mathbf{Q}_{dup}$, $\mathbf{Q}$, $\mathbf{D}_{\checkmark}$, $\mathbf{D}_{\times}$ and $\mathbf{D}_{\supset}$) which are no longer supersets of any *known* minimal diagnosis, are added to $\mathbf{Q}$ again (lines 48–56). Finally, those minimal diagnoses returned in the previous iteration and consistent with the latest added measurement (the elements of $\mathbf{D}_{\checkmark}$), are put back to the ordered queue $\mathbf{Q}$ (lines 57–58). This is necessary to preserve the best-first property, as there might be "new" minimal diagnoses for the current DPI that are more probable than known ones.

PRUNE: Using its given argument $X$, the tree pruning runs through all (active and duplicate) nodes of the current tree (node collections $\mathbf{Q}_{dup}$, $\mathbf{Q}$, $\mathbf{D}_{\supset}$ and $\mathbf{D}_{calc}$ for call in line 33, and $\mathbf{Q}_{dup}$, $\mathbf{Q}$, $\mathbf{D}_{\supset}$, $\mathbf{D}_{\times}$ and $\mathbf{D}_{\checkmark}$ for call in line 44), and

- *(relabeling of old conflicts)* replaces by $X$ all labels $\mathsf{nd.cs}[i]$ which are proper supersets of $X$ for all nodes nd and for all $i = 1, \ldots, |\mathsf{nd.cs}|$, and
- *(deletion of redundant nodes)* deletes each redundant node nd for which $X$ is a witness of redundancy, and
- *(potential replacement of deleted nodes)* for each of the deleted nodes nd, if available, uses a suitable (non-redundant) node nd′ (constructed) from the elements of $\mathbf{Q}_{dup}$ to replace nd by nd′.

A node nd′ qualifies as a *replacement node for* nd iff nd′ is non-redundant and nd′ is *set*-equal to nd (i.e., the *sets*, not lists, of edge labels are equal). This node replacement is necessary from the point of view of completeness (cf. [8]). Importantly, duplicates ($\mathbf{Q}_{dup}$) must be pruned prior to all other nodes ($\mathbf{Q}$, $\mathbf{D}_{\supset}$, $\mathbf{D}_{\times}$, $\mathbf{D}_{\checkmark}$, $\mathbf{D}_{calc}$), to guarantee that all surviving nodes in $\mathbf{Q}_{dup}$ represent possible *non-redundant* replacement nodes at the time other nodes are pruned.

| | | | |
|---|---|---|---|
| $\mathcal{K}$ = | $\{ax_1 : A \to \neg B$ | $ax_2 : A \to B$ | $ax_3 : A \to \neg C$ |
| | $ax_4 : B \to C$ | $ax_5 : A \to B \vee C$ | $\}$ |
| $\mathcal{B} = \emptyset$ | | $P = \emptyset$ | $N = \{\neg A\}$ |

Table 1: Example DPI stated in propositional logic.

| iteration $j$ | $P'$ | $N'$ | $\mathbf{diag}(dpi_{j-1})$ | $\mathbf{conf}(dpi_{j-1})$ |
|---|---|---|---|---|
| 1 | – | – | $[1,3], [1,4], [2,3], [2,5]$ | $\langle 1,2\rangle, \langle 2,3,4\rangle, \langle 1,3,5\rangle, \langle 3,4,5\rangle$ |
| 2 | – | $A \to C$ | $[1,4], [2,5]$ | $\langle 1,2\rangle, \langle 2,4\rangle, \langle 1,5\rangle, \langle 4,5\rangle$ |
| 3 | – | $A \to C,\ A \to \neg B$ | $[1,4], [1,2,3,5]$ | $\langle 1\rangle, \langle 2,4\rangle, \boxed{\langle 3,4\rangle}, \langle 4,5\rangle$ |
| 4 | $A \to \neg C$ | $A \to C,\ A \to \neg B$ | $[1,4]$ | $\langle 1\rangle, \langle 4\rangle$ |

Table 2: Evolution of minimal diagnoses and minimal conflicts after successive extension of the example DPI $dpi_0$ (Tab. 1) by positive ($P'$) or negative ($N'$) measurements $m_i$ shown in Figures 1 and 2. Newly arisen conflicts (cf. $\mathcal{C}_{new}$ in Property 1.4) are framed.

Additionally, the argument $X$ is used to update the conflicts stored for reuse (set $\mathbf{C}_{calc}$), i.e., all proper supersets of $X$ are removed from $\mathbf{C}_{calc}$ and $X$ is added to $\mathbf{C}_{calc}$.

**Example 1** Consider the propositional DPI $dpi_0$ in Tab. 1. The goal is to locate the faulty axioms in the KB $\mathcal{K}$ that prevent the satisfaction of given measurements $P$ and $N$ (here, only one negative measurement $\neg A$ is given, i.e., $\neg A$ must not be entailed by the correct KB). We now illustrate the workings of HS-Tree (Fig. 2) and DynamicHS (Fig. 1) in terms of a complete sequential diagnosis session for $dpi_0$ under the assumption that $[1,4]$ is the actual diagnosis. The initial set of minimal conflicts and diagnoses can be read from Tab. 2.
*Inputs (Sequential Diagnosis):* We set $ld := 5$ (if existent, compute five diagnoses per iteration), heur to be the "split-in-half" heuristic [23] (prefers measurements the more, the more diagnoses they eliminate in the worst case), and $pr$ in a way the hitting set trees are constructed breadth-first.
*Notation in Figures:* Axioms $ax_i$ are simply referred to by $i$ (in node and edge labels). Numbers ⓘ indicate the chronological node labeling order. We tag conflicts $\langle 1, \ldots, k\rangle$ by $^C$ if they are freshly computed (FINDMINCONFLICT call, line 35, DLABEL), and leave them untagged if they result from a redundancy check and subsequent relabeling (lines 43–44, UPDATETREE). For the leaf nodes, we use the following labels: $\checkmark_{(\mathcal{D}_i)}$ for a minimal diagnosis, named $\mathcal{D}_i$, stored in $\mathbf{D}_{calc}$; $\times$ for a duplicate in HS-Tree (see (L2) criterion); $\times_{(\supset \mathcal{D}_i)}$ for a non-minimal diagnosis (stored in $\mathbf{D}_{\supset}$ by DynamicHS), where $\mathcal{D}_i$ is one diagnosis that proves the non-minimality; and $dup$ for a duplicate in DynamicHS (stored in $\mathbf{Q}_{dup}$). Branches representing minimal diagnoses are additionally tagged by a $^*$ if logical reasoning (FINDMINCONFLICT call, line 35, DLABEL function) is necessary to prove it is a diagnosis, and untagged otherwise (i.e., branch is diagnosis from previous iteration, stored in $\mathbf{D}_{\checkmark}$; only pertinent to DynamicHS).
*Iteration 1:* In the first iteration, HS-Tree and DynamicHS essentially build the same tree (compare Figs. 1 and 2). The only difference is that DynamicHS stores the duplicates and non-minimal diagnoses (labels $dup$ and $\times_{(\supset \mathcal{D}_i)}$), whereas HS-Tree discards them (labels $\times$ and $\times_{(\supset \mathcal{D}_i)}$). Note, duplicates are stored by DynamicHS at *generation* time (line 19),

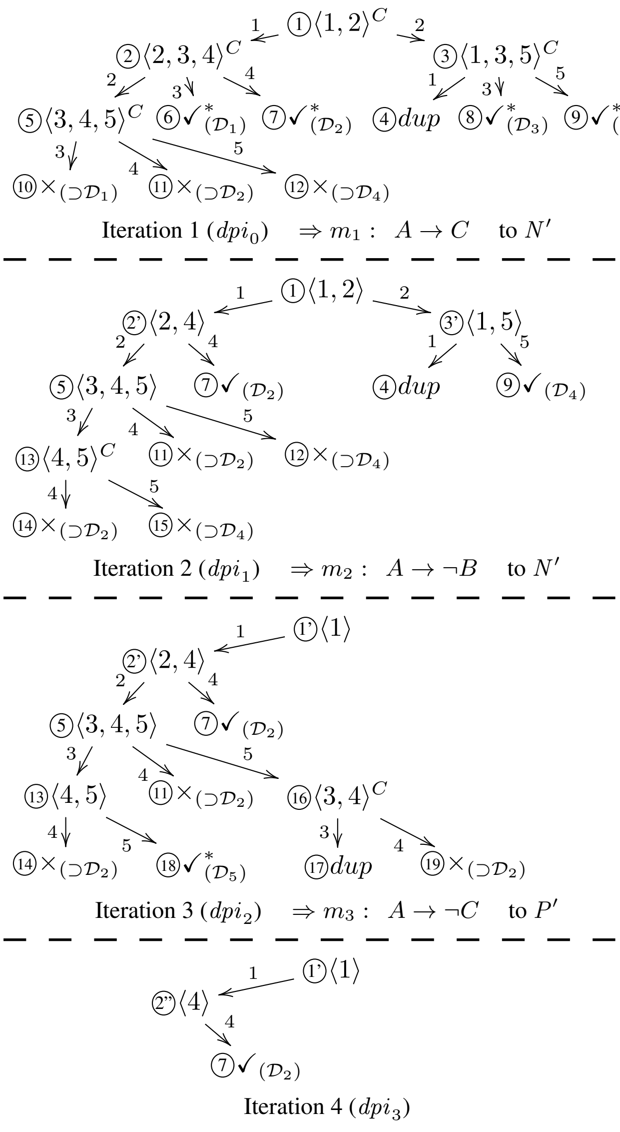


Figure 1: DynamicHS executed on example DPI given in Tab. 1.

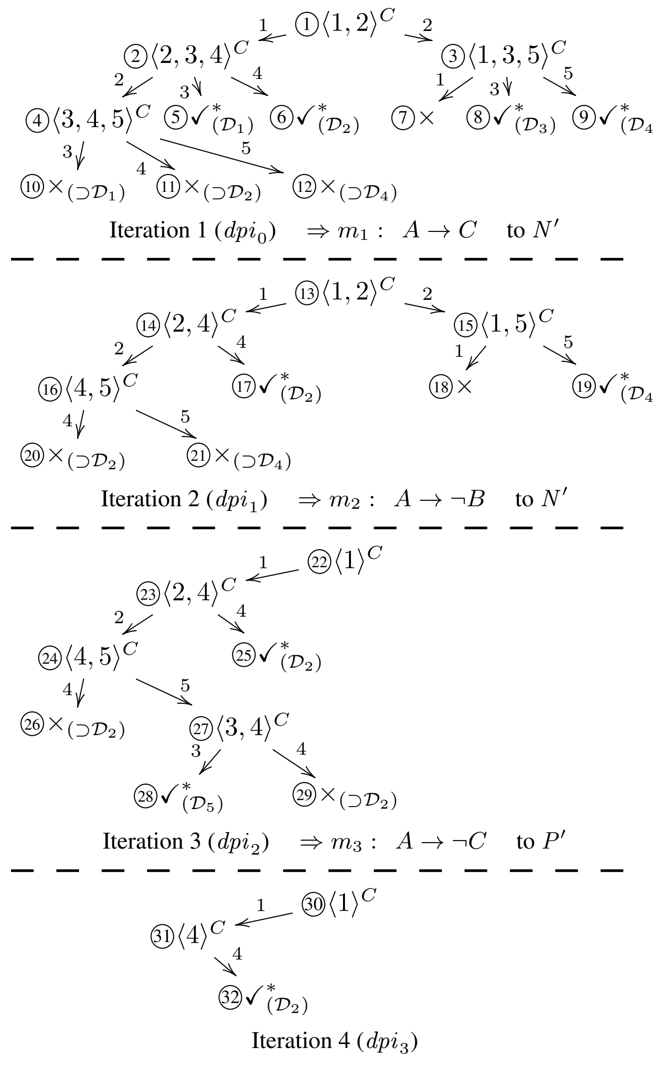


Figure 2: HS-Tree executed on example DPI given in Tab. 1.

hence the found duplicate ($dup$) has number ④ (not ⑦, as in HS-Tree). The diagnoses computed by both algorithms are $\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\} = \{[1,3],[1,4],[2,3],[2,5]\}$ (cf. Tab. 2). Notably, the returned diagnoses *must* be equal for both algorithms in any iteration (given same parameters $ld$ and $pr$) since both are sound, complete and best-first minimal diagnosis searches. Thus, when using the same measurement selection (and heuristic heur), both methods *must* also give rise to the same proposed next measurement $m_i$ in each iteration.

*First Measurement:* Accordingly, both algorithms lead to the first measurement $m_1 : A \to C$, which corresponds to the question "Does $A$ imply $C$?". This measurement turns out to be negative, e.g., by consulting an expert for the domain described by $\mathcal{K}$, and is therefore added to $N'$. This effectuates a transition from $dpi_0$ to a new DPI $dpi_1$ (which includes the additional element $A \to C$ in $N'$), and thus a change of the relevant minimal diagnoses and conflicts (see Tab. 2).

*Tree Update:* Starting from the second iteration ($dpi_1$), HS-Tree and DynamicHS behave differently. Whereas the former constructs a new tree from scratch for $dpi_1$, DynamicHS runs UPDATETREE to make the existing tree (built for $dpi_0$) reusable for $dpi_1$. In the course of this tree update, two witnesses of redundancy (minimal conflicts $\langle 2,4\rangle$, $\langle 1,5\rangle$) are found while analyzing the (conflicts along the) branches of the two invalidated diagnoses $[1,3]$ and $[2,3]$ (⑥ and ⑧). E.g., $\mathsf{nd} = [1,3]$ is redundant since the conflict $\mathsf{nd.cs}[2] = \langle 2,3,4\rangle$ is a proper superset of the *now minimal* conflict $X = \langle 2,4\rangle$ *and* nd's outgoing edge of $\mathsf{nd.cs}[2]$ is $\mathsf{nd}[2] = 3$ which is an element of $\mathsf{nd.cs}[2] \setminus X = \{3\}$. Since stored duplicates (here: only $[2,1]$) do not allow the construction of a replacement node for any of the redundant branches $[1,3]$ and $[2,3]$, both are removed from the tree. Further, the witnesses of redundancy replace the non-minimal conflicts at ② and ③, which is signified by the new numbers ②' and ③'.

Other than that, only a single further change is induced by UPDATETREE. Namely, the branch $[1, 2, 3]$, a non-minimal diagnosis for $dpi_0$, is returned to $\mathbf{Q}$ (unlabeled nodes) because there is no longer a diagnosis in the tree witnessing its non-minimality (both such witnesses $[1, 3]$ and $[2, 3]$ have been discarded). Note that, first, $[1, 2, 3]$ is in fact no longer a hitting set of all minimal conflicts for $dpi_1$ (cf. Tab. 2) and, second, there is still a non-minimality witness for all other branches (⑫ and ⑬) representing non-minimal diagnoses for $dpi_0$, which is why they remain labeled by $\times_{(\supset \mathcal{D}_i)}$.

*Iteration 2:* Observe the crucial differences between HS-Tree and DynamicHS in the second iteration (cf. Figs. 1 and 2).

First, while HS-Tree has to compute all conflicts that label nodes by (potentially expensive) FINDMINCONFLICT calls ($^C$ tags), DynamicHS has (cheaply) reduced existing conflicts during pruning (see above). Note, however, not all conflicts are necessarily always kept up-to-date after a DPI-transition (*lazy updating policy*). E.g., node ⑤ is still labeled by the non-minimal conflict $\langle 3, 4, 5 \rangle$ after UPDATETREE terminates. Hence, the subtree comprising nodes ⑬, ⑭ and ⑮ is not present in HS-Tree. Importantly, this lazy updating does not counteract sound- or completeness of DynamicHS, and the overhead incurred by such additional nodes is generally minor (all these nodes must be non-minimal diagnoses and are thus not further expanded). Second, the verification of the minimal diagnoses ($\mathcal{D}_2$, $\mathcal{D}_4$) found in iteration 2 requires logical reasoning in HS-Tree (see $^*$ tags of ✓ nodes) whereas it comes for free in DynamicHS (storage and reuse of $\mathbf{D}_{\checkmark}$).

*Remaining Execution:* After the second measurement $m_2$ is added to $N'$, causing a DPI-transition once again, DynamicHS reduces the conflict that labels the root node. This leads to the pruning of the complete right subtree. The left subtree is then further expanded in iteration 3 (see generated nodes ⑯, ⑰, ⑱ and ⑲) until the two leading diagnoses $\mathcal{D}_2 = [1, 4]$ and $\mathcal{D}_5 = [1, 2, 3, 5]$ are located and the queue of unlabeled nodes becomes empty (which proves that no further minimal diagnoses exist). Eventually, the addition of the third measurement $m_3$ to $P'$ brings sufficient information to isolate the actual diagnosis. This is reflected by a pruning of all branches except for the one representing the actual diagnosis $[1, 4]$.

*Comparison (expensive operations):* Generally, calling FINDMINCONFLICT (FC) is (clearly) more costly than REDUNDANT (RD), which in turn is more costly than a single consistency check (CC). HS-Tree requires 14/0/9, DynamicHS only 6/4/5 FC/RD/CC calls. This reduction of costly reasoning is the crucial advantage of DynamicHS over HS-Tree. □

**DynamicHS: Properties.** A proof of the following correctness theorem for DynamicHS can be found in [16, Sec. 12.4]:

**Theorem 1.** *DynamicHS is a sound, complete and best-first (as per $pr$) minimal diagnosis computation method.*

## 4 First Experiment Results

We provide a quick glance at first results of ongoing evaluations, where we compare HS-Tree and DynamicHS when applied for fault localization in (inconsistent) real-world KBs (same dataset as used in [23, Tabs. 8+12]). In this domain, HS-Tree is the most often used diagnosis search method. Average savings of DynamicHS over HS-Tree (both using same $ld := 6$ and random $pr$) wrt. *(i) number of expensive reasoner calls* (FINDMINCONFLICT), and *(ii) running time*, amounted to (i) 59 % and (ii) 42 %. Notably, DynamicHS required less time in *all* observed sequential sessions.

## 5 Related Work

Diagnosis algorithms can be compared along multiple axes, e.g., best-first vs. any-first, complete vs. incomplete, stateful vs. stateless, black-box (reasoner used as oracle) vs. glass-box (reasoner modified), and on-the-fly vs. preliminary (conflict computation). DynamicHS is best-first, complete, stateful, black-box, and on-the-fly. The most similar algorithms in terms of these features are: *(i) StaticHS* [18]: same features, but different focus, which is on reducing measurement conduction time (cf. Sec. 1); *(ii) Inv-HS-Tree* [24]: same features, but any-first, can handle problems with high-cardinality diagnoses that HS-Tree (and DynamicHS) cannot; *(iii) GDE and its successors* [2]: same features, but not black-box (e.g., uses bookkeeping of justifications for computed entailments). Generally, depending on the specific application setting of a diagnosis algorithm, different features are (un)necessary or (un)desirable. E.g., while HS-Tree-based tools are rather successful in the domain of knowledge-based systems, GDE-based ones prevail in the circuit domain (perhaps since bookkeeping methods are better suited for the entailment-justification structure of hardware than of KB systems[5]).

## 6 Conclusions

We suggest a novel sound, complete and best-first diagnosis computation method for sequential diagnosis based on Reiter's HS-Tree which aims at reducing expensive reasoning by the maintenance of a search data structure throughout a diagnostic session. First experimental results are very promising and attest the reasonability of the approach.

**Acknowledgments:** Thanks to Manuel Herold for implementing DynamicHS and the experiments.

## References

[1] R Abreu, P Zoeteweij, and AJ Van Gemund. Simultaneous debugging of software faults. *J. of Syst. and Softw.*, 84(4):573–586, 2011.

[2] J de Kleer and BC Williams. Diagnosing multiple faults. *Artif. Intell.*, 32(1):97–130, 1987.

[3] A Felfernig, G Friedrich, D Jannach, and M Stumptner. Consistency-based diagnosis of configuration knowledge bases. *Artif. Intell.*, 152(2):213–234, 2004.

[4] A Felfernig, E Teppan, G Friedrich, and K Isak. Intelligent debugging and repair of utility constraint sets in knowledge-based recommender applications. In *13th Int'l Conf. on Intell. User Interf.*, pp. 217–226, 2008.

[5] G Friedrich and K Schekotihin. A General Diagnosis Method for Ontologies. In *ISWC'05*, pp. 232–246, 2005.

[5] Based on a discussion in plenum at DX'17 in Brescia.

[6] G Friedrich, M Stumptner, and F Wotawa. Model-based diagnosis of hardware designs. *Artif. Intell.*, 111(1-2):3–39, 1999.

[7] BC Grau, I Horrocks, B Motik, B Parsia, P Patel-Schneider, and U Sattler. OWL 2: The next step for OWL. *JWS*, 6(4):309–322, 2008.

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

[9] M Horridge. *Justification based Explanation in Ontologies*. PhD thesis, Univ. of Manchester, 2011.

[10] U Junker. QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. In *AAAI'04*, pp. 167–172, 2004.

[11] A Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, Univ. of Maryland, 2006.

[12] J de Kleer, AK Mackworth, and R Reiter. Characterizing diagnoses and systems. *Artif. Intell.*, 56, 1992.

[13] C Meilicke. *Alignment Incoherence in Ontology Matching*. PhD thesis, Univ. Mannheim, 2011.

[14] KR Pattipati and MG Alexandridis. Application of heuristic search and information theory to sequential fault diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(4):872–887, 1990.

[15] R Reiter. A Theory of Diagnosis from First Principles. *Artif. Intell.*, 32(1):57–95, 1987.

[16] P Rodler. *Interactive Debugging of Knowledge Bases*. PhD thesis, Univ. Klagenfurt, 2015. *CoRR*, abs/1605.05950v1

[17] P Rodler. On active learning strategies for sequential diagnosis. In *DX'17*, pp. 264–283, 2018.

[18] P Rodler and M Herold. Statichs: A variant of reiter's hitting set tree for efficient sequential diagnosis. In *SOCS'18*, pp. 72–80, 2018.

[19] P Rodler and K Schekotihin. Reducing model-based diagnosis to KB debugging. In *DX'17*, pp. 284–296, 2018.

[20] P Rodler and W Schmid. On the impact and proper use of heuristics in test-driven ontology debugging. In *RuleML+RR'18*, pp. 164–184, 2018.

[21] P Rodler, W Schmid, and K Schekotihin. A generally applicable, highly scalable measurement computation and optimization approach to sequential model-based diagnosis. *CoRR*, abs/1711.05508, 2017.

[22] P Rodler, K Shchekotykhin, P Fleiss, and G Friedrich. RIO: Minimizing User Interaction in Ontology Debugging. In *RR'13*, pp. 153–167, 2013.

[23] K Shchekotykhin, G Friedrich, P Fleiss, and P Rodler. Interactive Ontology Debugging: Two Query Strategies for Efficient Fault Localization. *JWS*, 12-13:88–103, 2012.

[24] K Shchekotykhin, G Friedrich, P Rodler, and P Fleiss. Sequential diagnosis of high cardinality faults in KBs by direct diagnosis generation. In *ECAI'14*, pp. 813–818, 2014.

[25] F Wotawa. Fault localization based on dynamic slicing and hitting-set computation. In *10th Int'l Conf. on Quality Softw.*, pp. 161–170, 2010.

**Algorithm 2** DynamicHS

**Input:** tuple $\langle dpi, P', N', pr, ld, \mathbf{D}_{\checkmark}, \mathbf{D}_{\times}, \mathsf{state}\rangle$ comprising

- a DPI $dpi = \langle \mathcal{K}, \mathcal{B}, P, N\rangle$
- the acquired sets of positive ($P'$) and negative ($N'$) measurements so far
- a function $pr$ assigning a fault probability to each element in $\mathcal{K}$
- the number $ld$ of leading minimal diagnoses to be computed
- the set $\mathbf{D}_{\checkmark}$ of all elements of the set $\mathbf{D}_{calc}$ (returned by the previous DYNAMICHS run) which are minimal diagnoses wrt. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N'\rangle$
- the set $\mathbf{D}_{\times}$ of all elements of the set $\mathbf{D}_{calc}$ (returned by the previous DYNAMICHS run) which are no diagnoses wrt. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N'\rangle$
- state $= \langle \mathbf{Q}, \mathbf{Q}_{dup}, \mathbf{D}_{\supset}, \mathbf{C}_{calc}\rangle$ where
  - $\mathbf{Q}$ is the current queue of unlabeled nodes,
  - $\mathbf{Q}_{dup}$ is the current queue of duplicate nodes,
  - $\mathbf{D}_{\supset}$ is the current set of computed non-minimal diagnoses,
  - $\mathbf{C}_{calc}$ is the current set of computed minimal conflict sets.

**Output:** tuple $\langle \mathbf{D}, \mathsf{state}\rangle$ where

- $\mathbf{D}$ is the set of the $ld$ (if existent) most probable (as per $pr$) minimal diagnoses wrt. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N'\rangle$
- state is as described above

**procedure** DYNAMICHS($dpi, P', N', pr, ld, \mathbf{D}_{\checkmark}, \mathbf{D}_{\times}$, state)
  $\mathbf{D}_{calc} \leftarrow \emptyset$
  state $\leftarrow$ UPDATETREE($dpi, P', N', pr, \mathbf{D}_{\checkmark}, \mathbf{D}_{\times}$, state)
  **while** $\mathbf{Q} \neq [\,] \wedge (\,|\mathbf{D}_{calc}| < ld\,)$ **do**
    node $\leftarrow$ GETANDDELETEFIRST($\mathbf{Q}$) ▷ node is processed
    **if** node $\in \mathbf{D}_{\checkmark}$ **then** ▷ $\mathbf{D}_{\checkmark}$ includes only min...
      $L \leftarrow valid$ ▷ ...diags wrt. current DPI
    **else**
      $\langle L$, state$\rangle \leftarrow$ DLABEL($dpi, P', N', pr$, node, $\mathbf{D}_{calc}$, state)
    **if** $L = valid$ **then**
      $\mathbf{D}_{calc} \leftarrow \mathbf{D}_{calc} \cup \{$node$\}$ ▷ node is a min diag wrt. current DPI
    **else if** $L = nonmin$ **then**
      $\mathbf{D}_{\supset} \leftarrow \mathbf{D}_{\supset} \cup \{$node$\}$ ▷ node is a non-min diag wrt. current DPI
    **else**
      **for** $e \in L$ **do** ▷ $L$ is a min conflict wrt. current DPI
        node$_e \leftarrow$ APPEND(node, $e$) ▷ node$_e$ is generated
        node$_e$.cs $\leftarrow$ APPEND(node.cs, $L$)
        **if** node$_e \in \mathbf{Q}$ **then** ▷ node$_e$ is (*set-equal*) duplicate of node in $\mathbf{Q}$
          $\mathbf{Q}_{dup} \leftarrow$ INSERTSORTED(node$_e$, $\mathbf{Q}_{dup}, card, <$)
        **else**
          $\mathbf{Q} \leftarrow$ INSERTSORTED(node$_e$, $\mathbf{Q}, pr, >$)
  **return** $\langle \mathbf{D}_{calc}$, state$\rangle$

**procedure** DLABEL($\langle \mathcal{K}, \mathcal{B}, P, N\rangle, P', N', pr$, node, $\mathbf{D}_{calc}$, state)
  **for** nd $\in \mathbf{D}_{calc}$ **do**
    **if** node $\supset$ nd **then** ▷ node is a non-min diag
      **return** $\langle nonmin$, state$\rangle$
  **for** $\mathcal{C} \in \mathbf{C}_{calc}$ **do** ▷ $\mathbf{C}_{calc}$ includes conflicts wrt. current DPI
    **if** $\mathcal{C} \cap$ node $= \emptyset$ **then** ▷ reuse (a subset of) $\mathcal{C}$ to label node
      $X \leftarrow$ FINDMINCONFLICT($\langle \mathcal{C}, \mathcal{B}, P \cup P', N \cup N'\rangle$)
      **if** $X = \mathcal{C}$ **then**
        **return** $\langle \mathcal{C}$, state$\rangle$
      **else** ▷ $X \subset \mathcal{C}$
        state $\leftarrow$ PRUNE($X$, state, $pr$)
        **return** $\langle X$, state$\rangle$
  $L \leftarrow$ FINDMINCONFLICT($\langle \mathcal{K} \setminus$ node, $\mathcal{B}, P \cup P', N \cup N'\rangle$)
  **if** $L$ = 'no conflict' **then** ▷ node is a diag
    **return** $\langle valid$, state$\rangle$
  **else** ▷ $L$ is a *new* min conflict ($\notin \mathbf{C}_{calc}$)
    $\mathbf{C}_{calc} \leftarrow \mathbf{C}_{calc} \cup \{L\}$
    **return** $\langle L$, state$\rangle$

**procedure** UPDATETREE($dpi, P', N', pr, \mathbf{D}_{\checkmark}, \mathbf{D}_{\times}$, state)
  **for** nd $\in \mathbf{D}_{\times}$ **do** ▷ search for redundant nodes among invalidated diags
    **if** REDUNDANT(nd, $dpi$) **then**
      state $\leftarrow$ PRUNE($X$, state, $pr$)
  **for** nd $\in \mathbf{D}_{\times}$ **do** ▷ add all (non-pruned) nodes in $\mathbf{D}_{\times}$ to $\mathbf{Q}$
    $\mathbf{Q} \leftarrow$ INSERTSORTED(nd, $\mathbf{Q}, pr, >$)
    $\mathbf{D}_{\times} \leftarrow \mathbf{D}_{\times} \setminus \{$nd$\}$
  **for** nd $\in \mathbf{D}_{\supset}$ **do** ▷ add all (non-pruned) nodes in $\mathbf{D}_{\supset}$ to $\mathbf{Q}$, which...
    $nonmin \leftarrow false$ ▷ ...are no longer supersets of any diag in $\mathbf{D}_{\checkmark}$
    **for** nd$' \in \mathbf{D}_{\checkmark}$ **do**
      **if** nd $\supset$ nd$'$ **then**
        $nonmin \leftarrow true$
        **break**
    **if** $nonmin = false$ **then**
      $\mathbf{Q} \leftarrow$ INSERTSORTED(nd, $\mathbf{Q}, pr, >$)
      $\mathbf{D}_{\supset} \leftarrow \mathbf{D}_{\supset} \setminus \{$nd$\}$
  **for** $\mathcal{D} \in \mathbf{D}_{\checkmark}$ **do** ▷ add known min diags in $\mathbf{D}_{\checkmark}$ to $\mathbf{Q}$ to find diags...
    $\mathbf{Q} \leftarrow$ INSERTSORTED($\mathcal{D}, \mathbf{Q}, pr, >$) ▷ ...in best-first order (as per $pr$)
  **return** state