DynamicHS: Streamlining Reiter's Hitting-Set Tree for Sequential Diagnosis*

Patrick Rodler PATRICK.RODLER@AAU.AT
Universitätsstr. 65-67, 9020 Klagenfurt, Austria

Abstract

Given a system that does not work as expected, Sequential Diagnosis (SD) aims at suggesting a series of system measurements to isolate the true explanation for the system's misbehavior from a potentially exponential set of possible explanations. To reason about the best next measurement, SD methods usually require a sample of possible fault explanations at each step of the iterative diagnostic process. The computation of this sample can be accomplished by various diagnostic search algorithms. Among those, Reiter's HS-Tree is one of the most popular due its desirable properties and general applicability. Usually, HS-Tree is used in a stateless fashion throughout the SD process to (re)compute a sample of possible fault explanations in each iteration, each time given the latest (updated) system knowledge including all so-far collected measurements. At this, the built search tree is discarded between two iterations, although often large parts of the tree have to be rebuilt in the next iteration, involving redundant operations and calls to costly reasoning services.

As a remedy to this, we propose DynamicHS, a variant of HS-Tree that maintains state throughout the diagnostic session and additionally embraces special strategies to minimize the number of expensive reasoner invocations. DynamicHS provides an answer to a longstanding question posed by Raymond Reiter in his seminal paper from 1987, where he wondered whether there is a reasonable strategy to reuse an existing search tree to compute fault explanations after new system information is obtained.

We conducted extensive evaluations on real-world diagnosis problems from the domain of knowledge-based systems—a field where the usage of HS-Tree is state-of-the-art—under various diagnosis scenarios in terms of the number of fault explanations computed and the heuristic for measurement selection used. The results prove the reasonability of the novel approach and testify its clear superiority to HS-Tree wrt. computation time. More specifically: *(1)* DynamicHS required less time than HS-Tree in 96 % of the executed sequential diagnosis sessions. *(2)* DynamicHS exhibited substantial and statistically significant time savings over HS-Tree in most scenarios, with median and maximal savings of 52 % and 75 %, respectively. *(3)* The relative amount of saved time appears to neither depend on the number of computed fault explanations nor on the used measurement selection heuristic. *(4)* In the hardest (most time-intensive) cases per diagnosis scenario, DynamicHS achieved even higher savings than on average, and could avoid median and maximal time overheads of over 175 % and 800 %, respectively, as opposed to a usage of HS-Tree.

Remarkably, DynamicHS achieves these performance improvements while preserving all desirable properties as well as the general applicability of HS-Tree.

* Earlier and significantly shorter versions (Rodler, 2020a, 2020b, 2020c) of this paper were accepted and presented at the *31st International Workshop on Principles of Diagnosis (DX'20)*, the *Symposium on Combinatorial Search (SoCS'20)*, as well as at the *24th European Conference on Artificial Intelligence (ECAI'20)*. The present version extends these earlier versions in various regards, providing i.a. a more detailed discussion of the proposed algorithm, a correctness proof, additional examples, a more comprehensive treatment of related works, and a substantially augmented evaluation section.

1. Introduction

Model-based diagnosis (Reiter, 1987; de Kleer & Williams, 1987) is a popular, well-understood, domain-independent and principled paradigm that has over the last decades found widespread adoption for troubleshooting systems as different as programs, circuits, physical devices, knowledge bases, spreadsheets, production plans, robots, vehicles, or aircrafts (Ng, 1990; Sachenbacher, Malik, & Struss, 1998; Gorinevsky, Dittmar, Mylaraswamy, & Nwadiogbu, 2002; Steinbauer, Wotawa, et al., 2005; Felfernig, Mairitsch, Mandl, Schubert, & Teppan, 2009; Feldman, Provan, & van Gemund, 2010; Jannach & Engler, 2010; Wotawa, 2010; Rodler, 2015; Rodler & Teppan, 2020). The theory of model-based diagnosis (Reiter, 1987) assumes a *system* that is composed of a set of *components*, and a formal *system description*. The latter is given as a logical knowledge base and can be used to derive the expected behavior of the system by means of automated deduction systems. If the predicted system behavior, under the assumption that all components are functioning normally, is not in line with *observations* made about the system, the goal is to locate the abnormal system components that are responsible for this discrepancy. Given a *diagnosis problem instance (DPI)*—consisting of the system description, the system components, and the observations—a *diagnosis* is a 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 different diagnoses given the initial system observations. However, only one of the diagnoses (which we refer to as the *actual diagnosis*) pinpoints the actually faulty components. To isolate the actual diagnosis, *sequential diagnosis* (de Kleer & Williams, 1987) methods collect additional system observations (called *measurements*) to gradually refine the set of diagnoses.

The basic idea behind measurements is to exploit the fact that different diagnoses predict different system behaviors or properties (e.g., intermediate values or outputs). Observing a system aspect for which the predictions of diagnoses disagree then leads to the invalidation of those diagnoses whose predictions are inconsistent with the observation. Since *(i)* the amount of useful information gained differs significantly for different possible measurements, *(ii)* performing a measurement is usually costly (as, e.g., a human operator needs to take action), and *(iii)* determining (nearly) optimal measurement points is beyond the reach of humans for sufficiently complex systems, it is important to have a diagnosis system provide appropriate measurement suggestions automatically. To this end, several (efficiently computable) *heuristics* (Moret, 1982; de Kleer & Williams, 1987; Pattipati & Alexandridis, 1990; Shchekotykhin, Friedrich, Fleiss, & Rodler, 2012; Rodler, Shchekotykhin, Fleiss, & Friedrich, 2013; Rodler, 2016, 2018; Rodler & Schmid, 2018), that define the goodness of measurements based on various information-theoretic considerations, have been proposed to deal with the problem that optimal measurement proposal is NP-hard (Hyafil & Rivest, 1976). These heuristics, in principle, assess the impact of measurements by comparing a-priori (before measurement is taken) and expected a-posteriori (after measurement outcome is known) situations regarding the (known) diagnoses and their properties (e.g., probabilities). That is, the evaluation of the expected utility of measurements requires the knowledge of a sample of diagnoses. This sample is often termed the *leading diagnoses* and usually defined as the best diagnoses according to some preference criterion such as minimal cardinality (where a minimal *number* of components is assumed abnormal) or maximal probability. Beside such criteria, for computational and pragmatic reasons, a least requirement is usually that only minimal diagnoses are computed (cf. the "principle of parsimony" (Reiter, 1987)). A *minimal diagnosis* is one that does not strictly contain (in the sense of set-inclusion) any diagnosis.

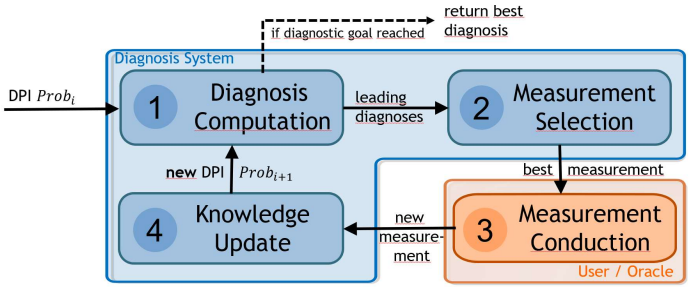


Figure 1: Overview of a generic sequential diagnosis process. Note that the diagnosis problem instance (DPI) changes between phases 4 and 1 in each iteration, i.e., $Prob_{i+1}$ is the reault of adding the new measurement obtained from phase 3 to $Prob_i$.

A generic *sequential diagnosis process* (see Fig. 1) can thus be thought of as a recurring execution of *(i)* the computation of a set of leading minimal diagnoses, *(ii)* the selection of the most informative measurement based on these, *(iii)* the conduction of measurement actions (by some oracle or user), and *(iv)* the exploitation of the measurement outcome to refine the system knowledge. This iterative process continues until sufficient diagnostic certainty is obtained (e.g., one diagnosis has overwhelming probability).

Decisive factors for successful and practicable sequential diagnosis are a low number of measurements and a low cost per measurement (effort for human operator), as well as reasonable system computation times (waiting time of human operator). While the first two factors depend on a suitable measurement selection, the latter depends largely on the efficiency of the algorithm for (leading) diagnoses computation (Shchekotykhin, Friedrich, Rodler, & Fleiss, 2014; Rodler, Schmid, & Schekotihin, 2017; Rodler & Elichanova, 2020).

One of the most popular and widely used such algorithms is Reiter's HS-Tree (Reiter, 1987), which is adopted in various domains such as for the debugging of software (Wotawa, 2010; Abreu, Zoeteweij, & van Gemund, 2011) or ontologies and knowledge bases (Friedrich & Shchekotykhin, 2005; Kalyanpur, 2006; Horridge, 2011; Meilicke, 2011; Rodler, 2015), or for the diagnosis of hardware (Friedrich, Stumptner, & Wotawa, 1999; Zaman, Steinbauer, Maurer, Lepej, & Uran, 2013) or of recommender and configuration systems (Felfernig, Friedrich, Jannach, & Stumptner, 2004; Felfernig, Teppan, Friedrich, & Isak, 2008). The main reasons for the widespread adoption of HS-Tree are that *(i) it is broadly applicable*, because all it assumes is a system description in some (monotonic[1]) knowledge representation language for which a sound and complete inference method exists, *(ii) it is sound and complete*, as it computes only and all[2] 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

[1] *Monotonicity* means that the entailment relation $\models$ of the logical language satisfies $\alpha \models \beta \implies \alpha \cup \gamma \models \beta$ for all sets of sentences α, β, γ over this language. That is, adding additional sentences γ to a knowledge base α does not invalidate any conclusions that could be drawn from α beforehand.

[2] Given a non-empty set of minimal diagnoses for a DPI, the problem of deciding whether there is a minimal diagnosis for this DPI which is not in the given set is NP-hard (Bylander, Allemang, Tanner, & Josephson, 1991). Hence, the completeness of HS-Tree (or any other algorithm that computes minimal diagnoses) guarantees the computation of all minimal diagnoses for an arbitrary DPI (only) if unlimited time and memory are assumed. Note that in practical diagnosis applications the computation of (even multiple) minimal diagnoses is often accomplishable in reasonable time.

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 (Reiter, 1987) 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 study or algorithm has yet been proposed that sheds light on this very question.

As a result, sequential approaches which draw on HS-Tree for diagnosis computation have to 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 data structure (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. What even exacerbates this fact is that these computations 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, a single consistency check performed by an inference service is already 2NEXPTIME-complete (Grau, Horrocks, Motik, Parsia, Patel-Schneider, & Sattler, 2008b).

Motivated by that, the **contribution of this work** is the development of DynamicHS, a novel *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 main objective of this new algorithm is to allow for more efficient computations than HS-Tree while maintaining all the aforementioned advantages (generality, soundness, completeness, best-first property) of the latter. For that purpose, DynamicHS implements sophisticated techniques such as a time-efficient lazy update strategy or procedures to minimize expensive reasoning by trading it for cheaper (reasoning or set) operations.

In extensive evaluations we put DynamicHS to the test, using a benchmark of 20 real-world diagnosis problems from the domain of knowledge-based systems, which is one prominent field where HS-Tree is the prevalent tool for (sequential) diagnosis computation (Friedrich & Shchekotykhin, 2005; Kalyanpur, 2006; Schlobach, Huang, Cornet, & van Harmelen, 2007; Horridge, 2011; Meilicke, 2011; Shchekotykhin et al., 2012; Rodler, 2015; Fu, Qi, Zhang, & Zhou, 2016; Baader, Kriegel, Nuradiansyah, & Penaloza, 2018). One major cause for the attractivity of HS-Tree in this domain are its mild assumptions made about the system description language and its resulting independence from the particular used language and reasoning engine, combined with the fact that a multitude of different logical languages are adopted in knowledge-based systems (with the goal to optimally trade off expressivity with reasoning efficiency). For each diagnosis problem in our dataset, we experimented with DynamicHS and HS-Tree under various *diagnosis scenarios* wrt. the number of diagnoses computed per sequential diagnosis iteration and the employed measurement selection heuristic. The **main insights from our evaluations** are:

- DynamicHS is superior to HS-Tree in terms of computation time in 99.4 % of the investigated diagnosis scenarios, and in 96 % of all single sequential diagnosis sessions run. Roughly,

these savings are achieved by trading less time (fewer redundant operations and reasoner calls) for more space (statefulness), where the additional memory required by DynamicHS was reasonable in the vast majority of the scenarios; and, whenever HS-Tree was applicable in our experiments in terms of memory requirements, DynamicHS was so as well.
- The runtime savings over HS-Tree achieved by DynamicHS are substantial and statistically significant in most scenarios, and reach median and maximal values of 52 % and 75 %. That is, HS-Tree requires up to an *average* of four times the computation time of DynamicHS in the tested diagnosis scenarios. In *single* diagnosis sessions, we observed that it took HS-Tree up to more than nine times as much time as DynamicHS—notably, while both algorithms always compute *the same* solutions.
- The median runtime savings of DynamicHS per scenario appear to be neither dependent on the number of diagnoses computed nor on the measurement selection heuristic used.
- Considering the hardest cases per diagnosis scenario, which were up to one order of magnitude harder than the average cases, the time savings obtained by means of DynamicHS are even more substantial than on average, and reach median and maximal values of 64 % and 89 %, respectively.

The **organization of this work** is as follows: Preliminaries on (sequential) model-based diagnosis are given in Sec. 2. The novel algorithm DynamicHS is discussed in detail in Sec. 3, where we follow a didactical approach by "deriving" DynamicHS from Reiter's well-known HS-Tree algorithm. Moreover, we prove the algorithm's correctness and detail some advanced techniques used by DynamicHS regarding tree management, reasoning operations and node storage. Related works are reviewed in Sec. 4. In Sec. 5, we describe our experimental settings, the used test dataset, and thoroughly discuss the obtained results. More specifically, we first study the performance of DynamicHS, and then provide explanations for the outcomes through additional analyses. Finally, Sec. 6 concludes this work.

2. Preliminaries

We briefly characterize the basic technical concepts used throughout this work, based on the framework of (Shchekotykhin et al., 2012; Rodler, 2015) which is (slightly) more general (Rodler & Schekotihin, 2018) than Reiter's theory of diagnosis (Reiter, 1987). In other words, any model-based diagnosis problem that can be formulated by means of Reiter's theory, can be equivalently stated by the following framework. The main reason for using this more general framework is its ability to handle negative measurements (things that must *not* be true for the diagnosed system) which are helpful, e.g., for diagnosing knowledge bases (Felfernig et al., 2004; Shchekotykhin et al., 2012; Schekotihin, Rodler, Schmid, Horridge, & Tudorache, 2018c).

2.1 Diagnosis Problem Instance (DPI)

We assume that the diagnosed system, consisting of a set of components $\{c_1, \ldots, c_k\}$, 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

Table 1: Example DPI stated in propositional logic.

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

nominal behavior of c_i (*weak fault model*[3]). For instance, 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 (gates) 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 (de Kleer & Williams, 1987; Reiter, 1987; Felfernig et al., 2004). 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 properties that must not hold for the system. For example, if the diagnosed system is a logical description of a university domain, a negative test case could be $\forall X\ (researcher(X) \rightarrow professor(X))$, i.e., that it must not be true that each researcher is a professor. We call $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ a *diagnosis problem instance (DPI)*.

Example 1 Tab. 1 depicts an example of a DPI, formulated in propositional logic. The "system" (which is the knowledge base itself in this case) comprises five "components" $c_1, \ldots, c_5$, and the "nominal behavior" of c_i is given by the respective axiom $ax_i \in \mathcal{K}$. There is neither any background knowledge ($\mathcal{B} = \emptyset$) nor any positive test cases ($P = \emptyset$) available from the start. But, there is one negative test case (i.e., $N = \{\neg A\}$), which postulates that $\neg A$ must *not* be an entailment of the correct system (knowledge base). Note, however, that $\mathcal{K}$ (i.e., the assumption that all "components" work nominally) in this case does entail $\neg A$ (e.g., due to the axioms ax_1, ax_2) and therefore some axiom in $\mathcal{K}$ must be faulty (i.e., some "component" is not healthy). □

2.2 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 a DPI 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 (de Kleer, Mackworth, & Reiter, 1992)), i.e., the set of all diagnoses is exactly given by the set of all supersets of all minimal diagnoses. Therefore, diagnosis approaches often 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 (unknown) *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.

[3] *Weak fault models*, in constrast to *strong fault models*, define *only the normal behavior* of the system components, and do not specify any behavior in case components are at fault (Feldman, Provan, & van Gemund, 2008). The weak fault model is also referred to as "Ignorance of Abnormal Behavior" property (de Kleer, 2008).

Example 2 For our DPI dpi in Tab. 1 we have four minimal diagnoses, given by $\mathcal{D}_1 := [ax_1, ax_3]$, $\mathcal{D}_2 := [ax_1, ax_4]$, $\mathcal{D}_3 := [ax_2, ax_3]$, and $\mathcal{D}_4 := [ax_2, ax_5]$, i.e., $\mathbf{diag}(dpi) = \{\mathcal{D}_1, \ldots, \mathcal{D}_4\}$.[4] For instance, $\mathcal{D}_1$ is a minimal diagnosis as $(\mathcal{K} \setminus \mathcal{D}_1) \cup \mathcal{B} \cup P = \{ax_2, ax_4, ax_5\}$ is both consistent and does not entail the given negative test case $\neg A$. □

2.3 Conflicts

Useful for the computation of diagnoses is the concept of a conflict (de Kleer & Williams, 1987; Reiter, 1987). 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 a DPI 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 (Reiter, 1987), 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 $S_i \in S$.

Example 3 For our running example, dpi, in Tab. 1, there are four minimal conflicts, given by $\mathcal{C}_1 := \langle ax_1, ax_2 \rangle$, $\mathcal{C}_2 := \langle ax_2, ax_3, ax_4 \rangle$, $\mathcal{C}_3 := \langle ax_1, ax_3, ax_5 \rangle$, and $\mathcal{C}_4 := \langle ax_3, ax_4, ax_5 \rangle$, i.e., $\mathbf{conf}(dpi) = \{\mathcal{C}_1, \ldots, \mathcal{C}_4\}$.[5] For instance, $\mathcal{C}_4$, in CNF equal to $(\neg A \vee \neg C) \wedge (\neg B \vee C) \wedge (\neg A \vee B \vee C)$, is a conflict because, adding the unit clause (A) to this CNF yields a contradiction, which is why the negative test case $\neg A$ is an entailment of $\mathcal{C}_4$. The minimality of the conflict $\mathcal{C}_4$ can be verified by rotationally removing from $\mathcal{C}_4$ a single axiom at the time and controlling for each so obtained subset that this subset is consistent and does not entail $\neg A$.

For example, the minimal diagnosis $\mathcal{D}_1$ (see Example 2) is a hitting set of all minimal conflict sets because each conflict in $\mathbf{conf}(dpi)$ contains ax_1 or ax_3. It is moreover a *minimal* hitting set since the elimination of ax_1 implies an empty intersection with, e.g., $\mathcal{C}_1$, and the elimination of ax_3 means that, e.g., $\mathcal{C}_4$ is no longer hit. □

Literature offers a variety of algorithms for conflict computation, e.g., (Junker, 2004; Marques-Silva, Janota, & Belov, 2013; Shchekotykhin, Jannach, & Schmitz, 2015). Given a DPI $dpi = \langle \mathcal{K}, \mathcal{B}, P, N \rangle$ as input, one call to such an algorithm returns one minimal conflict for dpi, if existent, and 'no conflict' otherwise. All algorithms require an appropriate theorem prover that is used as an oracle to perform consistency checks over the logic by which the DPI is expressed. In the worst case, none of the available algorithms requires fewer than $O(|\mathcal{K}|)$ theorem prover calls per conflict computation (Marques-Silva, Janota, & Belov, 2013). The performance of diagnosis computation methods depends largely on the complexity of consistency checking for the used logic and on the number of consistency checks executed (cf., e.g., (Pill, Quaritsch, & Wotawa, 2011)). Since consistency checking is often NP-complete or beyond for practical problems (Kalyanpur, 2006; Grau et al., 2008b; Horridge, 2011; Romero, Grau, & Horrocks, 2012; Rodler, Jannach, Schekotihin, & Fleiss, 2019), and diagnostic algorithms have no influence on the used system description language, it is pivotal to keep the number of conflict computations at a minimum.

[4] In this work, we denote diagnoses by square brackets.

[5] In this work, we denote conflicts by angle brackets.

2.4 Sequential Diagnosis (SD) Problem

We now define the sequential diagnosis (SD) problem, which calls for a set of measurements for a given DPI, such that adding these measurements to the DPI implies diagnostic certainty, i.e., a single remaining minimal diagnosis for the new DPI. In the optimization version (OptSD), the goal is to find such a measurement set of minimal cost.

Problem 1 ((Opt)SD)**.** ***Given:*** *A DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$. ***Find:*** *A (minimal-cost) set of measurements* $P' \cup N'$ *such that* $|\mathbf{diag}(\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle)| = 1$.[6]

A minimal-cost set of measurements $P' \cup N'$ minimizes $\sum_{m \in P' \cup N'} \mathsf{cost}(m)$, where $\mathsf{cost}(m)$ is the cost value assigned to measurement m. Possible cost functions range from simple unit cost assumptions for measurements ($\mathsf{cost}(m) = 1$ for all m) to sophisticated measurement time, effort, or complexity evaluations (Rodler, 2015; Rodler et al., 2017, 2019).

Example 4 Let $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ be the example DPI described in Tab. 1, and assume that $\mathcal{D}^* = \mathcal{D}_2$, i.e., $[ax_1, ax_4]$ is the actual diagnosis. Then, one solution to Problem 1 is $P' = \{A \wedge B\}$ and $N' = \{C\}$. In other words, $\mathbf{diag}(\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle) = \{\mathcal{D}_2\}$. An alternative solution would be $P' = \{A \rightarrow \neg C\}$ and $N' = \{A \rightarrow C, A \rightarrow \neg B\}$. If the solution cost is quantified by the number of measurements, then the first solution has a cost of 2, whereas the second has a cost of 3. We would thus prefer the first solution. If we alternatively adopt a more fine-grained cost analysis, considering also the estimated complexity of the measurements, e.g., that negation and implication are harder (to understand for an interacting domain expert who labels sentences as positive or negative measurements) than conjunction, then the first solution would be even more preferred to the second than under the assumption of uniform measurement costs. □

2.5 Computational Issues in Sequential Diagnosis

In principle, the OptSD problem can be tackled by appropriate *measurement point*[7] *selection*[8] (de Kleer & Williams, 1987) approaches. However, due to the NP-hardness of OptSD (Hyafil & Rivest, 1976), sequential diagnosis systems, in general, can merely attempt to approximate its optimal solution. This is usually accomplished by means of a one-step-lookahead approach (de Kleer, Raiman, & Shirley, 1992) which evaluates (the gain of) possible measurement points based on the a-posteriori situations encountered for its different outcomes, weighted by their respective (estimated) probabilities. To make these estimations, a sample of (minimal) diagnoses, the *leading diagnoses*, is usually taken as a basis (de Kleer & Williams, 1987; Feldman et al., 2010; Shchekotykhin et al.,

[6] There are different characterizations of the SD Problem in literature. E.g., it is sometimes formulated as the problem of finding a policy that suggests the next measurement given any possible situation that might be encountered starting from a given DPI. For the purpose of this paper, we basically stick to the conceptualization of (Reiter, 1987), where measurements are sets of logical sentences.

[7] In the scope of this work, the relation between measurement and measurement point is as follows. A *measurement* is a logical sentence that describes the outcome of a *measuring action* at some predefined *measurement point*. For instance, when diagnosing a circuit, the measurement action could be the application of a voltmeter, and the measurement point a particular terminal of a gate in a circuit. The outcome would then be whether the measured wire w is high (1) or low (0), yielding one of the (positive) measurements $w = 1$ or $w = 0$, respectively.

[8] What we call *measurement point selection* in this work is often simply referred to as *measurement selection* in literature, e.g., (Friedrich & Nejdl, 1992; Chen, Quan Hu, & Tang, 2015). Moreover, the measurement point selection might also include a prior *measurement point computation*, for (e.g., knowledge-based) systems for which the possible measurement points are not explicitly given (Shchekotykhin et al., 2012; Rodler et al., 2017).

2012; Rodler, 2015). In the context of one-step-lookahead analysis, a range of measurement point evaluation *heuristics* have been suggested in literature, such as the measurement point's information gain (de Kleer & Williams, 1987), its ability to guarantee a maximal worst-case diagnoses invalidation rate (Moret, 1982; Shchekotykhin et al., 2012), a dynamic combination thereof (Rodler et al., 2013), or the measurement point's goodness wrt. various active learning criteria (Rodler, 2016, 2018; Rodler & Schmid, 2018). Besides these more sophisticated assessment criteria, a common minimal requirement to a rational measurement point selection strategy is that it suggests only *discriminating measurement points*, i.e., for each measurement outcome there must be at least one leading diagnosis that is inconsistent with this outcome. In other words, proposing a discriminating measurement point guarantees the invalidation of some spurious fault hypothesis (or: the gain of some useful information) already a-priori, irrespective of the measurement outcome.

While appropriate measurement point selection is one important factor determining the overall efficiency of the diagnostic process, not only the cost associated with the *measurement conduction* is of relevance from the viewpoint of a user interacting with a sequential diagnosis engine. In fact, there is another crucial factor which is left implicit in Problem 1, namely the question how efficiently the *measurement point computation* is accomplished by the diagnosis system. Because, the *overall* (user) time to diagnose a system is given by the overall measurement conduction time *plus* the overall system computation time for the proposal of the measurement points. The following problem makes this explicit:

Problem 2 (Efficient (Opt)SD)**.** ***Given:*** *A DPI* $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$. ***Find:*** *A solution to (Opt)SD in minimal time.*

Approaching a solution to Problem 2 involves the minimization of what we call the system *reaction time*, i.e., the computation time required by the system after being informed about the outcome of one measuring action until the provision of the next measurement point. The reaction time is composed of the time required for *(i)* the *system knowledge update* (based on the information acquired through a new measurement), *(ii)* the (leading) *diagnosis computation*, as well as for *(iii)* the *measurement point selection*. These main factors affecting the sequential diagnosis time are summarized in Fig. 2.

Whereas the efficiency-optimization of measurement point selection algorithms[9] can be largely or even fully decoupled from diagnosis computation and knowledge update algorithms, the latter two might need to be studied in common, depending on whether the diagnosis computation is stateless (Shchekotykhin et al., 2012, 2014) or stateful (de Kleer & Williams, 1987; Siddiqi & Huang, 2011; Rodler, 2015). More specifically, the elaborateness of the knowledge update process might range from almost trivial (addition of a new measurement to the DPI; update of probabilities) in the stateless case, to very sophisticated in the stateful case, where, e.g., the state of a persistent data structure used throughout sequential diagnosis must be additionally updated.

We present in this work DynamicHS, a stateful diagnosis computation algorithm that addresses Problem 2 by *focusing on the efficiency of the diagnosis computation and knowledge update processes* (cf. Fig. 2). In fact, we will provide empirical evidence (in Sec. 5) that the novel algorithm is more time-efficient for sequential diagnosis than Reiter's HS-Tree, which is a state-of-the-art diagnosis computation approach, e.g., in the knowledge base and ontology debugging domains

[9]Measurement selection algorithms (de Kleer & Williams, 1987) are given a set of leading diagnoses, the current system knowledge (DPI), as well as the available probabilistic information and a selection heuristic, and should output "the best" next measurement according to the given heuristic.

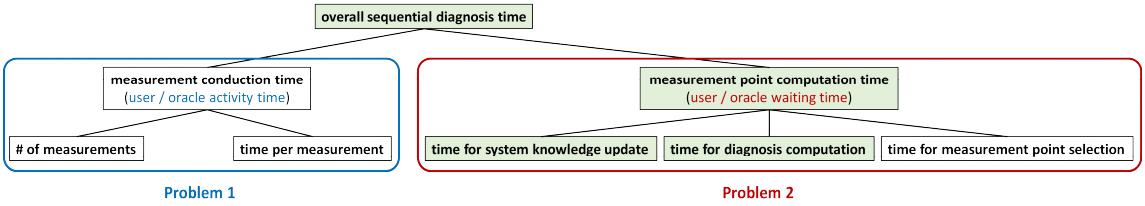


Figure 2: Categorization of the factors that influence the overall sequential diagnosis time. The blue / red frame refers to the OptSD / Efficient OptSD problem. The factors shaded green are those addressed by the DynamicHS algorithm suggested in this work. For the figure to reflect a more general viewpoint (including also resources other than time), one may replace each occurrence of "time" in the figure by "cost".

(Friedrich & Shchekotykhin, 2005; Kalyanpur, 2006; Schlobach et al., 2007; Horridge, 2011; Meilicke, 2011; Shchekotykhin et al., 2012; Rodler, 2015; Fu et al., 2016; Baader et al., 2018). In particular, DynamicHS aims at achieving an efficiency gain over HS-Tree while *computing the same solution* to Problem 1 as HS-Tree, while *preserving all the (desirable) properties* of HS-Tree, and while *maintaining full compatibility with other approaches* that tackle Problem 2 from the perspective of making the measurement point selection more efficient (such as (Rodler, Schmid, & Schekotihin, 2018)), or that attack Problem 1 to achieve a reduction of measurement conduction time (such as (Shchekotykhin et al., 2012; Rodler et al., 2013; Rodler & Eichholzer, 2019)).

3. DynamicHS Algorithm

3.1 Outline of DynamicHS

3.1.1 Motivation and Principle

Starting from Reiter's HS-Tree, the main observation that builds the ground for the development of DynamicHS is that, throughout a sequential diagnosis session, the initially given DPI is subject to gradual change. That is, new measurements are successively added to it, and usually significant portions of a built hitting set tree remain unaffected after the transition from one DPI to the next. Thus, discarding the existing tree and (re)constructing a similar tree in the next iteration potentially involves a significant number of redundant (and expensive) operations. With DynamicHS, we account for that by proposing to replace a discard-and-rebuild paradigm by an adapt-and-reuse one—that is, the built hitting set tree is persistently maintained and accordingly adapted each time a new measurement comes in. These tree updates, which mainly include tree pruning and node relabeling steps, are required to guarantee the correctness of the diagnosis computation in the next iteration, i.e., for the *newest* DPI. The underlying rationale of DynamicHS is to avoid unnecessary repeated actions and to cleverly leverage the reused search data structure to trade expensive reasoning against cheaper operations, such as set comparisons or less expensive (but equally informative) reasoning.

3.1.2 Inputs and Outputs

DynamicHS (depicted by Alg. 3 on page 18) is a procedure that computes a set of minimal diagnoses for a DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle$ in best-first order (as per some given goodness criterion, which we assume to be a probability measure pr in this work). DynamicHS accepts the following argu-

Algorithm 1 Sequential Diagnosis

Input: DPI $dpi_0 := \langle \mathcal{K}, \mathcal{B}, P, N \rangle$, probability measure pr (to compute diagnoses probabilities), number ld (≥ 2) of minimal diagnoses to be computed per iteration, heuristic heur for measurement selection, boolean $dynamic$ that governs which diagnosis computation algorithm is used (DynamicHS if true, HS-Tree otherwise)
Output: $\{\mathcal{D}\}$ where $\mathcal{D}$ is the final diagnosis after solving SD (Problem 1)

```
1:  P' ← ∅, N' ← ∅                                              ▷ performed measurements
2:  D✓ ← ∅, D× ← ∅                                              ▷ variables describing state...
3:  state ← ⟨[ [] ], [ ], ∅, ∅⟩                                 ▷ ...of DynamicHS tree
4:  while true do
5:      if dynamic then
6:          ⟨D, state⟩ ← DYNAMICHS(dpi_0, P', N', pr, ld, D✓, D×, state)
7:      else
8:          D ← HS-TREE(dpi_0, P', N', pr, ld)
9:      if |D| = 1 then return D
10:     mp ← COMPUTEBESTMEASPOINT(D, dpi_0, P', N', pr, heur)
11:     outcome ← PERFORMMEAS(mp)                               ▷ oracle inquiry (user interaction)
12:     ⟨P', N'⟩ ← ADDMEAS(mp, outcome, P', N')
13:     if dynamic then
14:         ⟨D✓, D×⟩ ← ASSIGNDIAGSOKNOK(D, dpi_0, P', N')
```

ments: *(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', respectively, *(3)* a probability measure pr which is exploited to compute diagnoses in descending order based on their probabilities, *(4)* a stipulated number $ld \geq 2$ of diagnoses to be returned,[10] *(5)* the set of those diagnoses returned by the previous run of DynamicHS that are consistent ($\mathbf{D}_{\checkmark}$) and those that are inconsistent ($\mathbf{D}_{\times}$) with the last added measurement, and *(6)* a tuple of variables state, which altogether describe DynamicHS's current state.

3.1.3 EMBEDDING IN SEQUENTIAL DIAGNOSIS PROCESS

Alg. 1 sketches a generic sequential diagnosis algorithm and shows how it accommodates DynamicHS (line 6) or, alternatively, Reiter's HS-Tree (line 8), as methods for iterative diagnosis computation. The algorithm reiterates a while-loop (line 4) until the solution space of minimal diagnoses includes only a single element.[11] Since both DynamicHS and HS-Tree are complete (see later) and always attempt to compute at least two diagnoses ($ld \geq 2$), this stop criterion is met iff a diagnoses set $\mathbf{D}$ with $|\mathbf{D}| = 1$ is output (line 9). 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 (COMPUTEBESTMEASPOINT, line 10), using the current system information—dpi_0, $\mathbf{D}$, and acquired measurements P', N'—as well as the given probabilistic information pr and some selection heuristic heur (which defines what "best" means, cf. (Rodler, 2018)). The conduction of the measurement at mp (PERFORMMEAS, line 11) 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 knowledge base. 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

[10] The reason why at least two diagnoses should be computed is that *(i)* two or more minimal diagnoses are needed for the computation of a discriminating measurement point (Rodler, 2015) (cf. Sec. 2.5), and *(ii)* this way the stop criterion "only a single minimal diagnosis remaining" (cf. Alg. 1) guarantees that the sequential diagnosis process ends only if diagnostic certainty has been achieved (correct diagnosis among *all* minimal diagnoses found).

[11] Of course, less rigorous stopping criteria are possible, e.g., one might appoint a probability threshold and stop once the probability of one computed diagnosis exceeds this threshold (de Kleer & Williams, 1987).

(ADDMEAS, line 12).[12] 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 (ASSIGNDIAGSOKNOK, line 14).

3.1.4 PROPERTIES

DynamicHS is *stateful* in that one and the same tree data structure is used throughout the entire sequential diagnostic process. Apart from that, it preserves all (desirable) properties of Reiter's HS-Tree, i.e., the *soundness* (only minimal diagnoses for the current DPI are found) and *completeness* (all minimal diagnoses for the current DPI can be found) of the diagnosis search in each iteration, as well as the generality (*logics- and reasoner-independence*) and the feature to *calculate diagnoses in most-preferred-first order*.

3.1.5 SPECIALIZED INCORPORATED TECHNIQUES

To keep the extent—and thus the impact on the computational complexity—of the regularly performed tree update actions at a minimum, DynamicHS incorporates a specialized *lazy update policy*, where only such updates are executed whose omission would (in general) compromise the soundness or completeness of the diagnosis search. Second, as regards the tree pruning, DynamicHS features an *efficient two-tiered redundancy testing technique* for evaluating tree branches with regard to whether they are necessary (still needed) or redundant (ready to be discarded). The principle behind this technique is to first execute a more performant sound but incomplete test, and second, only if necessary, a sound and complete test. Moreover, preliminarily unneeded *"duplicate" tree branches are stored in a space-saving manner* and only used on demand to reconstruct a proper replacement branch if the "original" associated with the duplicate has become redundant. Finally, DynamicHS embraces various *strategies to avoid the overall amount of logical reasoning* involved in its computations, since the calls of a logical inference engine normally account for most of the computation time of a diagnosis system (cf., e.g., (Pill et al., 2011)).

3.2 Description of DynamicHS

In this section we describe the DynamicHS algorithm, given by Alg. 3, in more detail. It inherits many of its aspects from Reiter's HS-Tree, depicted by Alg. 2. Hence, we first recapitulate HS-Tree and then focus on the differences to and idiosyncrasies of DynamicHS.

[12] Depending on the diagnosed system, the notion of a "measurement point" might be quite differently interpreted. For instance, in case of a physical system (de Kleer & Williams, 1987), it might be literally a point mp in the system where to make a measurement with some physical measurement instrument (e.g., a voltmeter). In case of a knowledge-based system (Shchekotykhin et al., 2012), on the other hand, it might be, e.g., a question mp in the form of a logical sentence asked to an expert about the domain described by a problematic knowledge base; e.g., one could ask $mp := \forall X\,(bird(X) \rightarrow canFly(X))$ ("can all birds fly?"). Likewise, the "outcome" of a measurement might be the measured value v at the measurement point mp in the former case, and a truth value answering the asked question mp in the latter. The new "measurement" (logical sentence), formulated from the measurement point and the respective outcome, could be $m_{phys} := mp = v$ in the former, and $m_{KB} := mp$ in the latter case. At this, m_{phys} would be added to the positive measurements P' (since m_{phys} is now a fact), whereas m_{KB} would be added to the positive (negative) measurements P' (N') if the answer to the asked question was positive (negative), i.e., m_{KB} must (not) be true in the correct knowledge base (cf. (Rodler, 2015)).

3.2.1 REITER'S HS-TREE

We briefly repeat the functioning of Reiter's HS-Tree (Alg. 2), which computes minimal diagnoses for a DPI $dpi = \langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle$ in a sound, complete[13] and best-first way.

Starting from a priority queue of unlabeled nodes $\mathbf{Q}$, initially comprising only an unlabeled root node, $\emptyset$, 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 (in Alg. 2), the set of edge labels along the branch from the root node of the tree to a node nd is associated with nd, i.e., nd stores this set of labels. E.g., the node at location ⑫ in iteration 1 of Fig. 4 on page 22 is referred to as $\{1, 2, 5\}$. Once the tree has been completed ($\mathbf{Q} = [\,]$), i.e., all nodes are labeled, the minimal diagnoses for dpi are given exactly by the set of all nodes labeled $valid$; formally: $\mathbf{diag}(dpi) = \{\mathsf{nd} \mid \mathsf{nd} \text{ is labeled } valid\}$.

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 FINDMINCONFLICT, a *(sound and complete) minimal conflicts searcher*, e.g., QuickXPlain (Junker, 2004; Rodler, 2020f), 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 $valid$.

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 (Rodler, 2015, Sec. 4.6)).

Importantly, pr must in any case be specified in a way the probability for each system component (element of $\mathcal{K}$) is below 0.5.[14] In other words, each system component must be more likely to be nominal than to be faulty. This additional assumption is necessary for the soundness of HS-Tree, i.e., to guarantee that *only* minimal diagnoses are returned. The crucial point is that, for a queue $\mathbf{Q}$ ordered by pr, this requirement guarantees that node $\mathsf{n} \in \mathbf{Q}$ will be processed prior to node $\mathsf{n}' \in \mathbf{Q}$ whenever $\mathsf{n} \subset \mathsf{n}'$. For more information consider (Rodler, 2015, p. 73 et seq.).

[13] Unlike Reiter (Reiter, 1987), we assume that only *minimal* conflicts are used as node labels. Thus, the issue pointed out by (Greiner, Smith, & Wilkerson, 1989) does not arise.

[14] This condition (Rodler, 2015) is fairly benign as it can be established from any probability model pr by simply choosing an arbitrary fixed $c \in (0, 0.5)$ and by setting $pr_{adj}(ax) := c \cdot pr(ax)$ for all $ax \in \mathcal{K}$. Observe that this adjustment does not affect the relative probabilities or the fault probability order of components in that $pr_{adj}(ax)/pr_{adj}(ax') = k$ whenever $pr(ax)/pr(ax') = k$.

Algorithm 2 HS-Tree

Input: tuple $\langle dpi, P', N', pr, ld \rangle$ 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

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

```
1:  procedure HSTREE(dpi, P', N', pr, ld)
2:      D_calc, C_calc ← ∅
3:      Q ← [∅]
4:      while Q ≠ [ ] ∧ ( |D_calc| < ld ) do
5:          node ← GETANDDELETEFIRST(Q)
6:          ⟨L, C⟩ ← LABEL(dpi, node, C_calc, D_calc, Q)
7:          C_calc ← C
8:          if L = valid then
9:              D_calc ← D_calc ∪ {node}
10:         else if L = closed then                    ▷ do nothing → node discarded
11:         else                                       ▷ L must be a minimal conflict set
12:             for e ∈ L do
13:                 Q ← INSERTSORTED(node ∪ {e}, Q, pr)
14:     return D_calc

15: procedure LABEL(⟨K, B, P, N⟩, node, C_calc, D_calc, Q)
16:     for nd ∈ D_calc do                             ▷ (L1)
17:         if node ⊇ nd then                          ▷ non-minimality
18:             return ⟨closed, C_calc⟩
19:     for nd ∈ Q do                                  ▷ (L2)
20:         if node = nd then                          ▷ remove duplicates
21:             return ⟨closed, C_calc⟩
22:     for C ∈ C_calc do                              ▷ (L3)
23:         if C ∩ node = ∅ then                       ▷ reuse conflict C
24:             return ⟨C, C_calc⟩
25:     L ← FINDMINCONFLICT(⟨K \ node, B, P, N⟩_R)     ▷ (L4)
26:     if L = 'no conflict' then                      ▷ node is a diagnosis
27:         return ⟨valid, C_calc⟩
28:     else                                           ▷ L is new minimal conflict set (∉ C_calc)
29:         C_calc ← C_calc ∪ {L}
30:         return ⟨L, C_calc⟩
```

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, an HS-Tree is built from scratch in each iteration, every time for different measurement sets P', N'.

3.2.2 DYNAMICITY OF DPI IN SEQUENTIAL DIAGNOSIS

In the course of the sequential diagnosis process (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 discriminating measurement[15] 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.

[15] A *discriminating measurement* is a measurement at a discriminating measurement point (cf. Sec. 2). That is, adding such a measurement to (the positive or negative measurements of) the DPI causes the invalidation of at least one diagnosis.

3.2.3 TOWARDS STATEFUL HITTING SET COMPUTATION

To understand the necessary design decisions to devise a sound and complete stateful hitting set algorithm in the light of the said DPI dynamicity, we next discuss a few more specifics of the conflicts and diagnoses evolution throughout sequential diagnosis:[16]

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 leading diagnoses $\mathbf{D}$ where $|\mathbf{D}| \geq 2$. Further, let the measurement m describe the outcome of measuring at a discriminating measurement point computed from $\mathbf{D}$, and let dpi_{j+1} be the DPI resulting from dpi_j through the addition of 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 appropriately updated.

2. *Each $\mathcal{D} \in \mathbf{diag}(dpi_{j+1})$ is either equal to or a superset of some $\mathcal{D}' \in \mathbf{diag}(dpi_j)$.*
   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 some $\mathcal{C}' \in \mathbf{conf}(dpi_{j+1})$ 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, at least one minimal conflict is reduced in size, and/or at least one entirely new minimal conflict (which is 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 (has an empty intersection with) the conflict $\mathcal{C}_{new}$.

3.2.4 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. We exemplify these modifications by referring to Figs. 3 and 4 (on pages 21 and 22), which are discussed in detail in Example 5 and which describe sequential diagnosis executions (as per Alg. 1) of DynamicHS and HS-Tree, respectively, for our example DPI given in Tab. 1.

(Mod1) A tree update is executed at the beginning of each DynamicHS execution, where the hitting set tree produced for a previously relevant DPI is adapted to a tree that allows to compute minimal diagnoses for the current DPI in a sound, complete and best-first manner.

[16] A more formal treatment and proofs for all statements can be found in (Rodler, 2015, Sec. 12.4).

Justification: The necessity of this update is attested by Property 1.1. The actions taken in the course of the update are motivated by the remaining bullet points of Property 1.

Example: Between iterations 1 and 2 in Fig. 3, DynamicHS runs a tree update, which adapts the tree constructed in iteration 1 for dpi_0 to one that can be reused to tackle dpi_1 in iteration 2. This update effectively involves the removal of the redundant nodes (minimal diagnoses for dpi_0, indicated by a ✓) numbered ⑥ and ⑧, the reduction of the conflicts numbered ② and ③ to their proper subsets $\langle 2, 4\rangle$ and $\langle 1, 5\rangle$ which are now (for dpi_1) minimal conflicts, and the reinsertion of node ⑩ into the queue $\mathbf{Q}$ (because, after the deletion of the two mentioned redundant nodes, there is no longer a "witness" diagnosis testifying the non-minimality of node ⑩).

(Mod2) 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 non-minimal diagnoses, as they might constitute (subbranches of) minimal diagnoses in the next iteration. 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.

Example: Node ④ in iteration 1 of Fig. 3 is added to $\mathbf{Q}_{dup}$ (DynamicHS), whereas the corresponding node ⑦ in iteration 1 of Fig. 4 is not further considered (HS-Tree). Similarly, node ⑩ is stored in $\mathbf{D}_{\supset}$ in case of DynamicHS (Fig. 3), whereas the same node is simply closed and eliminated in HS-Tree (Fig. 4).

(Mod3) Each node nd is no longer identified as the *set* of edge labels from the root to nd, but as an *ordered list* of these edge labels. In addition, an ordered list of the conflicts used to label internal nodes along the branch from the root to nd is stored in terms of nd.cs.

Justification: This modification is required due to Property 1.4. The necessity of storing both the edge labels and the internal node labels as lists has its reason 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.

Example: For the node nd at location ⑨ in iteration 1 of Fig. 3, we have $\mathsf{nd} = [2, 5]$ and $\mathsf{nd.cs} = [\langle 1, 2\rangle, \langle 1, 3, 5\rangle]$.

(Mod4) Before reusing a conflict $\mathcal{C}$ to label a node (labeling test (L3) in HS-Tree), a minimality check for $\mathcal{C}$ is performed. If a proper subset X of $\mathcal{C}$ is identified as a conflict for the current DPI in the course of this check, X is used to prune obsolete tree branches, to replace node-labeling conflicts that are supersets of X by X, and to update $\mathbf{C}_{calc}$ in that X is added and all of its supersets are deleted.

Justification: By Property 1.3, stored conflicts in $\mathbf{C}_{calc}$ and those appearing as labels in the existing tree (elements of the lists nd.cs for the different nodes nd) might not be minimal for the current DPI (since they might have been computed for a prior DPI). This minimality check helps both to prune the tree (reduction of the number of nodes) and to make sure that any extension to the tree uses

only minimal conflicts wrt. the current DPI as internal node labels (avoidance of the introduction of unnecessary new edges).

Example: Let us assume that a new node should be labeled in iteration 2 of DynamicHS (Fig. 3) and $\mathcal{C} := \langle 3, 4, 5 \rangle$ is a conflict that satisfies the reuse-test (L3). Then the performed minimality check would return $X := \langle 4, 5 \rangle$ as a subset of $\mathcal{C}$ (since $\mathcal{C}$ is a minimal conflict for dpi_0, but a non-minimal one for dpi_1, cf. Tab. 2).

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

Justification: Statefulness of DynamicHS.

Example: Consider iteration 2 for both HS-Tree (Fig. 4) and DynamicHS (Fig. 3). In the latter case, for the new DPI dpi_1, the tree built in iteration 1 is reused, whereas in the former a new one is built. One implication of this is, e.g., that DynamicHS does not need to (fully) recompute the conflicts $\langle 1, 2 \rangle$, $\langle 2, 4 \rangle$ and $\langle 1, 5 \rangle$.

3.2.5 DYNAMICHS: ALGORITHM WALKTHROUGH

We now explicate the workings of DynamicHS, depicted by Alg. 3.

Like HS-Tree, DynamicHS is basically processing a priority queue $\mathbf{Q}$ of nodes (while-loop; lines 4 and 5). That is, in each iteration of the while-loop, 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 already computed minimal diagnoses $\mathbf{D}_{\checkmark}$ from the previous iteration which are consistent with all measurements in $P' \cup N'$ (line 6). If so, the node is directly labeled $valid$ (line 7). Otherwise the DLABEL function is invoked to compute a label for node (line 9).

DLABEL: First, the non-minimality check is performed, just as done in HS-Tree, see (L1) above (lines 24–26). If the check is negative, a conflict-reuse check is carried out next (lines 27–34). Note that the duplicate check (L2), which is done at this stage in HS-Tree, is obsolete since no duplicate nodes can ever be elements of $\mathbf{Q}$. This is evaded by directly making the duplicate check after new nodes have been constructed and are to be added to $\mathbf{Q}$ (see lines 18 and 19). The conflict-reuse check starts equally as in HS-Tree. However, if a suitable conflict $\mathcal{C}$ for reuse is found in the set of so-far computed conflicts $\mathbf{C}_{calc}$ (line 28), then FINDMINCONFLICT (cf. (L4) above) is employed to check the minimality of the conflict wrt. the *current* DPI (line 29). If a proper subset X of $\mathcal{C}$ is found which is a conflict wrt. the current DPI (line 32), then X is used to prune the current hitting set tree (line 33) using the PRUNE function (see below). Finally, depending on the minimality check, either the conflict $\mathcal{C}$ from $\mathbf{C}_{calc}$, if minimal, or the computed subset X of $\mathcal{C}$ is used to label node (lines 31 and 34). In case no conflict is eligible for being reused, FINDMINCONFLICT is called (line 35) to test if a minimal conflict (wrt. the current DPI) to label node exists at all, or if node already corresponds to a (minimal) diagnosis wrt. the current DPI (again, this part, i.e., lines 35–40, is equal to HS-Tree, cf. (L4) above). As a last point, note that DLABEL gets and returns the tuple state as an argument. This tuple includes collections of nodes storing the current state of the hitting set tree maintained by DynamicHS. The reason for passing state to DLABEL is that the pruning actions potentially performed in the course of the reuse-check might modify state (while all other actions executed during DLABEL can never alter state).

Algorithm 3 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. $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle$
- $\mathsf{state} = \langle \mathbf{Q}, \mathbf{Q}_{dup}, \mathbf{D}_{\supset}, \mathbf{C}_{calc} \rangle$ 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

```
1:  procedure DYNAMICHS(dpi, P', N', pr, ld, D_✓, D_×, state)
2:      D_calc ← ∅
3:      state ← UPDATETREE(dpi, P', N', pr, D_✓, D_×, state)
4:      while Q ≠ [] ∧ ( |D_calc| < ld ) do
5:          node ← GETANDDELETEFIRST(Q)                          ▷ node is processed
6:          if node ∈ D_✓ then                                   ▷ D_✓ includes only min...
7:              L ← valid                                        ▷ ...diags wrt. current DPI
8:          else
9:              ⟨L, state⟩ ← DLABEL(dpi, P', N', pr, node, D_calc, state)
10:         if L = valid then
11:             D_calc ← D_calc ∪ {node}                         ▷ node is a min diag wrt. current DPI
12:         else if L = nonmin then
13:             D_⊃ ← D_⊃ ∪ {node}                               ▷ node is a non-min diag wrt. current DPI
14:         else
15:             for e ∈ L do                                     ▷ L is a min conflict wrt. current DPI
16:                 node_e ← APPEND(node, e)                     ▷ node_e is generated
17:                 node_e.cs ← APPEND(node.cs, L)
18:                 if node_e ∈ Q ∨ node_e ∈ D_⊃ then            ▷ node_e is (set-equal) duplicate of some node in Q or D_⊃
19:                     Q_dup ← INSERTSORTED(node_e, Q_dup, card, <)
20:                 else
21:                     Q ← INSERTSORTED(node_e, Q, pr, >)
22:     return ⟨D_calc, state⟩

23: procedure DLABEL(⟨K, B, P, N⟩, P', N', pr, node, D_calc, state)
24:     for nd ∈ D_calc do
25:         if node ⊃ nd then                                    ▷ node is a non-min diag
26:             return ⟨nonmin, state⟩
27:     for C ∈ C_calc do                                        ▷ C_calc includes conflicts wrt. current DPI
28:         if C ∩ node = ∅ then                                 ▷ reuse (a subset of) C to label node
29:             X ← FINDMINCONFLICT(⟨C, B, P ∪ P', N ∪ N'⟩)
30:             if X = C then
31:                 return ⟨C, state⟩
32:             else                                             ▷ X ⊂ C
33:                 ⟨state, D_calc⟩ ← PRUNE(X, {state, D_calc})
34:                 return ⟨X, state⟩
35:     L ← FINDMINCONFLICT(⟨K \ node, B, P ∪ P', N ∪ N'⟩)
36:     if L = 'no conflict' then                                ▷ node is a diag
37:         return ⟨valid, state⟩
38:     else                                                     ▷ L is a new min conflict (∉ C_calc)
39:         C_calc ← C_calc ∪ {L}
40:         return ⟨L, state⟩
```

Algorithm 3 DynamicHS (continued)

```
41: procedure UPDATETREE(dpi, P', N', pr, D_✓, D_×, state)
42:     for nd ∈ D_× do                                   ▷ search for redundant nodes among invalidated diags
43:         if REDUNDANT(nd, dpi) then
44:             ⟨state, D_✓, D_×⟩ ← PRUNE(X, {state, D_✓, D_×})
45:     for nd ∈ D_× do                                   ▷ add all (non-pruned) nodes in D_× to Q
46:         Q ← INSERTSORTED(nd, Q, pr, >)
47:         D_× ← D_× \ {nd}
48:     for nd ∈ D_⊃ do                                   ▷ add all (non-pruned) nodes in D_⊃ to Q, which...
49:         nonmin ← false                                ▷ ...are no longer supersets of any diag in D_✓
50:         for nd' ∈ D_✓ do
51:             if nd ⊃ nd' then
52:                 nonmin ← true
53:                 break
54:         if nonmin = false then
55:             Q ← INSERTSORTED(nd, Q, pr, >)
56:             D_⊃ ← D_⊃ \ {nd}
57:     for D ∈ D_✓ do                                    ▷ add known min diags in D_✓ to Q to find diags...
58:         Q ← INSERTSORTED(D, Q, pr, >)                 ▷ ...in best-first order (as per pr)
59:     return state
```

The output of DLABEL is then processed by DynamicHS (lines 10–22) in order to assign node to its appropriate node collection (note that the labels of processed nodes are implicitly stored by assigning the nodes to corresponding collections). More 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, on the other hand, the label is a minimal conflict L, then a child node 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 $\mathbf{Q}$ or in $\mathbf{D}_{\supset}$ that is *set*-equal to node_e, or to $\mathbf{Q}$ otherwise (line 21). At this, 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. **(Mod3)** above). Note that both $\mathbf{Q}$ and $\mathbf{Q}_{dup}$ are priority queues. $\mathbf{Q}$ is ordered by descending node probability as per the given measure pr, and $\mathbf{Q}_{dup}$ by ascending node cardinality, i.e., nodes with fewer edge labels along their branch from the root node are prioritized. The reason for this will become fully evident when we discuss the PRUNE function (see below).

The described processing of nodes from $\mathbf{Q}$ is successively continued until one of the stop criteria applies (while-loop, line 4). That is, the algorithm returns the set of computed minimal diagnoses $\mathbf{D}_{calc}$ along with the current tree state state when either all nodes have been processed ($\mathbf{Q} = [\,]$), i.e., the tree has been built to its entirety, or when the stipulated number ld of leading diagnoses have been found ($|\mathbf{D}_{calc}| = ld$).

UPDATETREE: The idea here is to adapt the current tree in a way it constitutes a basis for finding all and only minimal diagnoses in highest-probability-first order for the *current* DPI. As discussed above, the minimal diagnoses can only remain the same or be augmented in size (Property 1.2), whereas for existing conflicts it is the case that they can only remain the same or shrink in size (Property 1.3), or new conflicts can arise (Property 1.4). The principle followed by UPDATETREE is to search for non-minimal conflicts to be updated, and tree branches to be pruned, among the nodes that corresponded to minimal diagnoses for the previous DPI, but have been invalidated by the latest added measurement (the elements of $\mathbf{D}_{\times}$, cf. line 14 in Alg. 1).

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 $\mathsf{nd}[j] \in \mathsf{nd.cs}[j] \setminus X$.[17] Moreover, we call X a *witness of redundancy for* nd (wrt. dpi). In simple words, nd is redundant iff the branch from the root node to nd would not exist given that the (current) minimal conflict X had been used instead of the (old, formerly minimal, but by now) non-minimal conflict $\mathsf{nd.cs}[j]$.

If such a redundant node is detected among the elements of $\mathbf{D}_\times$ (function REDUNDANT, drawing on call(s) to FINDMINCONFLICT; explained in detail later), then the PRUNE function (see later) 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,[18] 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 according to pr (lines 45–47), as accomplished by the function INSERTSORTED. Likewise, all non-pruned nodes in $\mathbf{D}_\supset$ (note that pruning always considers all 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, the known minimal diagnoses that have been returned by the previous execution of DynamicHS and are consistent with the latest added measurement (the elements of $\mathbf{D}_\checkmark$), are put back to the ordered queue $\mathbf{Q}$. The justification for this step is given by the fact that, for the current DPI, there might be "new" minimal diagnoses that are more probable than the ones known from the previous iteration. Omitting this step therefore would (generally) compromise the best-first property of the hitting set computation.

PRUNE: Using its first argument X, the function runs through the node collections $\mathbf{Q}_{dup}$, $\mathbf{Q}$, $\mathbf{D}_\supset$ and $\mathbf{D}_{calc}$ when called in line 33, and through $\mathbf{Q}_{dup}$, $\mathbf{Q}$, $\mathbf{D}_\supset$, $\mathbf{D}_\times$ and $\mathbf{D}_\checkmark$ when called in line 44 (cf. second argument), and

- *(relabeling of old conflicts)* replaces all labels $\mathsf{nd.cs}[i]$ which are proper supersets of X by X for all nodes nd and for all $i = 1, \ldots, |\mathsf{nd}|$, 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'.

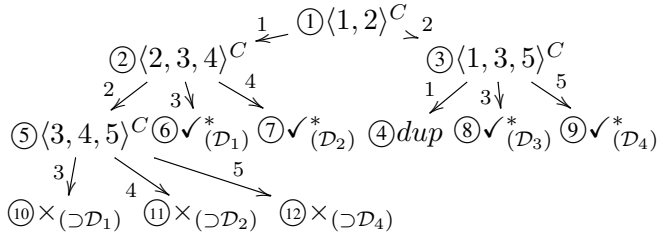
A node nd' qualifies as a *replacement node for* nd iff nd' is non-redundant and has the same *set* of edge labels along its branch from the root node as nd, i.e., iff nd is set-equal (not equal in terms of the list of edge labels) to nd'. This node replacement is necessary from the point of view of completeness (cf. (Greiner et al., 1989)). Importantly, $\mathbf{Q}_{dup}$ must be pruned before the other collections of nodes ($\mathbf{Q}$, $\mathbf{D}_\supset$, $\mathbf{D}_\times$, $\mathbf{D}_\checkmark$, $\mathbf{D}_{calc}$) are pruned, to guarantee that all nodes in $\mathbf{Q}_{dup}$ represent possible *non-redundant* replacement nodes when it comes to pruning nodes from these collections.

Additionally, the argument X is used to update the conflicts stored for reuse (set $\mathbf{C}_{calc}$). The principle is to run through $\mathbf{C}_{calc}$ once, while deleting all proper supersets of X, and to finally add X to $\mathbf{C}_{calc}$.

Example 5 Consider our example DPI dpi_0 given in Tab. 1. The goal is the localization of the faulty axioms, i.e., of those elements of $\mathcal{K}$ that prevent the satisfaction of all given measurements

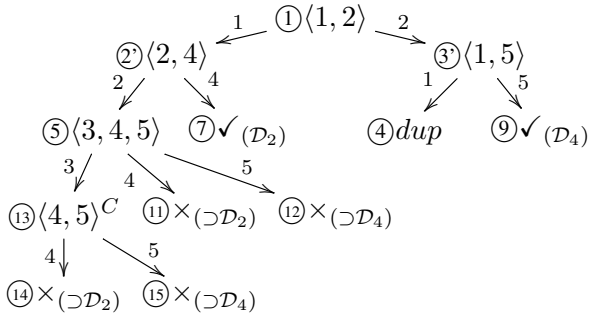
[17] By $\mathsf{nd}[j]$ and $\mathsf{nd.cs}[j]$, respectively, we refer to the j-th element of the ordered lists nd and $\mathsf{nd.cs}$ (cf. **(Mod3)** above), i.e., to the j-th edge label and to the j-th conflict along node nd's branch starting from the root.

[18] Note that $\mathbf{D}_\times$ itself might change during the execution of the tree pruning (line 44) in that redundant nodes might be deleted from it and corresponding replacement nodes (constructed) from the stored duplicates might be added to it. Thus, the semantics of the for-loop in line 42 is that it stops iff each node in $\mathbf{D}_\times$ *at the present time* has been processed.



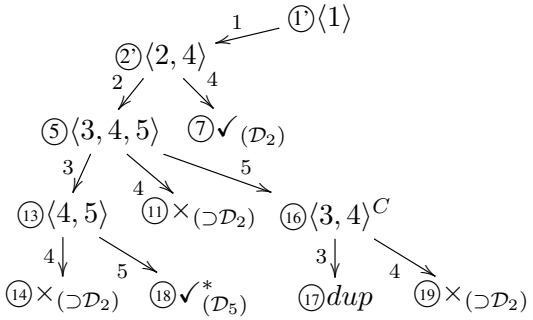
iteration 1 (dpi_0)

$\Rightarrow \mathbf{D} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}$ $\Rightarrow m_1 : A \rightarrow C$ to N' $\Rightarrow \mathbf{D}_\checkmark = \{\mathcal{D}_2, \mathcal{D}_4\}, \mathbf{D}_\times = \{\mathcal{D}_1, \mathcal{D}_3\}$



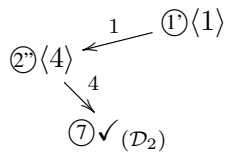
iteration 2 (dpi_1)

$\Rightarrow \mathbf{D} = \{\mathcal{D}_2, \mathcal{D}_4\}$ $\Rightarrow m_2 : A \rightarrow \neg B$ to N' $\Rightarrow \mathbf{D}_\checkmark = \{\mathcal{D}_2\}, \mathbf{D}_\times = \{\mathcal{D}_4\}$



iteration 3 (dpi_2)

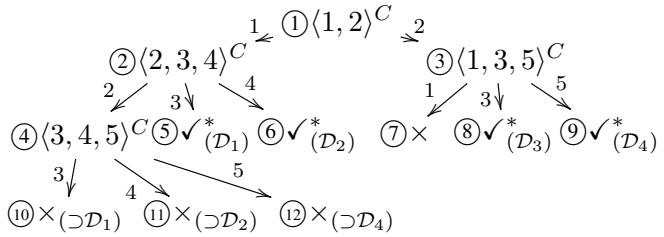
$\Rightarrow \mathbf{D} = \{\mathcal{D}_2, \mathcal{D}_5\}$ $\Rightarrow m_3 : A \rightarrow \neg C$ to P' $\Rightarrow \mathbf{D}_\checkmark = \{\mathcal{D}_2\}, \mathbf{D}_\times = \{\mathcal{D}_5\}$



iteration 4 (dpi_3)

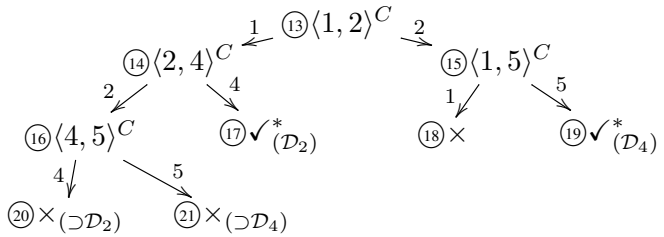
$\Rightarrow \mathbf{D} = \{\mathcal{D}_2\}$

Figure 3: DynamicHS executed on example DPI given in Tab. 1.



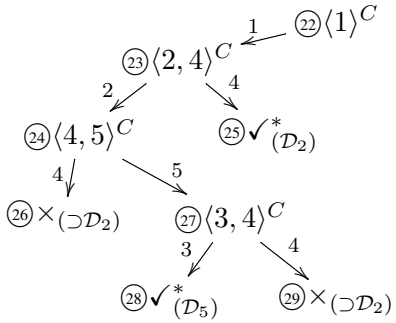
iteration 1 (dpi_0)

$\Rightarrow \mathbf{D} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\}$ $\Rightarrow m_1: \ A \rightarrow C$ to N'



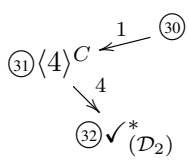
iteration 2 (dpi_1)

$\Rightarrow \mathbf{D} = \{\mathcal{D}_2, \mathcal{D}_4\}$ $\Rightarrow m_2: \ A \rightarrow \neg B$ to N'



iteration 3 (dpi_2)

$\Rightarrow \mathbf{D} = \{\mathcal{D}_2, \mathcal{D}_5\}$ $\Rightarrow m_3: \ A \rightarrow \neg C$ to P'



iteration 4 (dpi_3)

$\Rightarrow \mathbf{D} = \{\mathcal{D}_2\}$

Figure 4: HS-Tree executed on example DPI given in Tab. 1.

(in this case, there is only one negative measurement, i.e., $\neg A$ must not be entailed by the correct knowledge base). We now illustrate the workings of both DynamicHS (Fig. 3) and HS-Tree (Fig. 4) in that we play through a complete sequential diagnosis session on this particular example, under the assumption that $\mathcal{D}^* = [ax_1, ax_4]$ is the actual diagnosis.

Inputs (Sequential Diagnosis): Concerning the inputs to Alg. 1, we assume that $ld := 5$ (i.e., in each iteration, if existent, five leading diagnoses are computed), heur is a heuristic that prefers measurements the more, the more leading diagnoses they eliminate in the worst case (cf. the split-in-half heuristic in (Shchekotykhin et al., 2012)), and pr is chosen in a way all elements of $\mathcal{K}$ have an equal fault probability which leads to a breadth-first construction of the hitting set tree(s).

Inputs (HS-Tree and DynamicHS): The inputs to Alg. 1 are passed to DynamicHS in line 6. In addition to these, DynamicHS (see Alg. 3) accepts the arguments P', N' (current sets of positive and negative measurements gathered throughout the execution of Alg. 1 so far; both initially empty), $\mathbf{D}_\checkmark$ and $\mathbf{D}_\times$ (the minimal diagnoses found in the previous iteration which are consistent and inconsistent, respectively, with the last added measurement; both initially empty), and state (stores the state of the current DynamicHS-Tree within and between all its iterations) which consists of the collections $\mathbf{Q}$ (list of open nodes; initially the list containing an empty node $[\,]$), $\mathbf{Q}_{dup}$ (list of duplicate nodes; initially empty), $\mathbf{D}_\supset$ (set of non-minimal diagnoses; initially empty), as well as $\mathbf{C}_{calc}$ (set of already computed conflicts; initially empty). These variable initializations take place in lines 1–3 of Alg. 1. Unlike DynamicHS, HS-Tree takes only the arguments P' and N' beyond the general sequential diagnosis inputs ld, pr and heur. That is, the queue $\mathbf{Q}$, the computed diagnoses $\mathbf{D}_{calc}$, and the computed conflicts $\mathbf{C}_{calc}$ are instance variables in case of HS-Tree due to its statelessness.

Notation in Figures: Axioms ax_i in $\mathcal{K}$ are referred to by i (in node and edge labels) for simplicity. Circled numbers indicate the chronological order in which nodes are labeled, i.e., node ⓘ is labeled at point in time i (starting from 1). We use $\langle \dots \rangle$ to denote the conflicts that label the internal hitting set tree nodes, and tag these conflicts by ^{C} if they are freshly computed by a FINDMINCONFLICT call (line 35 in DLABEL), and leave them without a tag if they are the product of a redundancy check and a subsequent relabeling (lines 43–44 in UPDATETREE).[19,20] For the leaf nodes, we use the following labels: $\checkmark_{(\mathcal{D}_i)}$ to denote a minimal diagnosis, named $\mathcal{D}_i$, stored in $\mathbf{D}_{calc}$, $\times$ to denote a duplicate in HS-Tree (see (L2) criterion above), $\times_{(\supset \mathcal{D}_i)}$ to denote a non-minimal diagnosis (stored in $\mathbf{D}_\supset$ by DynamicHS), where $\mathcal{D}_i$ is some minimal diagnosis that proves the non-minimality, and dup to denote 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 in DynamicHS, and line 25 in HS-Tree) is necessary to prove it is a diagnosis, and untagged otherwise (i.e., the branch is known to be a diagnosis from a previous iteration, and stored in $\mathbf{D}_\checkmark$; only pertinent to DynamicHS). Below each graph showing the respective hitting set tree, the figures show the current iteration i of Alg. 1, the leading diagnoses $\mathbf{D}$ output (Alg. 1, line 6 for DynamicHS and line 8 for HS-Tree, respectively), the measurement m_i added to one of the measurement sets (P' or N') of the the DPI (Alg. 1, lines 10–12), as well as the sets $\mathbf{D}_\checkmark$ and $\mathbf{D}_\times$ (Alg. 1, line 14) in case of DynamicHS.

[19] The rationale behind this distinction between the individual conflicts used as node labels is that their computation time tends to differ significantly (as we will discuss in more detail in Sec. 5.4.2). For that reason, we will later also denote the operations used to compute the conflicts tagged by ^{C} as "hard", and those that yield conflicts without any tag as "easy" (see Sec. 3.3.3).

[20] Note that there are no reused conflicts (found in lines 27–34 in DLABEL) in this particular example. Therefore, we do not introduce a distinct conflict tag for this case.

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 3 and 4, respectively. Newly arisen minimal conflicts (which are neither subsets nor supersets of prior minimal conflicts) are underlined.

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	–	$A \rightarrow C$	$[1,4],[2,5]$	$\langle 1,2\rangle, \langle 2,4\rangle, \langle 1,5\rangle, \langle 4,5\rangle$
3	–	$A \rightarrow C,\ A \rightarrow \neg B$	$[1,4],[1,2,3,5]$	$\langle 1\rangle, \langle 2,4\rangle, \underline{\langle 3,4\rangle}, \langle 4,5\rangle$
4	$A \rightarrow \neg C$	$A \rightarrow C,\ A \rightarrow \neg B$	$[1,4]$	$\langle 1\rangle, \underline{\langle 4\rangle}$

Iteration 1: In their first iteration, HS-Tree and DynamicHS essentially do the same, i.e., they build the same tree (compare Figs. 3 and 4). The only difference is that DynamicHS stores the duplicate branches (labeled by dup) and the ones corresponding to non-minimal diagnoses (labeled by $\times_{(\supset \mathcal{D}_i)}$), whereas HS-Tree simply discards these branches (those labeled by $\times$ or $\times_{(\supset \mathcal{D}_i)}$). Note that duplicates are stored by DynamicHS at *generation* time (line 19), hence the duplicate (dup) found has number ④ (and not ⑦, as the respective branch for HS-Tree). The leading diagnoses computed by both algorithms are $\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4\} = \{[1,3],[1,4],[2,3],[2,5]\}$. The soundness and completeness of this result can also be read from Tab. 2 which illustrates the evolution of the minimal diagnoses ($\mathbf{diag}$) and minimal conflicts ($\mathbf{conf}$) for the varying DPI (successively extended P' and N' sets) in this example. We emphasize that the leading diagnoses returned by both algorithms *must* be equal in any iteration (given the same parameters ld and pr) since both methods are sound, complete and best-first minimal diagnosis computers. As a consequence, when using the same measurement computation technique (and heuristic heur), both algorithms also *must* give rise to the same proposed next measurement m_i in each iteration.

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

Tree Update: From the second iteration (where dpi_1 is considered) on, the behaviors of DynamicHS and HS-Tree start to differ. Whereas HS-Tree simply constructs a new hitting set tree from scratch for dpi_1, DynamicHS runs a tree update (function 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 (the new minimal conflicts $\langle 2,4\rangle$ and $\langle 1,5\rangle$) are found while analyzing the (conflicts along the) branches of the two invalidated diagnoses $[1,3]$ and $[2,3]$ (⑥ and ⑧). For instance, $\mathsf{nd} = [1,3]$ is redundant since the (*former*, for dpi_0, *minimal*) 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 there are no appropriate duplicates that allow the construction of a replacement node for any of the two redundant branches $[1,3]$ and $[2,3]$, both of these branches are removed from the tree. Further, the by now non-minimal conflicts at ② and ③ are replaced, each by the respective witness of redundancy that is a subset of it (e.g., $\langle 2,3,4\rangle$ is replaced by $\langle 2,4\rangle$). This relabeling is signified by the prime (') in the new node numbers ②' and ③', respectively.

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 the queue of unlabeled nodes $\mathbf{Q}$ 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 indeed no longer a diagnosis, i.e., 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 (compare Figs. 3 and 4).

First, while HS-Tree has to compute all the conflicts labeling internal nodes by potentially expensive FINDMINCONFLICT calls (see the ^{C} tags), DynamicHS performs a (generally) cheaper reduction of the existing conflicts in the course of the pruning actions we discussed above. However, note that not all conflicts are necessarily always kept up-to-date after a DPI-transition. This is part of the *lazy updating policy* pursued by DynamicHS, detailed in Sec. 3.3.2. For instance, node ⑤ is still labeled by the now non-minimal conflict $\langle 3, 4, 5\rangle$ at the time UPDATETREE terminates. Hence, the subtree comprising nodes ⑬, ⑭ and ⑮ is not present in case HS-Tree is used. Importantly, this lazy updating strategy has no negative impact on the soundness or completeness of DynamicHS (see Sec. 3.3.2 and Theorem 1).

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 due to the storage and reuse of $\mathbf{D}_\checkmark$ (the minimal diagnoses returned by the previous iteration which are consistent with the lastly added measurement).

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 $\mathbf{Q}$ 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 $[1, 4]$.

Performance Comparison: As Figs. 3 and 4 show, DynamicHS generates 19 nodes and requires 6 conflict computations, as opposed to 32 nodes and 14 computations in case of HS-Tree. These reductions in terms of tree rebuilding and conflict computation costs represent two main factors responsible for runtime improvements of DynamicHS over HS-Tree. □

3.3 Advanced Techniques in DynamicHS

DynamicHS embraces several sophisticated techniques specialized in improving its (time or space) performance, which we discuss next.

3.3.1 EFFICIENT REDUNDANCY CHECKING

We now detail the workings of the function REDUNDANT (called in line 43 of Alg. 3):

The definition of node redundancy given in Sec. 3.2.5 directly suggests a method for checking whether or not a node is redundant, which we call *complete redundancy check (CRC)*. It runs through all conflicts $\mathsf{nd.cs}[i]$ used as labels along the branch to node nd (i.e., $i \in \{1, \ldots, |\mathsf{nd.cs}|\}$) and calls FINDMINCONFLICT with arguments $(\langle \mathsf{nd.cs}[i] \setminus \{\mathsf{nd}[i]\}, \mathcal{B}, P \cup P', N \cup N' \rangle)$ to test if there is a witness of redundancy (see Sec. 3.2.5) for nd. A witness of redundancy exists for nd iff, for some i, this call to FINDMINCONFLICT returns a conflict X. Because, this conflict then must be a subset of $\mathsf{nd.cs}[i] \setminus \{\mathsf{nd}[i]\}$, meaning that nd is redundant. Hence, if such an X is found, then CRC is successful (returns $true$) and the witness X is used as an argument passed to the subsequent call of the PRUNE method (Alg. 3, line 44). Otherwise, i.e., if all calls of FINDMINCONFLICT made by the CRC return 'no conflict', it is proven that the node is not redundant and CRC returns $false$.

The CRC enables sound (if CRC true, then node redundant) and complete (if node redundant, then CRC true) redundancy checking. However, a drawback of the CRC is that it requires $|\mathsf{nd}|$ calls to the (expensive) method FINDMINCONFLICT in the worst case, where $|\mathsf{nd}|$ is in $O(|\mathbf{conf}(dpi)|)$ since a node cannot hit any more than each minimal conflict for the current DPI dpi. As a remedy to that, we devised a more efficient, sound but incomplete, so-called *quick redundancy check (QRC)*, which is executed previous to the CRC and requires only a single call of FINDMINCONFLICT. The concept is that a positive QRC makes the more expensive CRC obsolete; and, in case of a negative outcome, CRC must be executed, but the overhead amounts to only a single FINDMINCONFLICT call.

To check the redundancy of nd, QRC executes FINDMINCONFLICT with arguments $(\langle U_{\mathsf{nd.cs}} \setminus \mathsf{nd}, \mathcal{B}, P \cup P', N \cup N' \rangle)$.[21] If 'no conflict' is returned, the QRC terminates negatively, which prompts the execution of the CRC. Otherwise, if a conflict X is returned, QRC checks whether X is a proper subset of some conflict in $\mathsf{nd.cs}$, i.e., whether $X \subset \mathcal{C}$ for $\mathcal{C} = \mathsf{nd.cs}[k]$ for some k. In case of a positive subset-check, the QRC returns positively and it follows that nd is redundant, regardless of the particular k. The reason is that the argument $U_{\mathsf{nd.cs}} \setminus \mathsf{nd}$ passed to FINDMINCONFLICT does not include any element of nd, and hence the output conflict cannot include such elements either. Thus, if $X \subset \mathsf{nd.cs}[k]$ holds, then $X \subset \mathsf{nd.cs}[k] \setminus \mathsf{nd}$, and therefore $X \subset \mathsf{nd.cs}[k] \setminus \{\mathsf{nd}[i]\}$ for all i, and in particular for $i = k$. So, $X \subset \mathsf{nd.cs}[k] \setminus \{\mathsf{nd}[k]\}$, which is equivalent to the definition of redundancy (see page 19).

To see why the QRC is incomplete, i.e., that nd *can* be redundant even if the outcome of the QRC is negative, consider the following example:

Example 6 Let $\mathsf{nd} = [1, 2]$ and $\mathsf{nd.cs} = [\langle 1, 2 \rangle, \langle 2, 3 \rangle]$. Assume that $X := \langle 2 \rangle$ is a new minimal conflict and that $\{3\}$ is not a conflict. Clearly, this implies that nd is redundant because $\mathsf{nd.cs}[1] \setminus X = \{1\}$ and $\mathsf{nd}[1] = 1$. However, $U_{\mathsf{nd.cs}} \setminus \mathsf{nd} = \{3\}$, which is not a conflict, which is why FINDMINCONFLICT given the DPI $(\langle U_{\mathsf{nd.cs}} \setminus \mathsf{nd}, \mathcal{B}, P \cup P', N \cup N' \rangle)$ as argument returns 'no conflict'. Hence, the QRC returns negatively although nd is in fact redundant. □

The crucial aspect which makes this incompleteness possible is the potential overlapping of conflicts. Exactly this overlapping effectuates in the above example that more than one element (actually even all elements) of the outdated non-minimal conflict $\langle 1, 2 \rangle$ are eliminated from $U_{\mathsf{nd.cs}} = \{1, 2, 3\}$ by deleting $\mathsf{nd} = [1, 2]$. As a consequence, the new reduced conflict $\langle 2 \rangle$ is not contained any longer in the set tested by FINDMINCONFLICT.

[21] For a collection of sets Z, we denote by U_Z the union of all sets in Z.

In fact, we can conclude that the QRC is sound *and complete* in the special cases where all minimal conflicts are pairwise disjoint or, more generally, where nd does not include any element that occurs in multiple conflicts in nd.cs.

3.3.2 LAZY UPDATING POLICY

Updating DynamicHS's hitting set tree after the detection of some witness of redundancy X involves going through all nodes of the tree and checking their redundancy wrt. X (cf. Sec. 3.2.5). To avoid these costs as much as possible, DynamicHS aims at minimizing the number of performed updates under preservation of its correctness. This can be seen from line 42, where only the set $\mathbf{D}_{\times}$ including the most "suspicious" nodes (i.e., the diagnoses invalidated by the latest added measurement) is checked for redundancy. In general, this means that we allow some differences between the tree used by DynamicHS to compute diagnoses and the one that would be obtained when building Reiter's HS-Tree for the current DPI from scratch. More specifically, we allow the presence of non-minimal conflicts that are used as node labels as well as—conditioned by these non-minimal conflicts—the presence of unnecessary nodes (while, of course, seeking to minimize the number of such occurrences; cf. PRUNE function). Still, every time a conflict is used to label a (newly processed) node, the algorithm guarantees that it is a minimal conflict for the current DPI (cf. the conflict-minimality test in the course of the conflict reuse check in DLABEL, lines 29–34).

This *lazy updating policy* takes effect, e.g., in iteration 2 of the example execution of DynamicHS shown in Fig. 3 (see Example 5). Here the, at this point, already non-minimal conflict $\mathcal{C}_{\neg min} := \langle 3, 4, 5 \rangle$ still appears as a node label, while $\mathcal{C}_{min} := \langle 4, 5 \rangle$ is now a minimal conflict for the current DPI.

Notwithstanding the correctness proof of DynamicHS given in Sec. 3.4, we next argue briefly why the presence of such non-minimal conflicts $\mathcal{C}_{\neg min}$ does neither counteract the soundness nor the completeness of DynamicHS. To this end, we first point out that (*) only nodes can be labeled $valid$ by DynamicHS which are diagnoses for the current DPI (see line 35 in DLABEL; and line 6 along with the definition of $\mathbf{D}_{\checkmark}$):

- Assume the *completeness* is compromised. Then the processing of some minimal diagnosis must be prevented from reaching line 35 in DLABEL (note that it is unavoidable that processed nodes are recognized as diagnoses in line 6). Since the presence of all still relevant (i.e., non-redundant) nodes is not harmed by the presence of redundant nodes, which just constitute *additional* branches, each node corresponding to a minimal diagnosis $\mathcal{D}$ will sooner or later be processed by DLABEL. As $\mathcal{D}$ does not reach line 35 and since a minimal diagnosis does hit all (minimal) conflicts, a labeling of $\mathcal{D}$ by $nonmin$ (line 26) is the only possible case. That is, there must be a node labeled $valid$ (and thus stored in $\mathbf{D}_{calc}$) which is a proper subset of $\mathcal{D}$. This is a contradiction to (*).

- Assume the *soundness* is compromised. Then some node nd is labeled $valid$ which is not a minimal diagnosis. Due to (*), it must be the case that nd is a diagnosis, but a non-minimal one. First, a non-minimal diagnosis can never be identified in line 6 because of Property 1.2. Second, since the queue $\mathbf{Q}$ is always sorted such that node n is ranked prior to node n′ whenever $\mathsf{n} \subset \mathsf{n}'$ (cf. Sec. 3.2.1), the non-minimal diagnosis nd can only be labeled $valid$ if some minimal diagnosis $\mathcal{D}$ ($\subset$ nd) —processed prior to nd— is not found to be a diagnosis (and thus not labeled $valid$ and not stored in $\mathbf{D}_{calc}$). Hence, the completeness must be compromised. This is a contradiction to the argumentation in the above bullet.

3.3.3 AVOIDANCE OF EXPENSIVE REASONING

As explained in Sec. 2, DynamicHS aims at reducing the reaction *time* of a sequential diagnosis system. One crucial time-consuming and recurring operation in hitting-set-based diagnosis computation algorithms is the reasoning in terms of logical consistency checks. To optimize computation time, DynamicHS is therefore equipped with strategies that minimize the amount of and time spent for reasoning by exploiting its statefulness in terms of the hitting set tree maintained throughout its iterations. We discuss the concept behind these strategies next.

Ways of Reducing the Cost for Reasoning. In DynamicHS, the logical inference engine is called (solely) by the FINDMINCONFLICT function, which is involved in the determination of node labels in the tree expansion phase (DLABEL function) and in the evaluation of node redundancy in the tree update phase (REDUNDANT function). As discussed in Sec. 2.3, a single execution of FINDMINCONFLICT given the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ generally requires multiple reasoner calls and their number depends critically on the size of the universe $\mathcal{K}$ from which a minimal conflict should be computed. Note, what we, for simplicity, refer to as a *reasoner call* actually corresponds to a check if some $\mathcal{C} \subseteq \mathcal{K}$ is a conflict for a DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$. By the definition of a conflict (see Sec. 2.3), this means checking whether some $x \in N \cup \{\bot\}$ exists such that $\mathcal{C} \cup \mathcal{B} \cup P \models x$. Consequently, a reasoner call corresponds to a maximum of $|N|+1$ logical consistency checks. E.g., if QuickXPlain (Junker, 2004; Rodler, 2020f) is used to implement the FINDMINCONFLICT function, as in our evaluations (cf. Sec. 5), then the worst-case number of consistency checks executed by a single call of FINDMINCONFLICT on the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ is in $O(|\mathcal{K}|(|N|+1))$ (Marques-Silva, Janota, & Belov, 2013). The hardness of consistency checking tends to increase with the size of the knowledge base on which the check is performed (cf., e.g., (Gonçalves, Parsia, & Sattler, 2012)).[22] In other words, the smaller the size of $\mathcal{K} \cup \mathcal{B} \cup P$ is, the more efficient consistency checking will tend to be in the course of FINDMINCONFLICT operating on the DPI $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$.

In summary, the lower the cardinality of the first entry $\mathcal{K}$ (number of system components) of the tuple $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ provided as an input to FINDMINCONFLICT is, the lower the hardness and the number of executed consistency checks will tend to be, and thus the faster FINDMINCONFLICT will tend to execute. Hence, there are basically three different ways of scaling down the necessary reasoning throughout DynamicHS:

(i) Reducing the number and hardness of consistency checks made while FINDMINCONFLICT executes,
(ii) reducing the number of FINDMINCONFLICT calls, or
(iii) entirely avoiding FINDMINCONFLICT calls (and replacing them by equivalents that do not involve reasoning).

In its various stages, DynamicHS embraces all these three approaches, as we explain next.

In the tree expansion phase, any FINDMINCONFLICT call in the course of the conflict reuse check (lines 27–34) starts from an *already computed* conflict—and *not* from the entire set of system components $\mathcal{K}$—trying to verify its minimality or, alternatively, extracting a subset which constitutes a minimal conflict (for the current DPI). That is, FINDMINCONFLICT is given a set of at most $|\mathcal{C}_{\max}|$

[22] Further evidence that larger knowledge bases lead to worse reasoning performance is given in (Kang, Li, & Krishnaswamy, 2012; Karlsson, Nyström, & Cornet, 2014).

elements as an input,[23] where $\mathcal{C}_{\max}$ is the conflict of maximal size (for the original[24] DPI, i.e., the one given as an input to Alg. 1). Note that, in many practical applications involving systems of non-negligible size, $|\mathcal{C}_{\max}|$ is significantly (if not orders of magnitude) smaller than the number of components of the diagnosed system (cf., e.g., (Horridge, Parsia, & Sattler, 2012; Shchekotykhin et al., 2012, 2015)). Thus, DynamicHS applies strategy *(i)* in this stage.

In the tree update phase, and particularly during redundancy detection (line 43), the quick redundancy check (QRC) is employed to potentially replace the complete redundancy check (CRC) by making only a single FINDMINCONFLICT call instead of multiple ones (cf. Sec. 3.3.1). Moreover, any call of FINDMINCONFLICT made in the course of the redundancy detection in general involves a significantly reduced input set, as compared to the overall number of components $|\mathcal{K}|$ of the diagnosed system. The cardinality of this input set is bounded by the cardinality of the union of all minimal conflicts (for the original[25] DPI). So, during redundancy checking, both strategies *(i)* and *(ii)* are pursued.

In the tree pruning phase (PRUNE function), which constitutes a part of the tree update phase, no reasoning is required at all (i.e., no FINDMINCONFLICT calls). This is accomplished by leveraging the stored hitting set tree as well as adequate instructions operating on sets and lists (cf. Example 8 below). At this point, note that a stateless algorithm, in contrast, *has to* draw on logical reasoning to reconstruct (the still relevant) parts of the tree, and thus to achieve essentially the same as DynamicHS's pruning actions. Importantly, operations relying on a logical inference engine can be expected to have a (much) higher time complexity than the list concatenations, set-equality or subset checks performed by DynamicHS in lieu of these operations. For instance, reasoning with propositional logic is already NP-complete (Cook, 1971), not to mention more expressive languages such as Description logics (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2007), whereas set- and list-operations are (mostly linear) polynomial time operations. So, as far as the tree pruning is concerned, DynamicHS can be viewed as trading cheaper (reasoner-free) operations for expensive reasoner calls. It thus makes use of strategy *(iii)*.

Different Types of Reasoning Operations. Given the preceding discussion, we can at the core distinguish the following three categories of FINDMINCONFLICT function calls based on their input DPI $\langle X, \mathcal{B}, P, N\rangle$ (where $\langle \mathcal{K}, \mathcal{B}, P, N\rangle$ is the DPI relevant to the current iteration of DynamicHS):[26]

- *"hard"*: Size of X in the order of number of system components, i.e., $|X| \approx |\mathcal{K}|$, *and* a conflict is returned. (*multiple "hard" reasoner calls*)

[23] When we say that FINDMINCONFLICT gets a set S as an input, we always mean by S the *first* argument of the 4-tuple (DPI) passed to the function as an argument.

[24] Recall from Property 1.3 that minimal conflicts can only become smaller throughout a sequential diagnosis session, i.e., there cannot be any minimal conflict whose size exceeds $|\mathcal{C}_{\max}|$.

[25] Cf. Footnote 24

[26] It is important to note that the used *intuitive* terminology "hard", "medium" and "easy" is to be understood *by tendency*, but does not allow *general* conclusions about the relative or absolute hardness of the respective FINDMINCONFLICT calls. That is, e.g., an "easy" call might not be fast or easy at all. Or a "medium" call might be faster than an "easy" one, e.g., because the latter operates on a set of logical sentences that is particularly hard to reason with (cf. (Gonçalves et al., 2012)). However, as we verified in our experimental evaluation (see Sec. 5), the used terminology does largely reflect the actual relative computation times of the FINDMINCONFLICT calls in our considered dataset. More precisely, on average "hard" calls turned out to be *always* (and at least four times and up to more than 100 times) more time-intensive than "medium" and "easy" calls; and, "easy" calls terminated faster than "medium" ones in 77 % of the studied cases.

Table 3: Stats wrt. the number of different kinds of reasoning operations (FINDMINCONFLICTS calls) throughout the execution of DynamicHS (DHS) and HS-Tree (HST), respectively, on the example DPI from Tab. 1. "Ui" in the first column refers to the tree update performed by DynamicHS subsequent to iteration i. Note, HS-Tree does not (need to) perform any tree updates.

	# "hard"		# "medium"		# "easy"	
iteration	DHS	HST	DHS	HST	DHS	HST
1	4	4	4	4	0	0
U1	0	–	0	–	2	–
2	1	4	0	2	0	0
U2	0	–	0	–	1	–
3	1	4	1	2	0	0
U3	0	–	0	–	1	–
4	0	2	0	1	0	0
total	6	14	5	9	4	0

- *"medium"*: Size of X in the order of number of system components, i.e., $|X| \approx |\mathcal{K}|$, *and* 'no conflict' is returned. (*single "hard" reasoner call*)
- *"easy"*: Size of X low compared to the number of system components, i.e., $|X| \ll |\mathcal{K}|$. (*few "easy" reasoner calls*)

"Hard" FINDMINCONFLICT calls are those executions of line 35 (in DLABEL) that compute a fresh conflict, and "medium" ones those which lead to the finding of a diagnosis (output 'no conflict'). In contrast, "easy" FINDMINCONFLICT invocations are those geared towards redundancy checking (line 43, UPDATETREE) and minimality testing for reused conflicts (line 29, DLABEL). In terms of this characterization, compared against HS-Tree, DynamicHS tries to substantially reduce the "hard" (and "medium") FINDMINCONFLICT operations at the cost of performing an as small as possible number of "easy" ones.

Example 7 Reconsider our example DPI in Tab. 1 and the evolution of the hitting set computation throughout a sequential diagnosis session for DynamicHS and HS-Tree discussed in Example 5. Tab. 3 shows the number of "hard", "medium" and "easy" FINDMINCONFLICT calls throughout the sequential session executed by both algorithms. The "hard" and "medium" calls are denoted by ^{C} and *, respectively, in the hitting set trees depicted by Figs. 3 and 4 ("easy" calls are not indicated as tree updates are not displayed in the figures). We can see that DynamicHS trades a significant reduction of "hard" (57 %) and "medium" (44 %) reasoner operations for some "easy" ones. □

3.3.4 SPACE-SAVING DUPLICATE STORAGE AND ON-DEMAND RECONSTRUCTION

Basically, there are several options how to organize the storage of duplicate nodes. These options range from storing all of them *explicitly* to storing only a minimal set of (stubs of) duplicate nodes that *implicitly* allow all duplicates to be reconstructed on demand. Since DynamicHS performs the duplicate check at node generation time (cf. line 18), it uses the more natural way of handling duplicate storage given by the latter strategy. This means directly adding detected duplicate nodes—generated tree branches whose *set* of edge labels equals the *set* of edge labels of an active branch (node in $\mathbf{Q}$ or $\mathbf{D}_{\supset}$)—to the collection $\mathbf{Q}_{dup}$ without further extending them as the hitting set tree grows. Hence, each node stored in $\mathbf{Q}_{dup}$ is potentially only a partial duplicate node and might

need to be combined with some other (partial) duplicate node or some active node in the hitting set tree to explicitly generate (or: reconstruct) a duplicate that is only implicitly stored. For instance, assume two nodes $\mathsf{n}_1, \mathsf{n}_2$ that have been detected as duplicates and added to $\mathbf{Q}_{dup}$, where $\mathsf{n}_1 = [3,2], \mathsf{n}_1.\mathsf{cs} = [\langle 1,2,3\rangle, \langle 2,4\rangle]$ and $\mathsf{n}_2 = [2,3,1], \mathsf{n}_2.\mathsf{cs} = [\langle 1,2,3\rangle, \langle 3,4\rangle, \langle 1,4\rangle]$ (cf. nodes with numbers ⑤ and ⑨ in Fig. 5, discussed in more detail in Example 8). Then, an implicit duplicate constructible from n_1 and n_2 is $\mathsf{n}_{1,2} = [3,2,1], \mathsf{n}_{1,2}.\mathsf{cs} = [\langle 1,2,3\rangle, \langle 2,4\rangle, \langle 1,4\rangle]$, where the last node label (conflict $\langle 1,4\rangle$) and edge (labeled by 1) of n_2 have been appended to n_1. The rationale behind this node combination is as follows: n_1 was recognized as duplicate first, while the first part (i.e., the first two node and edge labels) of n_2 was still in the queue $\mathbf{Q}$ of open nodes. This first part of n_2 was then extended by the node label $\langle 1,4\rangle$ and the edge label 1, but was subsequently itself spotted as a duplicate. Node n_1, however, given it had still been in $\mathbf{Q}$, would have undergone the same extension. This extension is so to say "made good for" by combining n_1 with n_2 to (re)construct $\mathsf{n}_{1,2}$.

In general, to reconstruct a duplicate node $\mathsf{n}_{i,j}$ from a combination of two nodes $\mathsf{n}_i, \mathsf{n}_j$, the following criteria have to be met:

(RD1) The first $|\mathsf{n}_i|$ elements of n_j interpreted as a set are equal to the elements of n_i interpeted as a set; there are no conditions on the conflict labels $\mathsf{n}_i.\mathsf{cs}$ and $\mathsf{n}_j.\mathsf{cs}$ of the combined nodes.
(RD2) The reconstructed node $\mathsf{n}_{i,j}$ is built by setting[27] $\mathsf{n}_{i,j}[1..|\mathsf{n}_i|] = \mathsf{n}_i$ and $\mathsf{n}_{i,j}[|\mathsf{n}_i|+1..|\mathsf{n}_j|] = \mathsf{n}_j[|\mathsf{n}_i|+1..|\mathsf{n}_j|]$ as well as $\mathsf{n}_{i,j}.\mathsf{cs}[1..|\mathsf{n}_i|] = \mathsf{n}_i.\mathsf{cs}$ and $\mathsf{n}_{i,j}.\mathsf{cs}[|\mathsf{n}_i|+1..|\mathsf{n}_j|] = \mathsf{n}_j.\mathsf{cs}[|\mathsf{n}_i|+1..|\mathsf{n}_j|]$, i.e., the first part of $\mathsf{n}_{i,j}$ and $\mathsf{n}_{i,j}.\mathsf{cs}$, respectively, is equal to n_i and $\mathsf{n}_i.\mathsf{cs}$, to which the last part of n_j and $\mathsf{n}_j.\mathsf{cs}$ is appended.
(RD3) Either *(a)* $\mathsf{n}_i, \mathsf{n}_j$ are each (reconstructed[28] or explicit) nodes from $\mathbf{Q}_{dup}$, or *(b)* node n_i is from the node combination closure $\mathbf{Q}^*_{dup}$ of $\mathbf{Q}_{dup}$ which is the union of $\mathbf{Q}_{dup}$ with the set of all nodes reconstructible[29] through *(a)*, and node n_j is from a node collection including active nodes, i.e., from one of $\mathbf{Q}$, $\mathbf{D}_{\supset}$, $\mathbf{D}_{\checkmark}$, $\mathbf{D}_{\times}$, or $\mathbf{D}_{calc}$.

In the example above, criterion *(RD3)(a)* is met, where n_1 and n_2 correspond to n_i and n_j, respectively, which are both (explicit) elements of $\mathbf{Q}_{dup}$; criterion *(RD1)* is satisfied as well because the first $|\mathsf{n}_i| = |\mathsf{n}_1| = 2$ elements of $\mathsf{n}_j = \mathsf{n}_2$ correspond to the set $\{2,3\}$, which is equal to the set of elements of $\mathsf{n}_i = \mathsf{n}_1$; the validity of criterion *(RD2)* can be easily verified by comparing $\mathsf{n}_{1,2}$ with n_1 and n_2. Let us consider some important remarks:

1. *Node reconstruction is sound and complete:* The set of all nodes constructible by means of *(RD1)*, *(RD2)* and *(RD3)(b)* is exactly the set of all duplicates of the currently active nodes in $\mathbf{Q} \cup \mathbf{D}_{\supset} \cup \mathbf{D}_{\checkmark} \cup \mathbf{D}_{\times} \cup \mathbf{D}_{calc}$.
2. *Relationship between reconstructed node and combined source nodes: (RD1)* implies that $|\mathsf{n}_i| \leq |\mathsf{n}_j|$. By *(RD2)* node reconstruction means that the node n_i of lower (or equal) length replaces the first part of (or the complete) node n_j; we can thus call n_i the *modifying node*

[27] Notation: Given a list n, $\mathsf{n}[k..l]$ refers to the sublist including all elements from the k-th (included) until the l-th (included).

[28] Note the recursive character of this definition. That is, all combinations of explicit and already reconstructed nodes are possible. E.g., a reconstructed node n_6 can be the result of combining two explicit nodes n_1 and n_2 to reconstruct a node n_3, which in turn is combined with some explicit node n_4, which in turn is combined with a reconstructed node n_5.

[29] Formally, $\mathbf{Q}^*_{dup}$ can be defined as the fixpoint S^* of the sequence of sets $S_0, S_1, \ldots$ resulting from the iterative application of the $Comb$ function starting from $S_0 := \mathbf{Q}_{dup}$ where $S_{i+1} = Comb(S_i)$ and $Comb$ is defined as $Comb(S) = S \cup \{\mathsf{n}_{i,j} \mid \mathsf{n}_{i,j}$ is the result as per *(RD2)* of combining two nodes $\mathsf{n}_i, \mathsf{n}_j \in S$ which meet *(RD1)*$\}$.

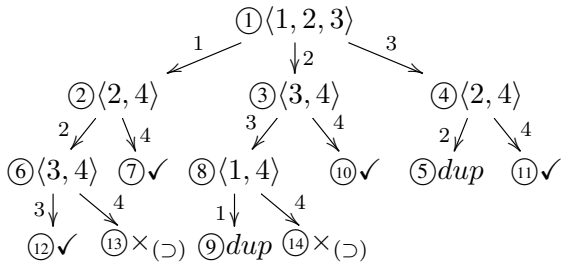
and n_j the *modified node*. Moreover, the reconstructed node has the same length as and is *set*-equal (wrt. *edge* labels) to node n_j. As a consequence of this, node reconstructions can never lead to nodes that are new in terms of their sets of edge labels. Hence, as far as *sets* of edge labels of nodes are concerned, $\mathbf{Q}_{dup}$ is representative of $\mathbf{Q}^*_{dup}$.

3. *Reconstruction of nodes only on demand:* Neither $\mathbf{Q}^*_{dup}$ (as per *(RD3)(a)*) nor the set of all duplicates of active nodes (as per *(RD3)(b)*) is ever exlicitly generated by DynamicHS. Instead, only a minimal number of node reconstructions necessary for the proper-functioning (completeness) of DynamicHS are performed. More specifically, node reconstructions can only take place in case one node has been pruned and a replacement node for it is sought (cf. PRUNE function, Sec. 3.2.5). And, for each pruned node, either just one replacement node is reconstructed, or none at all if no suitable replacement node exists.
4. *Principle of node reconstruction in the course of tree pruning:* Assume the PRUNE function is called given the minimal conflict X and finds some redundant node nd, i.e., X is a witness of redundancy for nd (cf. Sec. 3.2.5). Since there might be multiple edge and conflict labels in nd and $\mathsf{nd.cs}$ due to which nd is redundant given X, let k be the maximal index such that $X \subset \mathsf{nd.cs}[k]$ and $\mathsf{nd}[k] \in \mathsf{nd.cs}[k] \setminus X$ (redundancy criterion, cf. page 19). After deleting nd, a replacement node for it is sought. A replacement node nd' of nd needs to be *(i)* non-redundant (as per the current knowledge, i.e., X must not be a witness of redundancy for nd') and *(ii)* set-equal to nd (cf. Sec. 3.2.5).
   Due to (ii) and Remark 2 above, the redundant node nd can be interpreted as n_j and the sought replacement node as $\mathsf{n}_{i,j}$. With that said, the task of finding a replacement node is equivalent to finding a non-redundant (as per X) node n_i in $\mathbf{Q}^*_{dup}$, see *(RD3)*, such that $|\mathsf{n}_i| \geq k$ (i.e., at least the redundant part of $\mathsf{n}_j = \mathsf{nd}$ is replaced by n_i), see *(RD2)*, and the set of elements of n_i is equal to the set of the first $|\mathsf{n}_i|$ elements of $\mathsf{n}_j = \mathsf{nd}$, see *(RD1)*. Since $\mathbf{Q}_{dup}$ is representative of $\mathbf{Q}^*_{dup}$ in terms of the *sets* of edge labels of nodes (see Remark 2) and because n_i must only be suitable in terms of *set*-equality, it is sufficient to search for n_i in $\mathbf{Q}_{dup}$ as opposed to $\mathbf{Q}^*_{dup}$. Due to (i) and since n_i is a node from $\mathbf{Q}_{dup}$, it must be provided that each node from $\mathbf{Q}_{dup}$ that qualifies as n_i in the search for a replacement node is non-redundant. This imposes two requirements, as pointed out in Sec. 3.2.5: $\mathbf{Q}_{dup}$ must be pruned previous to all other node collections (to account for case *(RD3)(b)*), and nodes of $\mathbf{Q}_{dup}$ must be pruned in ascending order of their length (to account for case *(RD3)(a)*[30]).

Example 8 We now showcase the workings of DynamicHS's tree update on a simple example, thereby also illustrating the discussed advanced techniques regarding the *efficient redundancy checking* (Sec. 3.3.1), the *avoidance of expensive reasoning* (Sec. 3.3.3), as well as the *space-saving duplicate storage and on-demand reconstruction* (Sec. 3.3.4). Note that we already discussed the *lazy updating policy* (Sec. 3.3.2) in terms of Example 5.

Consider Fig. 5 (with a similar notation as used in Figs. 3 and 4) which depicts the hitting set tree produced by DynamicHS (iteration 1) for some DPI dpi_0 in breadth-first order (see the

[30]Recall from Remark 2 that $|\mathsf{n}_i| \leq |\mathsf{n}_j|$. If $|\mathsf{n}_i| < |\mathsf{n}_j|$, then a processing of nodes in order of ascending node cardinality guarantees that all still available (non-pruned) nodes n_i are already verified non-redundant when some n_j might need to be replaced. If, on the other hand, $|\mathsf{n}_i| = |\mathsf{n}_j|$, then there are two cases: n_i is processed prior to n_j, or the opposite holds. In the former case, n_i (unless pruned) must already be verified non-redundant when n_j is considered. In the latter case, n_i is not available as a replacement node at the time n_j is addressed, but n_i itself will be processed later. Thus, if n_j is pruned and n_i non-redundant, then the latter will remain in $\mathbf{Q}_{dup}$ which means that n_i has essentially replaced n_j.

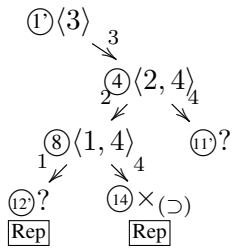


DynamicHS-Tree for dpi_0

$\mathbf{conf}(dpi_0) = \{\langle 1,2,3\rangle, \langle 2,4\rangle, \langle 3,4\rangle, \langle 1,4\rangle\}$
$\mathbf{diag}(dpi_0) = \{[1,2,3],[1,4],[2,4],[3,4]\} = \mathbf{D}$

one measurement m added $\Rightarrow$ $\mathbf{D}_\checkmark = \{[1,2,3],[3,4]\}, \mathbf{D}_\times = \{[1,4],[2,4]\}$ $\Rightarrow$ result: dpi_1

$\mathbf{conf}(dpi_1) = \{\langle 3\rangle, \langle 2,4\rangle, \langle 1,4\rangle\}$
$\mathbf{diag}(dpi_1) = \{[1,2,3],[3,4]\}$



DynamicHS-Tree after UPDATETREE

Figure 5: Tree pruning and redundancy checking example.

node numbers ⓣ signalizing that the respective node was generated at point in time t). The sets of minimal conflicts and minimal diagnoses for dpi_0 are given by $\mathbf{conf}(dpi_0)$ and $\mathbf{diag}(dpi_0)$ in the figure. We assume that DynamicHS uses the parameter $ld := 5$, i.e., five minimal diagnoses (if existent) should be computed. Since there are only four minimal diagnoses (see $\mathbf{diag}(dpi_0)$), DynamicHS executes until the queue is empty ($\mathbf{Q} = [\,]$, see line 4 in Alg. 3), i.e., until the hitting set tree is complete. The resulting set of leading diagnoses $\mathbf{D}$ corresponds to $\mathbf{diag}(dpi_0)$ (nodes labeled by ✓ in Fig. 5). Further, we assume that a (discriminating) measurement m is added to dpi_0, which leads to the new minimal conflict $\langle 3\rangle$ for the resulting DPI dpi_1. As can be seen through a comparison of $\mathbf{diag}(dpi_0)$ with $\mathbf{diag}(dpi_1)$ in Fig, 5, the leading diagnoses eliminated by the measurement m are the nodes numbered ⑦ (corresponding to the node $[1,4]$) and ⑩ (node $[2,4]$). These two nodes are included in the set $\mathbf{D}_\times$ given as an input argument to the second call of DynamicHS (iteration 2) in line 6 of Alg. 1.

Now, when UPDATETREE is invoked in iteration 2, the first step is the examination of the elements in $\mathbf{D}_\times$ regarding their redundancy status (see description of the UPDATETREE function in Sec. 3.2.5). For node $\mathsf{nd} = [1,4]$, we have $\mathsf{nd.cs} = [\langle 1,2,3\rangle, \langle 2,4\rangle]$. The QRC (see Sec. 3.3.1)

executed on nd involves calling FINDMINCONFLICT with argument[31] $(\langle U_{\mathsf{nd.cs}} \setminus \mathsf{nd}, \dots \rangle)$ which is equal to $(\langle \{1,2,3,4\} \setminus \{1,4\}, \dots \rangle) = (\langle \{2,3\}, \dots \rangle)$. Hence, the conflict $X = \langle 3 \rangle$ is returned (cf. $\mathbf{conf}(dpi_1)$), which is a subset of $\mathsf{nd.cs}[1]$ and thus must constitute a witness of redundancy of $\mathsf{nd} = [1,4]$.

As a next step, PRUNE is called with argument X (line 44 in Alg. 3). At first, PRUNE considers $\mathbf{Q}_{dup} = [[3,2],[2,3,1]]$ (nodes numbered ⑤ and ⑨ in Fig. 5) and cleans it up from redundant nodes. At this, for each node $\mathsf{nd} \in \mathbf{Q}_{dup}$, the algorithm runs through the conflicts $\mathsf{nd.cs}[i]$ from small to large i and, if $X \subset \mathsf{nd.cs}[i]$, *(i)* checks if $\mathsf{nd}[i] \in \mathsf{nd.cs}[i] \setminus X$ (is X a witness of redundancy for nd?) as well as *(ii)* replaces $\mathsf{nd.cs}[i]$ by X (update of internal node labels). At the end of this traversal, the maximal index $i = k$, for which (i) is executed and true, is stored. Since, importantly, nodes are processed in ascending order of their size, the first processed node from $\mathbf{Q}_{dup}$ in this concrete example is $\mathsf{nd} = [3,2]$ with $\mathsf{nd.cs} = [\langle 1,2,3 \rangle, \langle 2,4 \rangle]$. For index $i = 1$, we have $\langle 3 \rangle \subset \langle 1,2,3 \rangle$, which is why $\mathsf{nd.cs}[1]$ is replaced by $\langle 3 \rangle$ in the course of step (ii). However, since $\mathsf{nd}[1] = 3$ is not an element of $\langle 1,2,3 \rangle \setminus \langle 3 \rangle$, check (i) is negative (no redundancy detected). For index $i = 2$, it does not even hold that $\langle 3 \rangle \subset \langle 2,4 \rangle$, hence neither (i) nor (ii) are executed. The overall conclusion from this analysis is that X is not a witness of redundancy for nd. Consequently, there is no evidence up to this point that nd is redundant, which is why nd remains an element of $\mathbf{Q}_{dup}$, however, with the modified conflict labels set $\mathsf{nd.cs} = [\langle 3 \rangle, \langle 2,4 \rangle]$.

For the second node $[2,3,1]$ in $\mathbf{Q}_{dup}$, a similar evaluation leads to the insight that this node is redundant and $k = 1$ (because $\langle 3 \rangle \subset \langle 1,2,3 \rangle$ and $2 \in \langle 1,2,3 \rangle \setminus \langle 3 \rangle$). However, instead of only discarding the node, the algorithm seeks a replacement node that is non-redundant and *set*-equal to $[2,3,1]$. To this end, it iterates through all already processed (and thus provenly non-redundant) nodes in $\mathbf{Q}_{dup}$ (in this case only the node $[3,2]$) and tries to find some node of size l which is set-equal to the first l elements of the redundant node, for some $l \geq k$.

Indeed, since the first $2 \geq k = 1$ elements of nodes $[3,2]$ and $[2,3,1]$ are equal (when considered as a set), a replacement node for the latter can be constructed. This (re)constructed node is given by $\mathsf{ndnew} = [3,2,1]$ with $\mathsf{ndnew.cs} = [\langle 3 \rangle, \langle 2,4 \rangle, \langle 1,4 \rangle]$ (i.e., figuratively spoken, the label dup at node ⑤ is replaced by the path including nodes ⑧ and ⑨, cf. Fig. 5).

Since all nodes of $\mathbf{Q}_{dup}$ have at this point been processed, the PRUNE method shifts its focus to the other collections $\mathbf{Q}$, $\mathbf{D}_{\supset}$, $\mathbf{D}_{\times}$ and $\mathbf{D}_{\checkmark}$. Essentially, the considerations made for these collections are equal to those explicated for $\mathbf{Q}_{dup}$, with the only difference that solely elements of (the already cleaned up) $\mathbf{Q}_{dup}$ are in line for being used in the construction of replacement nodes for redundant ones found in these collections. For instance, the node $\mathsf{n} = [1,2,3]$ with the number ⑫ is detected to be redundant ($k = 1$) when it comes to pruning $\mathbf{D}_{\checkmark}$. Since, however, the first $3 \geq k = 1$ elements of $\mathsf{ndnew} = [3,2,1]$ are set-equal to the first 3 elements of n, the latter is replaced by ndnew. After carrying out all pruning actions provoked by the witness of redundancy $X = \langle 3 \rangle$ (detected by analysing the first node $[1,4] \in \mathbf{D}_{\times}$), the reconstructed replacement node ndnew is now the only remaining node in the tree that corresponds to the set of edge labels $\{1,2,3\}$. The fact that this set does constitute a minimal diagnosis wrt. dpi_1 (cf. $\mathbf{diag}(dpi_1)$) corroborates that the storage and adequate reconstruction of duplicates is pivotal for the completeness of DynamicHS.

[31] Note that we just mention the first element $\mathcal{K}$ of the tuple $\langle \mathcal{K}, \mathcal{B}, P, N \rangle$ passed to FINDCONFLICT and write "..." for the remaining ones for simplicity and brevity. The reason is that we did not discuss the specific DPI underlying this example and that $\mathcal{K}$ (the set from which a minimal conflict is to be computed) is sufficient to understand the discussed points.

Note that the second node $[2, 4]$ that was originally an element of $\mathbf{D}_\times$ has meanwhile already been removed from $\mathbf{D}_\times$ in the course of the pruning actions taken. The reason is that the conflict $X = \langle 3 \rangle$ that was used as a basis for pruning is also a witness of redundancy for $[1, 4]$. In fact, $\mathbf{D}_\times = \emptyset$ holds after conflict $X = \langle 3 \rangle$ has been processed. Hence, the for-loop in line 42 of Alg. 3 terminates and no further pruning operations are conducted.

The pruned tree resulting from the execution of UPDATETREE is shown at the bottom of Fig. 5. Replacement nodes (i.e., those nodes which substitute deleted redundant nodes), are marked by Rep; their node number ⓚ in the pruned tree is the number of the original deleted redundant node. Note that the second replacement node $[3, 2, 4]$ results from (the deleted) node ⑭ by a substitution of its first two edges $[2, 3]$ by the duplicate $[3, 2]$ (node ⑤). In addition, node labels changed during the pruning process are indicated by a prime (') symbol. In this concrete example, e.g., both relabeled nodes ⑪ and ⑫ originally represented minimal diagnoses for dpi_0 and were returned by the first run of DynamicHS in terms of $\mathbf{D}_{calc}$. Then, they were added to $\mathbf{D}_\checkmark$ (line 14 in Alg. 1) since they are consistent with the added measurement m. Finally, at the end of UPDATETREE, node ⑪ and the replacement node of node ⑫ were reinserted into the queue $\mathbf{Q}$ of unlabeled nodes (preservation of best-first property; line 58 in Alg. 3) because they "survived" the pruning.

As to the complexity of the tree pruning performed by DynamicHS in this example, the overall number of reasoner-invoking function calls amounts to merely a single "easy" call of FINDMINCONFLICT (the executed QRC), whereas the (re)construction of a tree equivalent to DynamicHS's pruned tree, carried out in iteration 2 when adopting (the stateless) HS-Tree, requires three "hard" FINDMINCONFLICT calls (computations of conflicts $\langle 3 \rangle$, $\langle 2, 4 \rangle$, $\langle 1, 4 \rangle$). □

3.4 Correctness of DynamicHS

Theorem 1. *Let* FINDMINCONFLICT *be a sound and complete method for conflict computation, i.e., given a DPI, it outputs a minimal conflict for this DPI if a minimal conflict exists, and 'no conflict' otherwise. Then, DynamicHS (Alg. 3) is a sound, complete and best-first minimal diagnosis computation method. That is, given a DPI, DynamicHS computes all (completeness) and only (soundness) minimal diagnoses for this DPI in descending order (best-first property) of their probability as per a probability measure pr given as an input.*

Proof. (Sketch) A fully detailed proof of this theorem can be found in (Rodler, 2015, Sec. 12.4). Here, we explicate the main proof ideas.

Proof of Completeness and of the Best-First Property: We have to show that DynamicHS, given the parameter $ld := k$, outputs the k best minimal diagnoses (according to pr). First, we make two general observations, and then we prove the completeness by contradiction.

(Observation 1): Only such nodes can be deleted by PRUNE which are provably redundant (i.e., irrelevant), and, whenever existent, a suitable replacement node is extracted from $\mathbf{Q}_{dup}$ (which stores all possible replacement nodes) to replace the deleted node (cf. Sec. 3.2.5).

(Observation 2): During the execution of DynamicHS given the DPI dpi_k, only diagnoses for dpi_k can be added to $\mathbf{D}_{calc}$.

(Proof by Contradiction): Assume that DynamicHS returns a set $\mathbf{D}_{calc}$ with $|\mathbf{D}_{calc}| = k$ and one of the k best (according to pr) minimal diagnoses is not in $\mathbf{D}_{calc}$ (i.e., has not been computed by DynamicHS). We denote this non-found diagnosis by $\mathcal{D}'$. Since $\mathcal{D}'$ is one of the k best minimal diagnoses, we have that some of the returned k diagnoses in $\mathbf{D}_{calc}$ must have a lower probability as

per pr than $\mathcal{D}'$; let us call this diagnosis $\mathcal{D}''$. Below, we will show that, for any minimal diagnosis $\mathcal{D}$ for the relevant DPI $\langle \mathcal{K}, \mathcal{B}, P \cup P', N \cup N' \rangle$ considered by DynamicHS, the following invariant (*INV*) holds during the execution of the main while-loop (between lines 4–21) of DynamicHS: $\mathcal{D} \in \mathbf{D}_{calc}$ *or there is some node* $\mathsf{n} \subseteq \mathcal{D}$ *in* $\mathbf{Q}$. Due to the properties of pr (cf. Sec. 3.2.1) and because $\mathcal{D}'$ has a greater probability than $\mathcal{D}''$, each subset of $\mathcal{D}'$ has a greater probability than $\mathcal{D}''$. Since $\mathcal{D}' \notin \mathbf{D}_{calc}$ by assumption, we infer that $\mathsf{n} \in \mathbf{Q}$ for some node $\mathsf{n} \subseteq \mathcal{D}'$ when DynamicHS terminates. This is a contradiction to $\mathcal{D}'' \in \mathbf{D}_{calc}$ due to the sorting of $\mathbf{Q}$ in descending order of probability and the fact that only elements of $\mathbf{Q}$ can be added to $\mathbf{D}_{calc}$ in DynamicHS.

We now demonstrate that the invariant INV holds, by induction over the number n of times DynamicHS has already been called in Alg. 1.

(Induction Base): Assume $n = 1$, i.e., DynamicHS is called for the first time in Alg. 1. This has three implications: *(1)* At the time the while-loop is entered, the empty node $[]$ (cf. line 3 in Alg. 1), which is a subset of any minimal diagnosis, is in $\mathbf{Q}$. Hence, INV holds from the outset. *(2)* $\mathbf{D}_{\checkmark} = \emptyset$ (cf. line 2 in Alg. 1 and note that UPDATETREE does not modify $\mathbf{D}_{\checkmark}$). As a consequence, for each node from $\mathbf{Q}$ that is processed throughout the execution of the while-loop of DynamicHS, the DLABEL function is called (because line 7 cannot be reached). *(3)* The PRUNE function (line 33) cannot be called during the execution of the while-loop (as there cannot be any non-minimal conflicts due to the soundness of FINDMINCONFLICT).

Now, assume an arbitrary minimal diagnosis $\mathcal{D}$ for the DPI dpi_0 relevant in the first call of DynamicHS. We next show that INV remains true for $\mathcal{D}$ after any node node is processed within DynamicHS's while-loop. There are two possible cases: *(a)* $\mathsf{node} \not\subseteq \mathcal{D}$ and *(b)* $\mathsf{node} \subseteq \mathcal{D}$.

Let us consider case (a) first. Since no pruning is possible as argued above, i.e., no nodes can be deleted from any node collection stored by DynamicHS except $\mathbf{Q}$ (cf. line 5), the following holds. The processing (and deletion from $\mathbf{Q}$) of node, which is in no subset-relationship with $\mathcal{D}$, *(i)* cannot effectuate an elimination of any other node from $\mathbf{Q}$ (in particular, this holds for all nodes being equal to or a subset of $\mathcal{D}$), and *(ii)* cannot modify $\mathbf{D}_{calc}$. Hence, since INV held before node was processed, INV must still hold thereafter in case (a).

Now, assume case (b). Here, we have again two cases: *(b1)* $\mathsf{node} \subset \mathcal{D}$ and *(b2)* $\mathsf{node} = \mathcal{D}$. Suppose (b1) first, i.e., let $\mathsf{node} \subset \mathcal{D}$ be processed. Because DLABEL is called for any processed node by the argumentation above, we have that line 37 is the only place where nodes can be assigned the label $valid$. Hence, if some node is assigned the label $valid$, this means that FINDMINCONFLICT in line 35 must have returned 'no conflict' for it, which is why this node is a diagnosis by the completeness of FINDMINCONFLICT. Consequently, node cannot be labeled $valid$ because it is a proper subset of a minimal diagnosis. Moreover, node cannot be labeled $nonmin$ in line 26 as there cannot be a subset of node in $\mathbf{D}_{calc}$ due to Observation 2 and the fact that node is a proper subset of a minimal diagnosis. As a result, the DLABEL function must return in either of the lines 31, 34 or 40, in each of which cases a minimal conflict set is returned. This conflict L is then used to generate a new node node_e for each $e \in L$ (lines 15–17), where $|\mathsf{node}_e| = |\mathsf{node}| + 1$ and, for some e, $\mathsf{node}_e \subseteq \mathcal{D}$ must hold (if the latter was not the case then $\mathcal{D}$ would not hit the minimal conflict L, which is a contradiction to $\mathcal{D}$ being a minimal diagnosis). For each of these nodes node_e, either a set-equal node is already in $\mathbf{Q}$ or node_e is added to $\mathbf{Q}$ (cf. lines 18–21 and note that no node set-equal to node_e can be in $\mathbf{D}_{\supset}$ due to Observation 2 and lines 24–26). Hence, in case (b1), INV remains true after node has been processed.

Finally, assume case (b2). Because DLABEL is called for any processed node by the argumentation above, it must be called for node. Since node however is equal to the minimal diagnosis $\mathcal{D}$,

DLABEL will return $valid$ (this follows from Observation 2 and the fact that diagnoses are hitting sets of all conflicts). Due to lines 10–11, this means that $\mathcal{D} \in \mathbf{D}_{calc}$ will hold at the beginning of the next iteration of the while-loop. Consequently, also in case (b2), INV still holds after the processing of node. This completes the proof of the Induction Base.

(Induction Assumption): Assume INV holds for all $n \leq k$.

(Induction Step): Now, let $n = k + 1$. That is, we consider the $(k + 1)$-th call of DynamicHS in Alg. 1. We assume again an arbitrary minimal diagnosis $\mathcal{D}$ for the DPI dpi_k relevant in this call of DynamicHS.[32] By the fact that each minimal diagnosis for dpi_k is either equal to or a superset of some diagnosis for dpi_{k-1} (Property 1.2), and since for each minimal diagnosis $\mathcal{D}'$ for dpi_{k-1} either $\mathcal{D}'$ was in $\mathbf{D}_{calc}$ or some node $\mathsf{n} \subseteq \mathcal{D}'$ was in $\mathbf{Q}$ when the k-th call of DynamicHS returned (Induction Assumption), we infer that some node corresponding to a subset of $\mathcal{D}$ is either in one of $\mathbf{D}_{\checkmark}$ or $\mathbf{D}_{\times}$ (cf. line 14 in Alg. 1 where $\mathbf{D}_{calc}$ is split into $\mathbf{D}_{\checkmark}$ and $\mathbf{D}_{\times}$) or in $\mathbf{Q}$ at the beginning of the $(k + 1)$-th execution of DynamicHS. The first steps in this execution are setting $\mathbf{D}_{calc} = \emptyset$ and calling the function UPDATETREE. Throughout UPDATETREE, some nodes might be pruned (and potentially replaced by set-equal nodes), and all non-pruned nodes from $\mathbf{D}_{\times}$ as well as all nodes from $\mathbf{D}_{\checkmark}$ are finally reinserted into $\mathbf{Q}$. Moreover, each non-pruned node from $\mathbf{D}_{\supset}$ for which there is no known diagnosis that is a subset of it is added to $\mathbf{Q}$ at the end of UPDATETREE. By Observation 1 and since $\mathcal{D}$ is a minimal diagnosis and thus relevant, we have that there must be a node $\mathsf{n} \subseteq \mathcal{D}$ in $\mathbf{Q}$ when the while-loop of the $(k + 1)$-th DynamicHS call is entered. That is, INV holds at the beginning of the while-loop.

That INV remains true for $\mathcal{D}$ after any node node is processed within the while-loop, is shown analogously (i.e., same case analysis and argumentation) as expounded for the Induction Base, except for two aspects: $\mathbf{D}_{\checkmark} \neq \emptyset$ and the PRUNE function (line 33) might be called. Consequences of these aspects are: *(1)* By Observation 1, during the execution of the while-loop of DynamicHS, the last remaining node in $\mathbf{Q}$ which is a subset of some minimal diagnosis cannot be pruned without being replaced by a set-equal node. Neither can a minimal diagnosis be removed from $\mathbf{D}_{calc}$ without being substituted by a set-equal node. Therefore, for every execution of PRUNE (line 33), if INV holds prior to it, INV holds after it finishes. *(2)* Assume $\mathsf{node} \subset \mathcal{D}$ and $\mathsf{node} \in \mathbf{D}_{\checkmark}$. Due to Observation 2 and line 14 in Alg. 1, $\mathbf{D}_{\checkmark}$ includes only diagnoses for the DPI dpi_{k-1} relevant to the preceding (k-th) call of DynamicHS in Alg. 1, and each diagnosis in $\mathbf{D}_{\checkmark}$ during the $(k + 1)$-th call of DynamicHS throughout Alg. 1 is a diagnosis for the DPI dpi_k. Hence, the assumptions $\mathsf{node} \in \mathbf{D}_{\checkmark}$ and $\mathsf{node} \subset \mathcal{D}$ are in contradiction to our assumption that $\mathcal{D}$ is a minimal diagnosis for dpi_k. Equivalently: $\mathsf{node} \subset \mathcal{D}$ implies $\mathsf{node} \notin \mathbf{D}_{\checkmark}$.

The impact of (1) and (2) on the case analysis (cf. Induction Base) is as follows: The argumentation for the case where $\mathsf{node} \not\subseteq \mathcal{D}$ is processed is analogous to case (a) for the Induction Base. The proof for the case $\mathsf{node} \subset \mathcal{D}$ is equal to case (b1) for the Induction Base since DLABEL must be called for node (due to $\mathsf{node} \notin \mathbf{D}_{\checkmark}$). Finally, the case $\mathsf{node} = \mathcal{D}$ is treated as demonstrated in case (b2) in the Induction Base because, if $\mathsf{node} \in \mathbf{D}_{\checkmark}$, then it is simply directly labeled $valid$ in line 7 (no call of DLABEL)—hence, whether or not the DLABEL function is called, $\mathcal{D} \in \mathbf{D}_{calc}$ will hold after node having been processed. This completes the proof of the Induction Step, and thus the entire proof.

[32] Please note that the original DPI considered by the first call of DynamicHS during a sequential diagnosis session is referred to as dpi_0 (cf. line 6 in Alg. 1). Hence, the DPI relevant to the k-th call of DynamicHS is denoted by dpi_{k-1}.

Proof of Soundness: We have to show that DynamicHS outputs only minimal diagnoses. That is, we need to demonstrate that every element in $\mathbf{D}_{calc}$ satisfies the diagnosis property and the minimality property. Since each call of DynamicHS in the course of Alg. 1 outputs one set $\mathbf{D}_{calc}$, we prove the soundness by induction over the number n of times DynamicHS has already been called in Alg. 1.

(Induction Base): Assume $n = 1$, i.e., DynamicHS is called for the first time in Alg. 1 and returns $\mathbf{D}_{calc}$. Let $\mathsf{node} \in \mathbf{D}_{calc}$. A node is added to $\mathbf{D}_{calc}$ iff it has been labeled $valid$. There are two ways a node may be labeled $valid$, i.e., *(i)* in line 7 and *(ii)* in line 9. Note that case (i) is impossible since $n = 1$ which means that $\mathbf{D}_{\checkmark} = \emptyset$ (cf. Alg. 1) and thus line 7 can never be reached. Therefore, case (ii) applies to node. That is, its label $valid$ is assigned by the DLABEL function. Hence, DLABEL must return in line 37. From this we conclude that the FINDMINCONFLICT call in line 35 returns 'no conflict' which implies that node is a diagnosis (due to the completeness of FINDMINCONFLICT). Assume there is a diagnosis $\mathcal{D}$ such that $\mathcal{D} \subset \mathsf{node}$. Since case (ii) is true for node, there cannot be any such diagnosis $\mathcal{D}$ in $\mathbf{D}_{calc}$ due to lines 24–26, because otherwise node would have been labeled $nonmin$ and line 37 could not have been reached. However, due to the completeness of DynamicHS, and since $\mathcal{D}$ must be ranked higher as per pr than node (cf. the definition of pr in Sec. 3.2.1), and since $\mathcal{D}$ is a diagnosis, $\mathcal{D}$ must already be included in $\mathbf{D}_{calc}$ when node is added. This is a contradiction. Therefore, for $n = 1$ (i.e., for the first call of DynamicHS in Alg. 1), the output $\mathbf{D}_{calc}$ contains only minimal diagnoses.

(Induction Assumption): Assume $\mathbf{D}_{calc}$ contains only minimal diagnoses for $n \leq k$.

(Induction Step): Now, let $n = k + 1$ and $\mathsf{node} \in \mathbf{D}_{calc}$. Analogously to the argumentation above, we again have the two possibilities (i) and (ii) of how node might have attained its label $valid$. Suppose case (i) first. That is, $\mathsf{node} \in \mathbf{D}_{\checkmark}$. By line 14 of Alg. 1 (ASSIGNDIAGSOKNOK, cf. Sec. 3.1.3), $\mathbf{D}_{\checkmark} \subseteq \mathbf{D}_{calc}$ where $\mathbf{D}_{calc}$ is the output of the the previous, i.e., the k-th, call of DynamicHS. Due to the Induction Assumption, we have that $\mathbf{D}_{\checkmark}$ includes only minimal diagnoses for the DPI dpi_{k-1} considered in the k-th iteration, i.e., the DPI dpi_k considered in the $(k+1)$-th iteration without the most recently added measurement. However, ASSIGNDIAGSOKNOK adds to $\mathbf{D}_{\checkmark}$ exactly those diagnoses that are consistent with the new measurement. Consequently, the diagnoses in $\mathbf{D}_{\checkmark}$ are consistent with all measurements included in dpi_k, and thus are diagnoses for dpi_k. Due to Property 1.2, no diagnosis for dpi_k can be a proper subset of any diagnosis for dpi_{k-1}. Thus, all elements of $\mathbf{D}_{\checkmark}$ must be minimal diagnoses which is why node must be a minimal diagnosis. For the other case (ii), the argumentation is exactly as for the Induction Base. □

4. Related Work

Algorithms for diagnosis computation can be categorized (at least) according to the following dimensions:

1. *Incomplete* vs. *complete*:[33] Complete algorithms guarantee to output all minimal diagnoses (given arbitrary time and memory). Examples of complete algorithms are HS-Tree (Reiter, 1987), HS-DAG (Greiner et al., 1989), GDE (de Kleer & Williams, 1987), HST-Tree (Wotawa, 2001), StaticHS (Rodler & Herold, 2018), Inv-HS-Tree (Shchekotykhin et al.,

[33] This dimension is sometimes also characterized as *approximate* vs. *exhaustive*, cf., e.g., (Jannach, Schmitz, & Shchekotykhin, 2016). However, note that "approximation" in this categorization refers to the approximation of the set of all (minimal) diagnoses achieved by the algorithm, and not, e.g., that computed sets are only approximate hitting sets of all conflicts, i.e., hit only a predefined minimal fraction of all conflicts (Vinterbo & Øhrn, 2000).

2014), BHS-Tree and Boolean algorithm (Lin & Jiang, 2003), HSSE-Tree (Xiangfu & Dantong, 2006), SDE (Stern, Kalech, Feldman, & Provan, 2012), RBF-HS and HBF-HS (Rodler, 2020d, 2020e), the suite of algorithms presented in (Rodler, 2015), the parallel hitting set algorithms suggested by (Jannach et al., 2016), and a hypergraph inversion method proposed in (Haenni, 1998). Incomplete approaches, in contrast, are usually geared towards computational efficiency, at the cost of not giving a completeness guarantee. Examples of (generally) imcomplete algorithms are Genetic Algorithm (Li & Yunfei, 2002), SAFARI (Feldman et al., 2008), STACCATO (Abreu & van Gemund, 2009), CDA* (Williams & Ragno, 2007), HDiag (Siddiqi, Huang, et al., 2007), and NGDE (de Kleer, 2009). Also, incompleteness might be a consequence of a special focus of the diagnosis search, e.g., if the goal is to determine only (the) minimal cardinality (of) diagnoses (Siddiqi et al., 2007; Shi & Cai, 2010; de Kleer, 2011). In critical diagnosis applications, such as in medical applications (Schulz, Schober, Tudose, & Stenzhorn, 2010; Rector, Brandt, & Schneider, 2011), completeness is an important criterion as incompleteness might prevent the finding of the actual fault in the diagnosed system.[34]

2. *Best-first* vs. *any-first*: Best-first approaches generate diagnoses in a specific order, usually guided by some preference criterion or heuristic, e.g., a probability measure or minimum-cardinality-first. Among those, we further distinguish between *general* best-first and *focused* algorithms. The former, e.g., (de Kleer & Williams, 1987; Reiter, 1987; Greiner et al., 1989; de Kleer, 1991; Rodler, 2015; Rodler & Herold, 2018; Rodler, 2020d, 2020e), can output diagnoses in best-first order according to any (set-)monotonic function[35] that is used as a preference criterion, whereas the latter, e.g., (Darwiche, 2001; Torasso & Torta, 2006; Siddiqi et al., 2007; Abreu & van Gemund, 2009; de Kleer, 2009, 2011), consider only one particular preference criterion, usually minimum-cardinality, or cannot handle arbitrary (monotonic) preference functions. Any-first methods, e.g., (Lin & Jiang, 2003; Pill & Quaritsch, 2012; Shchekotykhin et al., 2014), on the other hand, do not ensure any particular order in which diagnoses are output, but still might attempt to generate preferred diagnoses first in a heuristic way (Rodler & Elichanova, 2020). One particular advantage of best-first methods is that they allow for a more reliable early termination[36] of a sequential diagnosis process (cf. Alg. 1). That is, if one of k computed diagnoses has a very high probability compared to each of the other $k-1$ computed diagnoses, then it has a very high probability compared to each unknown (not yet computed) diagnosis as well (because it is guaranteed that all unknown

[34] Note that soundness, i.e., that the algorithm outputs *only* minimal diagnoses, is a very important property as well (and probably even more crucial than completeness). However, since soundness is usually a minimum presupposed property of diagnosis computation algorithms (and in fact satisfied by almost all algorithms in literature), we do not include a separate dimension that distinguishes between sound and unsound methods. Nevertheless, it is noteworthy that there are two ways an algorithm may violate soundness: it may violate the diagnosis property or the minimality property. Usually, the former is a more severe issue (components that are not related in any way with the system's abnormality might be output), whereas the latter (unnecessarily many components, which however do explain the system's abnormality, may be output) can be fixed by performing a post-processing of the output diagnoses, see, e.g., the μ function used to clean the output of BHS-Tree from non-minimal diagnoses (Haenni, 1998; Lin & Jiang, 2003).

[35] A *(set-)monotonic function* f is one for which $f(X) \leq f(Y)$ whenever $X \subseteq Y$. Note that component fault probabilities satisfying the condition that each component's probability is below 0.5 yield a diagnosis probability measure that is monotone (under the common framework for deriving diagnosis probabilities from component probabilities proposed in (de Kleer & Williams, 1987)). See (Rodler, 2015, Sec. 4.6.2) for more details.

[36] We refer by *early termination* to the stopping of a sequential diagnosis session while there are *multiple* minimal diagnoses left for the considered diagnosis problem, cf. (de Kleer & Williams, 1987).

diagnoses have a lower probability than the k known ones). A pro of any-first algorithms is that they are more flexible in terms of the search strategy pursued in the diagnosis search. For example, the Inv-HS-Tree method proposed by (Shchekotykhin et al., 2014) does not need to stick to a (worst-case exponential-space) breadth-first or uniform-cost search (as comparable general best-first complete algorithms usually have to), but can use a more space-efficient depth-first search strategy. Hence, there are problems that can be solved by means of any-first methods, where similar best-first approaches run out of memory (Shchekotykhin et al., 2014). Unfortunately, any-first methods are not always appropriate and useful, e.g., for diagnosis tasks where (only) minimum-cardinality diagnoses are of interest.[37]

3. *Conflict-based* vs. *direct*: Conflict-based algorithms[38], e.g., (de Kleer & Williams, 1987; Reiter, 1987; Greiner et al., 1989; Wotawa, 2001; Lin & Jiang, 2003; Xiangfu & Dantong, 2006; de Kleer, 2011; Stern et al., 2012; Rodler, 2015; Jannach et al., 2016; Rodler & Herold, 2018; Rodler, 2020d, 2020e), rely on the (pre-)computation of (a set of) conflicts and diagnoses are determined as hitting sets of all conflicts for a diagnosis problem. On the contrary, direct algorithms compute diagnoses without the indirection via conflicts. Direct algorithms can be divided into approaches based on *(i) divide-and-conquer computation* (Felfernig, Schubert, & Zehentner, 2012; Marques-Silva, Heras, Janota, Previti & Belov, 2013; Mencia, & Marques-Silva, 2014; Shchekotykhin et al., 2014) and *(ii) compilation techniques* that translate the problem to a target language such as prime implicates (de Kleer, 1990), decomposable negation normal form (DNNF) (Darwiche, 2001), ordered binary decision diagrams (OBDD) (Torasso & Torta, 2006) or SAT (Metodi, Stern, Kalech, & Codish, 2014; Marques-Silva, Janota, Ignatiev & Morgado, 2015). Methods of class (i), which are realized by means of (adapted) algorithms (Junker, 2004; Marques-Silva, Janota, & Belov, 2013) for solving the MSMP[39] problem, are attractive, e.g., in diagnosis domains[40] involving systems where a high number of components tend to be simultaneously faulty. In such a case, they mitigate memory issues arising for (complete) conflict-based algorithms by their ability to utilize space-efficient search techniques while preserving soundness of diagnosis computation (Shchekotykhin, Rodler, Fleiss, & Friedrich, 2012). A drawback of these algorithms is their usual inability to enumerate diagnoses best-first as per a predefined preference order. An advantage of the problem-rewriting techniques of class (ii) is their ability to leverage highly optimized solving techniques from other domains (e.g., SAT), or the usually achieved polynomial complexity of (diagnostic) inference once the target representation has been compiled.

[37] Any-first methods would need to compute *all* minimal diagnoses to find the minimum cardinality of diagnoses and thus to return a set of (proven) minimum-cardinality diagnoses.

[38] These are sometimes referred to as *conflict-to-diagnoses algorithms*, see, e.g., (de Kleer, 2011).

[39] MSMP refers to the problem of finding a *minimal set over a monotone predicate* (Marques-Silva, Janota, & Belov, 2013; Marques-Silva, Janota, & Mencia, 2017). A *monotone predicate* p is a function that maps a set to a value in $\{0, 1\}$ with the property that $p(Y) = 1$ whenever $p(X) = 1$ and $Y \supseteq X$. Instances of MSMP problems are manifold, among them the problems of computing prime implicates (Marquis, 2000), minimal conflicts (Junker, 2004), minimal hitting sets (Felfernig et al., 2012), or minimal equally discriminating measurements for sequential diagnosis (Rodler et al., 2017).

[40] One such domain is given by *ontology alignment / matching* (Euzenat, Meilicke, Stuckenschmidt, Shvaiko, & Trojahn, 2011) applications, where automated systems propose a set of (logical) correspondences between terms of two input knowledge bases describing similar domains, with the aim to integrate the expressed knowledge in a meaningful way. An example of such a correspondence would be that *human* (term of the first ontology) is equivalent to *person* (term of the second ontology). Since many correspondences are added to the union of both ontologies at once, it is often the case that a large set of conflicts arises (Meilicke, 2011).

On the downside, there is no guarantee that the representation is not exponential in size, and the published theories for compiling system descriptions mainly focus on propositional logic, which makes the (efficient) applicability of such techniques questionable for systems modeled in languages whose expressivity goes beyond propositional logic, e.g., ontology debugging problems (Kalyanpur, 2006; Shchekotykhin et al., 2012).

Conflict-based techniques can be further categorized as follows:

(a) *On-the-fly* vs. *preliminary* (conflict computation): Since the precomputation of (all) minimal conflicts can be very expensive and is generally intractable[41], on-the-fly algorithms, e.g., (Reiter, 1987; Greiner et al., 1989; Wotawa, 2001; Rodler, 2015; Jannach et al., 2016; Rodler & Herold, 2018; Rodler, 2020d, 2020e), compute conflicts on demand in the diagnosis computation process. The main rationale behind this on-demand strategy is that diagnoses (hitting sets of all conflicts) can be computed even if not all conflicts are explicitly known[42], and that conflict computation is usually (by far) the most expensive operation in the course of diagnosis computation (cf., e.g., (Pill et al., 2011)). Especially in a sequential diagnosis setting (which is also the focus of this work), where a sample of (in principle only two (Rodler, 2015)) computed diagnoses suffices and the diagnoses have the primary function to serve as a guideline and give evidence for proper measurement selection, normally not nearly all conflicts need to be precomputed to obtain this sample. Preliminary algorithms, on the other hand, assume the collection of conflicts to be given as an input to the diagnosis computation procedure. Instances of preliminary algorithms are (Li & Yunfei, 2002; Lin & Jiang, 2003; Xiangfu & Dantong, 2006; Abreu & van Gemund, 2009; de Kleer, 2011; Pill & Quaritsch, 2012). A way to view preliminary approaches is that they decouple the (model-based) reasoning, or, equivalently, the conflict computation, from the hitting set calculations. Sometimes the conflicts also do not result from reasoning, but from simulation, e.g., in spectrum-based diagnosis approaches (Abreu, Zoeteweij, & van Gemund, 2007), where "conflicts" correspond to so-called (program) spectra.[43] There are also systems for which the categorization is not clear-cut, such as GDE (de Kleer & Williams, 1987), which interleaves conflict computation phases with diagnosis finding phases. We finally note that any on-the-fly method can be used in a preliminary fashion (by precomputing, if possible, all conflicts and then using the collection of these conflicts instead of the on-the-fly reasoning); the inverse does not hold in general.

(b) *Centralized* vs. *distributed*: Centralized algorithms (of which *on-the-fly* and *preliminary* instances exist) consider the diagnosis computation from conflicts as *one* problem. Note that all of the works referenced when discussing conflict-based algorithms in 3 above are centralized. In contrast, distributed approaches (which are collectively *preliminary*) attempt to leverage the structure inherent in the collection of conflicts, e.g.,

[41] The computation of conflicts is dual to the computation of diagnoses (Stern et al., 2012), and hence it is NP-hard to decide if there is an additional minimal conflict given a set of minimal conflicts (Bylander et al., 1991).

[42] Note that to verify that a set is a diagnosis (i.e., hits all, possibly unknown, conflicts) a consistency check is sufficient. The hit conflicts thus do not need to be explicitly given or computed.

[43] Roughly, a (program execution) spectrum is a set (sequence) of the system components (often: lines of program code) involved in one particular execution of the system (often: program). If an execution exhibits faulty behavior, the associated spectrum can be viewed as a conflict since at least one component in it must be faulty.

linearity (Zhao, 2016), or equivalence classes based on conflict overlapping and disjointness (Zhao & Ouyang, 2013), to speed up the hitting set computation. More precisely, they seek to decompose the hitting set computation problem into suitable sub-problems, consider these sub-problems separately and finally merge the obtained partial results to determine the overall result in terms of the hitting sets. Obviously, the (potentially large) performance gains achieved by distributed approaches depend strongly on the "richness" of the exploitable structure in the conflict data. A drawback of such approaches is that a reasonable a-priori analysis for proper problem decomposition requires the precomputation of (a substantial set of) conflicts.

4. *Black-box* vs. *glass-box*: Black-box techniques, e.g., (Reiter, 1987; Greiner et al., 1989; Wotawa, 2001; Shchekotykhin et al., 2014; Rodler, 2015; Jannach et al., 2016; Rodler & Herold, 2018; Rodler, 2020d, 2020e), use the logical inference engine as is, as a pure oracle that answers queries (i.e., performs consistency checks) throughout the diagnosis computation process.[44] Hence, black-box techniques are completely *reasoner-independent* and able to use *any logic* along with *any reasoner* that can perform sound and complete consistency checks over this logic. Glass-box techniques (Kalyanpur, 2006; Schlobach et al., 2007; Baader & Penaloza, 2008; Horridge, 2011) try to make the determination of diagnoses more efficient by exploiting non-trivial modifications of the internals of the reasoner. These can be algorithmic modifications—e.g., the extraction of a conflict or conflict-related information as a byproduct of a negative consistency check (Parsia et al., 2005)—as well as strategies based on gainful memory utilization—e.g., the TMS-based[45] bookkeeping of sets of logical axioms that are sufficient for particular inferred entailments to hold (de Kleer, 1986; de Kleer & Williams, 1987; de Kleer, 1991). Glass-box approaches *depend on* the (suitable adaptation of) *one particular reasoner* that can deal with *one particular (class of) logic(s)*.[46] While glass-box approaches are highly optimized for particular logics and can bring noticeable performance gains over black-box approaches in certain cases (Kalyanpur, 2006; Horridge, 2011), advantages of black-box methods in the context of model-based diagnosis are their *robustness* (no sophisticated, and potentially error-prone, modifications of complex reasoning algorithms), *simplicity* (internals of reasoner irrelevant), *generality* (applicable to diagnosis problem instances formulated over any knowledge representation formalism for which a sound and complete reasoner exists) and *flexibility* (e.g., black-box methods can use a portfolio reasoning approach where the most efficient reasoner is used depending on the expressivity of the knowledge base that provides the reasoning context[47]). In-depth com-

[44]The distinction between black-box and glass-box techniques in this context was first suggested by Parsia et al. (Parsia, Sirin, & Kalyanpur, 2005).

[45]TMS is a shortcut for *truth maintenance system*.

[46]Note that Baader and Penaloza (Baader & Penaloza, 2008) provide a more general approach for extending different types of reasoners to extract debugging-relevant information throughout the reasoning process. The suitable adaptations however still need to be done on each particular reasoner in order to use it in a glass-box scenario.

[47]Recall from Sec. 3.3.3 that DynamicHS performs its consistency checks often with regard to significantly different parts of the knowledge base (diagnosed system), where the respective reasoning context is determined by (the labels along) one particular path in the hitting set tree. Although the logical expressivity of the complete knowledge base (system description) might necessitate the use of a reasoner with unfavorable worst-case runtime, parts of the knowledge base might well fall into a lower logical expressivity class, thus enabling the use of more performant reasoning approaches that need not be complete wrt. the expressivity of the full knowledge base. E.g., if the goal is to diagnose a system described by a Description Logic (Baader et al., 2007) knowledge base, the full system descrition might be stated, say, in

parisons (Kalyanpur, 2006; Horridge, 2011) between black-box and glass-box strategies in the domain of knowledge-base debugging conclude that, in terms of performance, black-box methods overall compete fairly reasonably with glass-box methods while offering a higher generality and being more easily and robustly implementable.

5. *Holistic* vs. *abstraction-based* vs. *alteration-based*: Holistic approaches, to which most of the above-mentioned algorithms belong, compute diagnoses by considering (the description of) the system to be diagnosed as is. In contrast, abstraction-based methods abstract from the original system by reformulating the problem in a more concise way, and alteration-based methods exploit modifications of the system description with well-known properties. The goal of these steps is to achieve better computation efficiency or to be able to solve problems that are otherwise too large or complex. Examples of abstraction-based techniques are (Mozetič, 1991; Out, van Rikxoort, & Bakker, 1994; Siddiqi et al., 2007), which pursue a stepwise hierarchical approach where, starting from a maximally abstract system description, successively more refined (in the sense of: more similar to the original system description) abstractions of those system parts revealed to be abnormal while analyzing the abstract model are considered to obtain the diagnoses for the original system. The core principle behind these approaches is to reduce the search space by first using the abstract model and to reduce the search at the detailed level by excluding those system parts that have already been exonerated at the abstract level. An alteration-based technique is proposed in (Siddiqi & Huang, 2011) where components in a system model are cloned in order to achieve a reduction of the generated system abstraction, while not losing relevant diagnostic information. Other alteration-based techniques safely discard considerable portions of the system description while maintaining full soundness and completeness wrt. the diagnostic task, e.g., in the knowledge-base debugging domain this can be achieved by using modules based on syntactic locality (Grau, Horrocks, Kazakov, & Sattler, 2008a; Sattler, Schneider, & Zakharyaschev, 2009).

6. *Stateful* vs. *stateless* (during sequential diagnosis): Stateful approaches, e.g., (de Kleer & Williams, 1987; Siddiqi & Huang, 2011; Rodler, 2015; Rodler & Herold, 2018; Penaloza, 2019), maintain state (e.g., in terms of stored and later reused data structures) throughout a sequential diagnosis session. That is, data produced during one iteration constitutes an input to the execution of the next iteration, where the transition from one to the next iteration is defined by the incorporation of new information (resulting from a performed measurement) to the diagnosis problem. Stateless algorithms, e.g., (Reiter, 1987; Greiner et al., 1989; Wotawa, 2001; Xiangfu & Dantong, 2006; de Kleer, 2011; Jannach et al., 2016), in contrast, do not propagate any information between two iterations. They can be thought of as getting a full reset after each iteration and starting completely anew, albeit with a modified input (previously considered diagnosis problem *including additional measurements*), in the next iteration.

The bottom line of this categorization and discussion of diagnosis computation algorithms is:

---

the logic $\mathcal{SROIQ}$ (Grau et al., 2008b), whereas parts thereof might be formulated, say, in $\mathcal{EL}$ (Baader, Brandt, & Lutz, 2005), which would allow to employ reasoners, say (Baader, Lutz, & Suntisrivaraporn, 2005) or (Kazakov, Krötzsch, & Simančík, 2014), that are particularly efficient for this more restricted language.

- Although in part highly different as regards algorithm type and properties, in principle, all algorithms have their right to exist; they are well-motivated by theoretical or practical problems or shortcomings of pre-existing algorithms.

- For most pairs of algorithms with a mutually different characterization in terms of the discussed dimensions 1–6, there will be problem classes that can be solved (more efficiently) with the one algorithm than with the other, and other problem classes for which the inverse holds. Simply put, the appropriateness of diagnosis computation algorithms is domain-dependent and different domains require different algorithmic techniques and algorithm properties. Thus, it is reasonable to expect that the one and only diagnosis algorithm that suits all possible model-based diagnosis scenarios best does not exist.
  Let us consider two examples: *(1)* Inv-HS-Tree appears to be not that well-suited for (spectrum-based) debugging problems focusing on *procedural* software because it can be difficult (or even impossible) to enforce the execution of exactly the program traces required by the splitting strategy of the algorithm; this, however, is trivial for debugging problems involving *declarative* knowledge bases. STACCATO, on the other hand, does not ideally cater for knowledge-base debugging problems where a pre-computation of conflicts is often impractical and unnecessary (especially in a sequential diagnosis setting), but is very powerful in the (spectrum-based) software domain. *(2)* While the GDE and its successors are highly optimized and particularly well-performing for physical devices or digital circuits, they have not found widespread adoption in some other diagnosis domains such as knowledge-base debugging. A likely reason for that is the typically different justification[48] structure inhering physical devices on the one hand and knowledge bases on the other. Specifically, the principle of *locality* (de Kleer, 1990) (constraints for a particular system component only interact with constraints for physically adjacent components) appears to benefit justification bookkeeping strategies as used in GDE, whereas the potential strong non-locality and the related richer justification structure present in knowledge bases[49] can make such bookkeeping methods inefficient.[50] It seems that other techniques, such as HS-Tree, that abstain from protocoling computed justifications, work better in this particular domain.

- If one algorithm only improves on another one's performance on all (reported) problems, then this often comes at the cost of sacrificing a (potentially useful) property of the improved algorithm. For instance, Inv-HS-Tree reduces the space-complexity of the diagnosis search from exponential to linear as compared to HS-Tree, but diagnoses cannot be enumerated best-first any longer. Other examples are distributed algorithms that can meliorate the diagnosis *search* time significantly, but typically require a (usually expensive) precomputation of all conflicts, or glass-box algorithms that can notably improve the cost of diagnosis determination while, however, requiring systems expressible in one particular (class of) logic(s).
  Exceptions to this are particularly notable, i.e., when one algorithm improves on another one

[48] A *justification* (for α) is a minimal set of assumptions (axioms) that are sufficient for a particular entailment α to hold (Horridge, Parsia, & Sattler, 2008). A justification for an unwanted entailment, such as $\bot$, is a minimal conflict. Depending on the context, justifications are sometimes referred to as *environments* (de Kleer & Williams, 1987).

[49] As studied by Horridge et al. (Horridge et al., 2012), the number of justifications for a single entailment α (and thus the number of conflicts) might well reach high three-digit numbers in knowledge-based systems.

[50] This issue was discussed in plenum during the "Workshop on Principles of Diagnosis 2017" (DX'17) in Brescia, Italy.

while preserving all its (desirable) properties. For instance, a compilation of the diagnosis problem to DNNF can be used whenever a compilation to OBDD is possible, and does not imply any disadvantages as far as "standard" diagnostic tasks are considered.[51] This is also what DynamicHS aims to achieve: Enhancing HS-Tree for adoption in a sequential diagnosis setting while retaining the same guarantees (soundness, completeness, best-first property, general applicability).

In terms of the described properties 1–6 above, DynamicHS is (sound and) complete, (general) best-first, conflict-based (on-the-fly, centralized), black-box, holistic, and stateful. The algorithms most similar to DynamicHS as per these features, which have been reportedly used for sequential diagnosis according to the literature, are:[52]

- StaticHS (Rodler & Herold, 2018), which is sound, complete, (general) best-first, conflict-based (on-the-fly, centralized), black-box, holistic, and stateful: The aim of StaticHS is orthogonal to the one of DynamicHS, as the former focuses on reducing the time spent (by the user) for measurement conduction whereas the latter is geared to reduce the computation time. Moreover, DynamicHS admits more powerful tree pruning techniques than StaticHS (Rodler, 2015).

- HS-Tree (Reiter, 1987), which is sound, complete, (general) best-first, conflict-based (on-the-fly, centralized), black-box, holistic, and *stateless*: As extensively discussed, the only difference between HS-Tree and DynamicHS is that the latter maintains its state throughout sequential diagnosis whereas the former does not. As we show in our evaluations, the transition from statelessness (HS-Tree) to statefulness (DynamicHS) has a significant positive impact on the algorithm's runtime. And there is still potential for further improvement of DynamicHS, e.g., by exploiting the parallelization techniques proposed by (Jannach et al., 2016). Besides the adoption of such techniques in DynamicHS's tree expansion phase (as shown by (Jannach et al., 2016) for HS-Tree), they appear particularly appropriate for tree pruning (where nodes can be processed independently of one another) and for redundancy checking (where multiple independent reasoning operations can be necessary).

- RBF-HS and HBF-HS (Rodler, 2020d, 2020e), which are sound, complete, (general) best-first, conflict-based (on-the-fly, centralized), black-box, holistic, and *stateless*: These two algorithms are specifically geared towards scenarios where the soundness, completeness as well as the best-first property are important and where memory is an issue, such as on low-end (e.g., IoT) devices or when facing particularly challenging diagnostic problems (e.g., with high-cardinality minimal diagnoses). In case the diagnosis problem at hand is solvable using HS-Tree given the available memory, the latter—and even more so DynamicHS, as we shall demonstrate in Sec. 5—can be expected to be more time-efficient than RBF-HS and HBF-HS on average (Rodler, 2020d).

- GDE (de Kleer & Williams, 1987), which is sound, complete, (general) best-first, conflict-based (centralized), *glass-box*, holistic, and stateful: GDE has mainly been developed, optimized and used for the diagnosis of physical devices like circuits. The main difference to

[51] However, in other domains, such as system verification, tractable queries regarding OBDDs become intractable when using DNNF (Darwiche, 2001).

[52] Differences in properties as compared to DynamicHS are italicized.

DynamicHS is that reasoning strategies drawing on a truth maintenance system (TMS) are employed. These store all justifications (environments) for already computed entailments and appear[53] to be not the ideal approach for systems with a highly non-local structure and up to thousands of justifications per entailment (Horridge, 2011). Moreover, when there is a large variety of logics used to conceptualize the (system) knowledge (Baader et al., 2007), flexible black-box techniques, like HS-Tree and DynamicHS, that are (ad-hoc) usable for all these logics, appear particularly attractive.

- Inv-HS-Tree (Shchekotykhin et al., 2014), which is sound, complete, *any-first*, *direct*, black-box, holistic, and *stateless*: Due to its ability to use linear-search techniques, Inv-HS-Tree can solve problems for which HS-Tree runs out of memory (Shchekotykhin et al., 2014). This result remains valid for DynamicHS as its worst-case memory consumption is, in general, not lower than HS-Tree's. On the other hand, Inv-HS-Tree is any-first and can thus not be used for diagnosis tasks where a particular class of diagnoses (e.g., minimum-cardinality) or the most preferred diagnoses need to be found (first). Specifically, in the sequential diagnosis setting considered in this work, Inv-HS-Tree prevents the reasonable use of early termination strategies (e.g., by stopping the sequential diagnosis session if some diagnosis exceeds a given probability threshold), which can be practical and save significant effort, yet with a high probability of finding the actual diagnosis.

- SDA (Siddiqi & Huang, 2011), which is sound, *direct*, *glass-box*, *abstraction-based*, stateful:[54] SDA is a probabilistic hierarchical compilation-based method that eventually (at the end of sequential diagnosis) computes *one* (best) diagnosis. In particular, no set of diagnoses is computed at intermediate steps. This method is crucially different to DynamicHS in the following regards: First, while the suggested abstractions have been shown to be highly beneficial as far as circuits are concerned, it remains unclear and seems highly non-trivial to apply similar techniques to logics more expressive than propositional logic and systems that are structurally different from typical circuit topologies (e.g., with non-local interactions between components). Second, SDA assumes the set of observables (possible measurement points) to be explicitly given a-priori. However, this assumption is generally violated in diagnosis domains different from physical systems, e.g., in general knowledge bases, where measurement points (logical sentences that an oracle classifies as entailed or non-entailed) are mostly given in implicit form (i.e., derivable through reasoning) from the diagnosis problem (Rodler et al., 2017), and a sample of *multiple* precomputed diagnoses is often the only means for extracting reasonable measurement points (Rodler, 2015).

5. Evaluation

5.1 Objective

The goals of our evaluation are

- to compare the performances of DynamicHS and HS-Tree in real-world sequential diagnosis scenarios under varying diagnostic settings,

[53] See discussion above.

[54] Whether SDA is complete and best-first (in the sense discussed in this work) cannot be unambiguously assessed as it does not compute *a set of* diagnoses.

- to understand the reasons of performance differences between both algorithms under particular consideration of DynamicHS's advanced techniques discussed in Sec. 3.3,
- to demonstrate the out-of-the-box applicability of DynamicHS to diagnosis problems over different and highly expressive knowledge representation languages.

Importantly, the goal is *not* to show that DynamicHS is better than all or most diagnosis computation algorithms in literature, or that it is the best method in most or all diagnosis application domains, which is pointless (cf. Sec. 4). Rather, the intention is to show the advantage of preferring DynamicHS to HS-Tree in sequential diagnosis scenarios where HS-Tree is a state-of-the-art technique for diagnosis computation. In fact, HS-Tree is among the best available diagnosis computation techniques when a sound, complete, best-first, and reasoner-independent approach for arbitrary (and potentially highly expressive) logics is required.

One important domain where these requirements are essential is ontology and knowledge base[55] debugging. In this field, practitioners and experts usually[56], and especially in critical applications of ontologies such as medicine (Rector et al., 2011), want a debugger to output exactly the faulty axioms that really explain the observed faults in the ontology (*soundness* and *completeness*) at the end of a debugging session. In addition, experts often wish to perpetually monitor the most promising fault explanation throughout the debugging process (*best-first property*) with the intention to stop the session early if they recognize the fault. As was recently studied by (Rodler & Elichanova, 2020), the use of best-first algorithms often also involves efficiency gains in sequential diagnosis as opposed to strategies that enumerate diagnoses in a different order. Apart from that, it is a big advantage for users of knowledge-based or semantic systems to have a debugging solution that works out of the box for different logical languages and with different logical reasoners (*general applicability*, cf. *black-box* property in Sec. 4). This is because *(i)* ontologies are formulated in a myriad of different (Description) logics (Baader et al., 2007) with the aim to achieve the required expressivity for each ontology domain of interest at the least cost for inference, and *(ii)* highly specialized reasoners exist for different logics (cf., e.g., (Baader et al., 2005; Kazakov et al., 2014)), and being able to flexibly switch to the most efficient reasoner for a particular debugging problem can bring significant performance improvements.

Hence, we ran our experiments to compare DynamicHS with HS-Tree on a dataset of faulty real-world knowledge bases, for which HS-Tree is the method of choice according to literature, cf., e.g., (Kalyanpur, 2006; Schlobach et al., 2007; Horridge, 2011; Meilicke, 2011; Shchekotykhin et al., 2012; Rodler, 2015; Fu et al., 2016; Baader et al., 2018).

5.2 Dataset

The benchmark of 20 inconsistent or incoherent[57] real-world ontologies we used for our experiments is given in Tab. 4.[58] Parts of this dataset have been investigated i.a. in (Kalyanpur, 2006; Qi &

[55] We will use the terms *ontology* and *knowledge base* interchangeably throughout this section. For the purposes of this paper, we consider both to be finite sets of axioms expressed in some monotonic logic, cf. $\mathcal{K}$ in Sec. 2.

[56] The discussed requirements emerged from discussions with ontologists whom we interviewed (e.g., in the course of a tool demo at the International Conference on Biological Ontology 2018) in order to develop and customize our ontology debugging tool *OntoDebug* (Schekotihin, Rodler, & Schmid, 2018a; Schekotihin, Rodler, Schmid, Horridge, & Tudorache, 2018b) to match its users' needs.

[57] A knowledge base $\mathcal{K}$ is called *incoherent* iff it entails that some predicate p must always be *false*; formally: $\mathcal{K} \models \forall \mathbf{X} \neg p(\mathbf{X})$ where p is a predicate with arity k and $\mathbf{X}$ a tuple of k variables. With regard to ontologies, incoherence means that some class (unary predicate) must not have any instances, and if so, the ontology becomes inconsistent.

[58] The benchmark problems can be downloaded from http://isbi.aau.at/ontodebug/evaluation.

Table 4: Dataset used in the experiments (sorted by the number of components/axioms of the diagnosis problem, 2nd column).

KB $\mathcal{K}$	$\lvert\mathcal{K}\rvert$	expressivity [1)]	#D/min/max [2)]
Koala (K)	42	$\mathcal{ALCON}^{(D)}$	10/1/3
University (U)	50	$\mathcal{SOIN}^{(D)}$	90/3/4
IT	140	$\mathcal{SROIQ}$	1045/3/7
UNI	142	$\mathcal{SROIQ}$	1296/5/6
Chemical (Ch)	144	$\mathcal{ALCHF}^{(D)}$	6/1/3
MiniTambis (M)	173	$\mathcal{ALCN}$	48/3/3
falcon-crs-sigkdd (fal-cs)	195	$\mathcal{ALCIF}^{(D)}$	5/1/2
coma-conftool-sigkdd (com-cs)	338	$\mathcal{SIN}^{(D)}$	570/3/10
ctxmatch-sigkdd-ekaw (ctx-se)	367	$\mathcal{SHIN}^{(D)}$	26/1/5
hmatch-cmt-conftool (hma-cmc)	434	$\mathcal{SIN}^{(D)}$	43/2/7
hmatch-conftool-cmt (hma-coc)	436	$\mathcal{SIN}^{(D)}$	41/3/8
coma-cmt-conftool (com-cc)	442	$\mathcal{SIN}^{(D)}$	905/2/8
ctxmatch-cmt-conftool (ctx-cc)	458	$\mathcal{SIN}^{(D)}$	934*/2/16*
falcon-conftool-ekaw (fal-ce)	465	$\mathcal{SHIN}^{(D)}$	201/2/7
Transportation (T)	1300	$\mathcal{ALCH}^{(D)}$	1782/6/9
Economy (E)	1781	$\mathcal{ALCH}^{(D)}$	864/4/8
DBpedia (D)	7228	$\mathcal{ALCHF}^{(D)}$	7/1/1
Opengalen (O)	9664	$\mathcal{ALEHIF}^{(D)}$	110/2/6
CigaretteSmokeExposure (Cig)	26548	$\mathcal{SI}^{(D)}$	1566*/4/7*
Cton (C)	33203	$\mathcal{SHF}$	15/1/5

1): Description Logic expressivity: each calligraphic letter stands for a (set of) logical operator(s) that are allowed in the respective language, e.g., $\mathcal{C}$ denotes negation ("complement") of concepts, for details see (Baader et al., 2007; Zolin,); roughly, the more letters, the higher the expressivity of a logic and the complexity of reasoning for this logic tends to be.

2): #D/min/max denotes the number/the minimal size/the maximal size of minimal diagnoses for the initial DPI dpi_0 resulting from each input KB $\mathcal{K}$. If tagged with a *, a value signifies the number or size determined within 1200sec using HS-Tree (for problems where the finding of *all* minimal diagnoses was impossible within reasonable time).

Hunter, 2007; Horridge et al., 2008; Stuckenschmidt, 2008; del Vescovo, Parsia, Sattler, & Schneider, 2010; Shchekotykhin et al., 2012; Rodler et al., 2013; Jannach et al., 2016; Rodler et al., 2019). For all ontologies whose name (Tab. 4, column 1) starts with an upper case letter, their faultiness is a result of (only) modeling errors of humans developing the ontology. The problems in ontologies termed by lowercase names, in contrast, result from automatically merging two human-modeled ontologies which describe the same domain in different ways. This merging process, which involves a matching system integrating two source ontologies by adding additional (potentially faulty) axioms stating relationships between concepts from both source ontologies, is called *ontology alignment* (Euzenat et al., 2011) and can lead to numerous independent faults in the resulting merged ontology. In our dataset, the domain described by the merged ontologies is "conference organization" (describing concepts such a papers, authors, program committee members, reviewers, etc.) and the faulty merged ontologies are results produced by four different matching systems (Euzenat, Mochól, Shvaiko, Stuckenschmidt, Sváb, Svátek, van Hage, & Yatskevich, 2006). The names of the respective ontologies in Tab. 4 follow the scheme $\mathsf{name1} - \mathsf{name2} - \mathsf{name3}$ where $\mathsf{name1}$ corresponds to the name of the used matching system, and $\mathsf{name2}$ as well as $\mathsf{name3}$, respectively, to the name of the first and second source ontology to be merged.

As Tab. 4 shows, the ontologies in the dataset cover a spectrum of different problem sizes (number of axioms or components; column 2), logical expressivities (which determine the complexity of consistency checking; column 3), as well as diagnostic structures (number and size of minimal diagnoses; column 4). Note that the complexity of consistency checks over the logics in Tab. 4 ranges from EXPTIME-complete to 2-NEXPTIME-complete (Grau et al., 2008b; Zolin,). Hence, from the point of view of model-based reasoning, ontology debugging problems represent a particularly challenging diagnosis domain as they usually deal with harder logics than more traditional diagnosis problems (which often use propositional knowledge representation languages that are not beyond NP-complete).

5.3 Experiment Settings

To study the performance and robustness of DynamicHS under varying circumstances, we considered a range of different *diagnosis scenarios* in our experiments. A diagnosis scenario is defined by the set of inputs given to Alg. 1, i.e., by an initial DPI dpi_0, a number ld of minimal diagnoses to be computed per sequential diagnosis iteration, a probability measure pr assigning fault probabilities to system components (axioms in the knowledge base), as well as a heuristic heur used for measurement selection. The DPIs dpi_0 for our tests were defined as $\langle \mathcal{K}, \emptyset, \emptyset, \emptyset \rangle$, one for each $\mathcal{K}$ in Tab. 4. That is, the task was to find a minimal set of axioms (faulty components) responsible for the inconsistency or incoherency of $\mathcal{K}$, without any background knowledge or measurements initially given. For the parameter ld we used the values $\{2, 4, 6, 10, 20, 30\}$. The fault probability $pr(ax)$ of each axiom (component) $ax \in \mathcal{K}$ was chosen uniformly at random from $(0, 1)$. As measurement selection heuristics, we used $\{\text{ENT}, \text{SPL}, \text{MPS}\}$, where ENT is the well-known entropy function proposed by (de Kleer & Williams, 1987), SPL is the "split-in-half" strategy (Moret, 1982; Shchekotykhin et al., 2012), and MPS ("most probable singleton") (Rodler, 2016, 2018) is the selection measure that overall performed most favorably in the evaluations conducted by (Rodler & Schmid, 2018). Roughly speaking, given a set of minimal diagnoses, ENT / SPL / MPS selects a discriminating measurement point (cf. Sec. 2.5) which maximizes the expected information gain / maximizes the worst-case diagnosis elimination rate / maximizes the probability of the elimination of a maximal number of diagnoses. For each diagnosis scenario, we randomly selected 20 different minimal diagnoses, each of which was used as the target solution (actual diagnosis; cf. Sec. 2.2) in one sequential diagnosis session (execution of Alg. 1). Finally, we executed DynamicHS (Alg. 1 with setting $dynamic = true$) and HS-Tree (setting $dynamic = false$) to find the same 20 target solutions for each diagnosis scenario. As a Description Logic reasoner, we adopted Pellet (Sirin, Parsia, Grau, Kalyanpur, & Katz, 2007), and we implemented the FINDMINCONFLICT function by means of the QuickXPlain algorithm (Junker, 2004; Rodler, 2020f).

5.4 Experiment Results

We first analyze the performance of DynamicHS compared to HS-Tree in Sec. 5.4.1, and then, in Sec. 5.4.2, explain the findings and give additional insights by examining the impact of the advanced techniques incorporated into DynamicHS discussed in Sec. 3.3.[59]

[59]The raw data obtained from our experiments can be downloaded from http://isbi.aau.at/ontodebug/evaluation. Moreover, our implementation of DynamicHS can be accessed under https://bit.ly/348Ny8f.

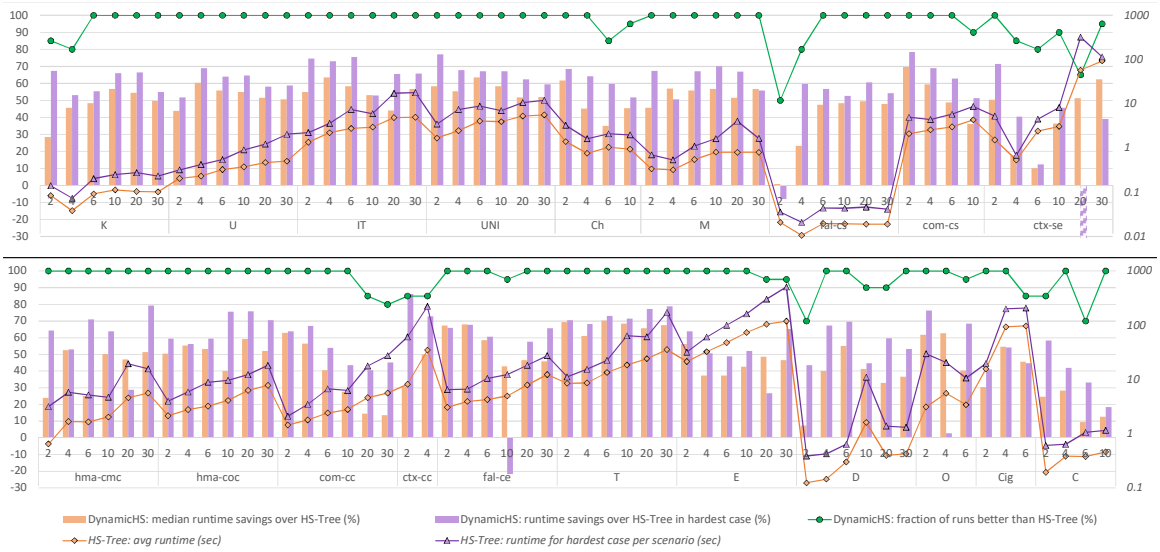


Figure 6: Experiment results (DynamicHS vs. HS-Tree) for measurement selection heuristic ENT: x-axis shows faulty ontologies $\mathcal{K}$ from Table 4 and number ld of leading diagnoses computed per call of hitting set algorithm (iteration of while-loop in Alg. 1). For each ($\mathcal{K}$,ld) scenario, *(left y-axis)* the orange bar shows the median % runtime savings (over 20 sequential diagnosis sessions) achieved by DynamicHS, the violet bar indicates the % runtime savings exhibited by DynamicHS for the hardest case of the scenario, and the green dot shows the fraction (in %) of the 20 runs for the scenario where DynamicHS manifested lower runtime than HS-Tree; *(right y-axis)* the orange diamond / violet triangle depicts the average / maximal runtime (in sec) of HS-Tree over the 20 sequential sessions. In the legend, features written in normal / italic font are plotted wrt. the left / right y-axis. The striped violet bar is not fully shown for clarity of the figure: In this case, HS-Tree required 86 % less time than DynamicHS.

5.4.1 PERFORMANCE ANALYSIS

The comparison of runtime performance between HS-Tree and DynamicHS over the 20 sequential diagnosis sessions for each diagnosis scenario is shown in Fig. 6 for the heuristic ENT, in Fig. 7 for the heuristic SPL, and in Fig. 8 for the heuristic MPS.[60] As the orange bars in these three figures testify, DynamicHS exhibits substantial median runtime savings over HS-Tree in almost all considered scenarios and for all three heuristics ENT, SPL and MPS.[61] Only in two (of the overall

[60] For some knowledge bases in Tab. 4, the figures do not include data for all settings of ld. The reason is that the experiment runs for these knowledge bases terminated with errors (for both algorithms). More specifically, the cause of the error was either an internal reasoner exception (recall that both algorithms use the reasoner as a black-box oracle, which is why this error is outside the range of influence of the algorithms) or an out-of-memory error. Regarding the latter, it is important to note that both algorithms give strong guarantees in terms of soundness, completeness and the best-first property, which requires them to keep track of all possible branches of the search tree, which in turn can become problematic if minimal diagnoses of high cardinality exist and tree depth is high (Shchekotykhin et al., 2014). Please note that the diagnostic structure (fourth column of Tab. 4) indicates only the sizes of minimal diagnoses for the *initial* DPI dpi_0 and that minimal diagnoses computed throughout a sequential diagnosis session grow in size (cf. Property 1.2) and can reach cardinalities that exceed the quoted initial values substantially.

[61] The computation time reduction of DynamicHS over HS-Tree was statistically significant wrt. the level $\alpha := 0.05$ in almost all (97 %) of the diagnosis scenarios, as verified by Wilcoxon Signed Rank Tests. More specifically, statistical significance could be ascertained in all scenarios except for the following: ctx-se for ENT 20 as well as SPL 10 and 20; fal-cs for ENT 2; com-cc for MPS 10 and 20; and D for MPS 10, 20, 30 as well as for SPL 20.

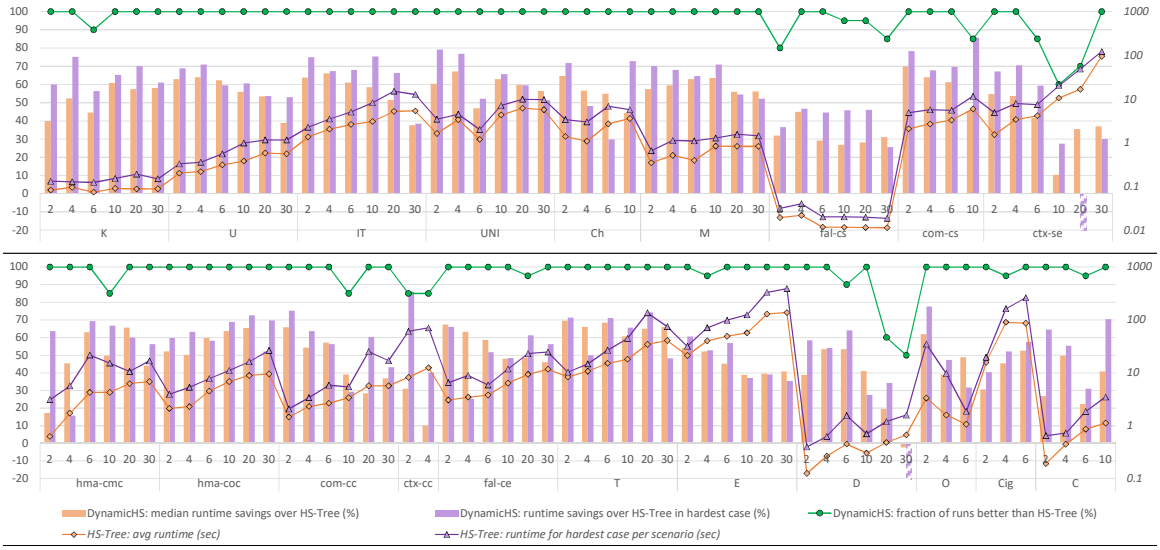


Figure 7: Experiment results (DynamicHS vs. HS-Tree) for measurement selection heuristic SPL: x-axis shows faulty ontologies $\mathcal{K}$ from Table 4 and number ld of leading diagnoses computed per call of hitting set algorithm (iteration of while-loop in Alg. 1). For each ($\mathcal{K}$,ld) scenario, *(left y-axis)* the orange bar shows the median % runtime savings (over 20 sequential diagnosis sessions) achieved by DynamicHS, the violet bar indicates the % runtime savings exhibited by DynamicHS for the hardest case of the scenario, and the green dot shows the fraction (in %) of the 20 runs for the scenario where DynamicHS manifested lower runtime than HS-Tree; *(right y-axis)* the orange diamond / violet triangle depicts the average / maximal runtime (in sec) of HS-Tree over the 20 sequential sessions. In the legend, features written in normal / italic font are plotted wrt. the left / right y-axis. The striped violet bars are not fully shown for clarity of the figure: In these cases, HS-Tree required 76 % (ctx-se, 20) and 32 % (D, 30) less time than DynamicHS.

312) scenarios slight losses against HS-Tree in terms of median runtime could be registered, namely -2 % for (SPL, D, 30) and -3 % for (MPS, fal-ce, 30). However, in terms of absolute runtime, DynamicHS never required more than 0.8 sec more time than HS-Tree in any of the 20 runs in the former scenario, and never more that 12 sec more in the latter.

The left boxplot in Fig. 9 depicts the overall distribution of the median savings (orange bars in Figs. 6–8) for the three heuristics. As we can see from the boxplot, median savings reach maximal values of 70 % / 70 % / 75 % for ENT / SPL / MPS, and amount to median values of 50 % / 52 % / 52 %. Moreover, the observed median savings do not differ in a statistically significant way between the three heuristics.[62] Also, the data does not show a general tendency that the percentage of runtime reduction depends on the number of minimal diagnoses ld computed per sequential diagnosis iteration;[63] see the boxplots in Fig. 10. For the values 2 / 4 / 6 / 10 / 20 / 30 of ld, over all scenarios, we observe median runtime savings of 49 % / 51 % / 49 % / 45 % / 46 % / 49 % for ENT, of 45 % / 50 % / 45 % / 45 % / 48 % / 49 % for SPL, and of 49 % / 51 % / 49 % / 45 % / 46 % / 49 % for MPS.

Considering the fraction of the 20 runs per diagnosis scenario where DynamicHS outperformed HS-Tree (green circles in Figs. 6–8 and right boxplot in Fig. 9), we observe values equal or very close to 100 % in almost all scenarios. That is, sequential diagnosis sessions where HS-Tree is

[62] We verified this by an ANOVA (p-value: 0.26).

[63] We checked this by a Kruskal-Wallis test, one for each of the three heuristics. The p-values are 0.81 (ENT), 0.22 (SPL), and 0.81 (MPS).

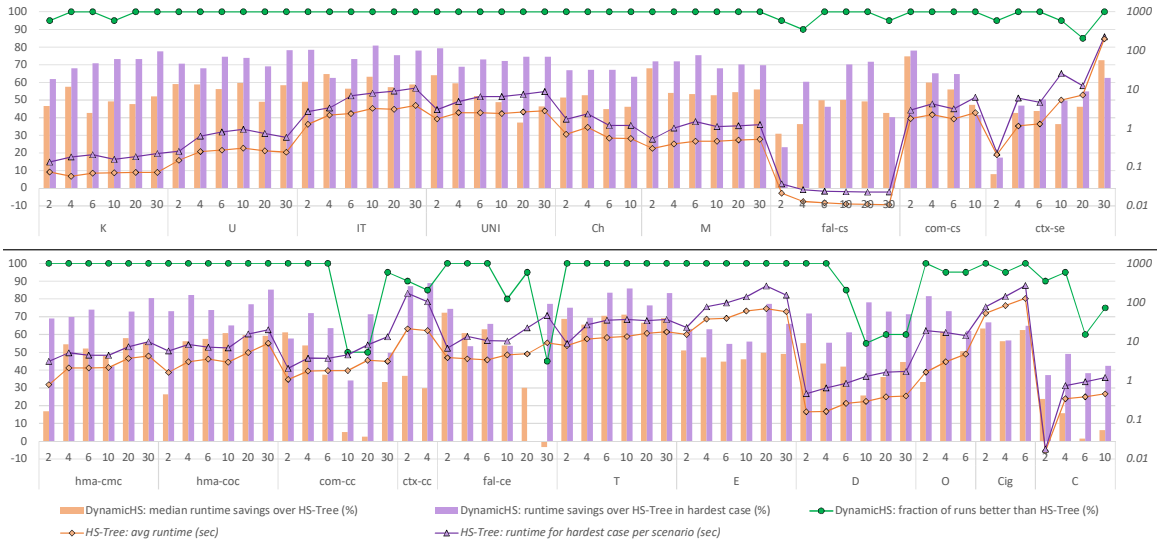


Figure 8: Experiment results (DynamicHS vs. HS-Tree) for measurement selection heuristic MPS: x-axis shows faulty ontologies $\mathcal{K}$ from Table 4 and number ld of leading diagnoses computed per call of hitting set algorithm (iteration of while-loop in Alg. 1). For each ($\mathcal{K}$,ld) scenario, *(left y-axis)* the orange bar shows the median % runtime savings (over 20 sequential diagnosis sessions) achieved by DynamicHS, the violet bar indicates the % runtime savings exhibited by DynamicHS for the hardest case of the scenario, and the green dot shows the fraction (in %) of the 20 runs for the scenario where DynamicHS manifested lower runtime than HS-Tree; *(right y-axis)* the orange diamond / violet triangle depicts the average / maximal runtime (in sec) of HS-Tree over the 20 sequential sessions. In the legend, features written in normal / italic font are plotted wrt. the left / right y-axis.

preferable to DynamicHS in terms of runtime are very rare, regardless of the adopted measurement selection strategy. In terms of numbers, DynamicHS requires less runtime than HS-Tree in 96 % (all data), 96 % (ENT), 97 % (SPL), and 95 % (MPS) of the scenarios.

Finally, since the hardest sequential diagnosis session in terms of the required time for diagnosis computation was up to one order of magnitude harder than the average session of one and the same diagnosis scenario (cf. orange diamonds and violet triangles in Figs. 6–8), we also included the savings of DynamicHS over HS-Tree for these hardest cases in the plots of Figs. 6–8 (see the violet bars) and summarized their distribution in the middle boxplot of Fig. 9. As readable from the boxplot, savings in these hardest cases reach maximal values up to 86 % / 86 % / 89 % for ENT / SPL / MPS, and amount to median values of 60 % / 60 % / 70 %. In fact, the observed savings even exceed the median savings (orange bars in Figs. 6–8) in 83 % (all data), 86 % (ENT), 75 % (SPL), and 87 % (MPS) of the diagnosis scenarios (compare the left and middle boxplots in Fig. 9).This indicates that savings achieved by DynamicHS versus HS-Tree tend to be above-average when diagnosis computation takes long. Tab. 5 enumerates for some of the per-scenario hardest cases the runtimes (rounded to full seconds) of both algorithms as well as the percental savings achieved by DynamicHS over HS-Tree. For instance, we see from the table that a runtime reduction of more than five and a half minutes could be obtained by DynamicHS for (ENT, E, 30), or that HS-Tree with (more than) 1:44 minutes computation time consumed more than nine times as much time as DynamicHS (less than 12 seconds) for the hardest case of scenario (MPS, ctx-cc, 4).

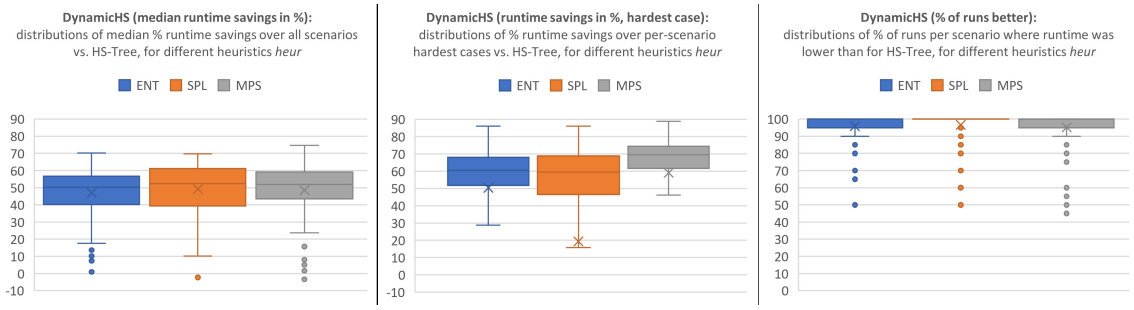


Figure 9: Boxplots showing the distribution of data given in Figs. 6–8. The left / middle / right plot relates to the orange bars / violet bars / green circles in Figs. 6–8, and the blue / orange / gray boxplots refer to Fig. 6 / Fig. 7 / Fig. 8.

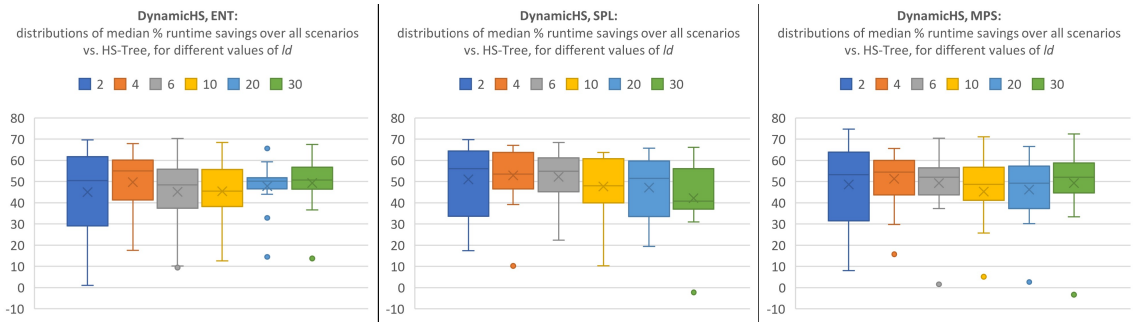


Figure 10: Boxplots showing the distributions of data (orange bars) displayed by Figs. 6–8 for different values of ld: Left / middle / right plot relates to Fig. 6 / Fig. 7 / Fig. 8.

Table 5: Runtimes of both algorithms and savings achieved by DynamicHS for five of the per-scenario hardest cases for each of the measurement selection heuristics ENT, SPL and MPS. The leftmost column lists scenarios in the form "heur, $\mathcal{K}$, ld" where heur is the heuristic, $\mathcal{K}$ the knowledge base from Tab. 4, and ld the number of leading diagnoses computed per sequential diagnosis iteration.

scenario	runtime HS-Tree (min)	runtime DynamicHS (min)	savings of DynamicHS (%)
ENT, E, 30	8:30	2:58	65
ENT, ctx-cc, 4	3:43	1:01	73
ENT, T, 30	2:52	0:36	79
ENT, ctx-cc, 2	1:00	0:08	86
ENT, O, 2	0:29	0:07	76
SPL, Cig, 6	4:22	1:51	58
SPL, T, 20	2:16	0:35	74
SPL, com-cc, 30	1:00	0:08	86
SPL, T, 10	0:44	0:15	66
SPL, O, 2	0:34	0:08	77
MPS, E, 20	4:23	1:00	77
MPS, ctx-cc, 2	2:51	0:22	87
MPS, ctx-cc, 4	1:44	0:12	89
MPS, T, 10	0:36	0:05	86
MPS, hma-coc, 30	0:20	0:03	85

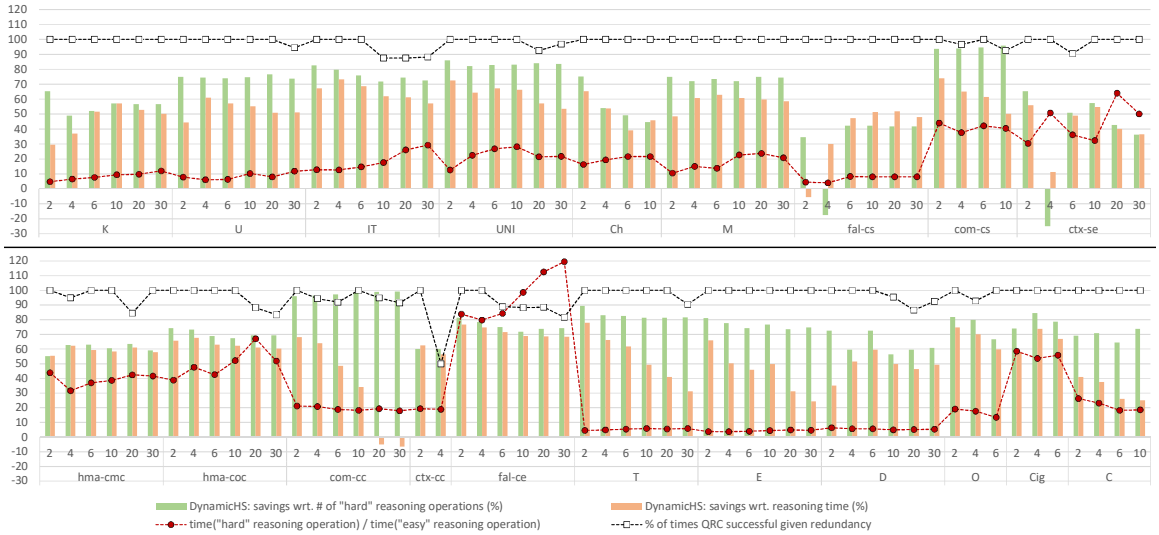


Figure 11: Analysis of DynamicHS wrt. avoidance of reasoning and efficient redundancy checking: x-axis shows faulty ontologies $\mathcal{K}$ from Table 4 and number ld of leading diagnoses computed per call of hitting set algorithm (iteration of while-loop in Alg. 1). The plot shows data for the heuristic ENT. All values depicted are medians over the 20 sequential diagnosis sessions executed per diagnosis scenario. See Sec. 3.3.3 for a definition of "hard" and "easy" reasoning operations; "QRC" refers to quick redundancy check, see Sec. 3.3.1; "savings" refers to the savings over HS-Tree.

5.4.2 Analysis of Advanced Techniques

As the performance comparison between DynamicHS and HS-Tree at the end of Example 5 already suggested, there are two major sources for the runtime savings obtained by means of DynamicHS: *(I)* fewer expensive reasoning operations and *(II)* saved effort for tree (re)construction. The price to pay for these reductions is *(III)* the storage of the existing hitting set tree (including duplicate nodes) throughout the diagnosis session and *(IV)* the execution of regular tree pruning actions. We examine these four aspects more closely in the next three paragraphs based on our experiment results. The first paragraph addresses aspect (I), the second analyzes aspects (II) and (III), and the third focuses on aspect (IV).

Avoidance of Expensive Reasoning and Efficient Redundancy Checking: Fig. 11 summarizes several statistics that illustrate aspect (I) in more detail. The reduction of costly reasoning operations ("hard" FINDMINCONFLICT calls, cf. Sec. 3.3.3) is achieved by trading them against more efficient reasoning operations ("easy" FINDMINCONFLICT calls, cf. Sec. 3.3.3). The red circles in the figure, which show how much harder an average "hard" call is than an average "easy" one in each diagnosis scenario, attest that this strategy of swapping "hard" for "easy" calls is indeed plausible, as the former are a median of 19 times and up to 120 times as time-consuming as the latter.

The green bars in Fig. 11 reveal that significant savings in terms of "hard" calls are indeed consistently generated by DynamicHS. More specifically, in more than 98 % of the diagnosis scenarios, the median relative savings wrt. "hard" FINDMINCONFLICT calls in comparison to HS-Tree are higher than 30 %; over all scenarios, median and maximal savings amount to 74 % and 99 %, respectively.

Rather unsurprisingly, these savings wrt. the number of "hard" FINDMINCONFLICT calls translate to a similarly substantial reduction of runtime spent for reasoning operations (orange bars in Fig. 11) manifested by DynamicHS. Savings in terms of the runtime dedicated to reasoning are achieved in 97 % of the diagnosis scenarios and amount to median and maximal values of 57 % and 78 %, respectively. The main reason why the diminution in terms reasoning time is most times lower than the decrease of "hard" reasoning operations is that DynamicHS, as opposed to HS-Tree, makes use of "easy" reasoner calls to compensate for these saved more expensive calls. That is, these "easy" calls account to a large extent for the difference between green and orange bars. As evident in some scenarios, e.g., for the knowledge base fal-cs, however, it is also possible that savings in reasoning time top savings in terms of "hard" FINDMINCONFLICT calls. This can happen, e.g., when DynamicHS saves many "medium" FINDMINCONFLICT calls against HS-Tree (cf. Example 7 and Tab. 3) in addition to its savings wrt. "hard" ones.

Last but not least, also the used efficient redundancy checking technique incorporated in DynamicHS contributes to the achieved runtime savings. In fact, the adoption of the QRC (quick redundancy check, cf. Sec. 3.3.1) attempts to minimize the ("easy") reasoning operations necessary to decide the redundancy of a particular node (wrt. a particular given minimal conflict). The white boxes in Fig. 11 indicate that this strategy is very powerful in that the QRC detects redundancy almost always when redundancy is actually given, thus allowing the alorithm to skip the more expensive CRC (complete redundancy check, cf. Sec. 3.3.1). One could say that the QRC appears to be "almost complete" in our experiments. In numbers, we observe that the QRC detected *all* redundancies in 75 %, at least nine of ten redundancies in 89 %, and at least eight of ten redundancies in 99 % of the scenarios.

Statefulness (DynamicHS) vs. Statelessness (HS-Tree): Let us now consider Fig. 12, which allows to investigate aspects (II) and (III). The figure depicts, per diagnosis scenario, the median factors how many more nodes HS-Tree had to process than DynamicHS (grey bars) and how many more nodes DynamicHS needed to store than HS-Tree (yellow bars). The fact that the majority of all bars attains values greater than 1 demonstrates that DynamicHS—expectedly—tends to trade less time for more space. It keeps the produced search tree in memory and utilizes the information contained in this tree to allow for more efficient diagnosis computation in the next sequential diagnosis iteration. We can identify from the figure that in two thirds of the diagnosis scenarios, the trade-off achieved by DynamicHS is favorable in the sense that the factor of more memory used by DynamicHS is less than the factor of more nodes processed by HS-Tree (yellow bars smaller than grey ones).

Moreover, Fig. 12 (yellow bars) reveals that DynamicHS required less memory than HS-Tree in 14 % of the scenarios, exhibited less than 25 % memory overhead in 35 % and less than 50 % overhead in 52 % of the scenarios, and consumed less than twice the memory of HS-Tree in 82 % of the diagnosis scenarios. Circumstances where DynamicHS can require even less memory than HS-Tree are when few or no duplicate nodes exist (e.g., when minimal conflicts are mostly disjoint), when DynamicHS's hitting set tree after tree updates is (largely) equal to the one produced by HS-Tree (cf. Sec. 3.3.2), and when DynamicHS happens to compute or select for reuse more "favorable" conflicts than HS-Tree in the course of node labeling.[64] In less than 10 % of the sce-

[64]The order in which conflicts are computed and selected for node labeling is neither controlled by HS-Tree nor by DynamicHS. If one of the algorithms happens to compute smaller minimal conflicts that are used to label nodes at the top of the hitting set tree and when the hitting set trees produced by both algorithms are not computed to their entirety (as is the case when ld diagnoses have been computed before the tree is complete), then the tree with the smaller conflicts at the top can be smaller than the one generated by the other algorithm.

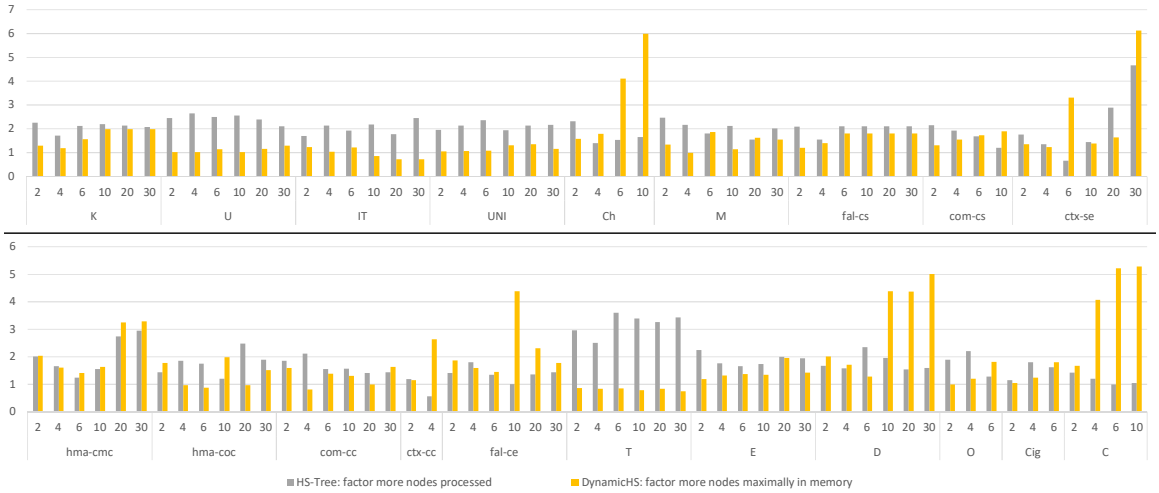


Figure 12: Comparison of stateful (DynamicHS) versus stateless (HS-Tree) hitting set tree management in terms of the nodes processed (time efficiency) and the nodes maximally in memory (space efficiency): x-axis shows faulty ontologies $\mathcal{K}$ from Table 4 and number ld of leading diagnoses computed per call of hitting set algorithm (iteration of while-loop in Alg. 1). The plot shows data for the heuristic ENT. All values depicted are medians over the 20 sequential diagnosis sessions executed per diagnosis scenario.

narios, however, the memory overhead shown by DynamicHS was substantial, reaching values of 4 or more times the amount of memory consumed by HS-Tree. Hence, although not observed in our evaluation where either both or none of the algorithms ran out of memory, there may be cases where DynamicHS is not applicable due to too little available memory while HS-Tree is. In general, this can be remedied by adding a mechanism to DynamicHS which simply discards the entire stored hitting set tree and starts over from scratch building a new tree whenever a certain fraction of the available memory has been exhausted. In other words, the stateful (DynamicHS) strategy can be flexibly and straightforwardly switched to a stateless (HS-Tree) one.

Fig. 12 (grey bars) also shows the benefit of the stateful strategy pursued by DynamicHS. That is, the overhead in terms of the time expended for tree (re)construction incurred when using HS-Tree instead of DynamicHS is significant in the majority of scenarios. In numbers, HS-Tree had to process at least 1.5 times as many nodes as DynamicHS in 78 % of the scenarios, at least twice as many in 42 %, and at least three times as many in 5 % of the diagnosis scenarios.

Finally, note that aspects (I) and (II), i.e., DynamicHS's reduction of time for reasoning and its savings in terms of tree construction costs, are orthogonal in the following sense: Even if one of these aspects turns out to be unfavorable from the viewpoint of DynamicHS in comparison to HS-Tree, the other aspect is not affected by that in general. For instance, in our experiments we observed three scenarios (fal-cs, 2; com-cc, 20; com-cc, 30) where DynamicHS actually required slightly more time for reasoning than HS-Tree (cf. orange bars in Fig. 11), i.e., aspect (I) was unfavorable for DynamicHS. Since DynamicHS however did save node processing time in these cases, i.e., aspect (II) was favorable, the overall computation time was still lower for DynamicHS. Similarly, there are cases (cf., e.g., ctx-cc, 4) where aspect (II) is unfavorable while aspect (I) is favorable, again resulting in overall time savings of DynamicHS.

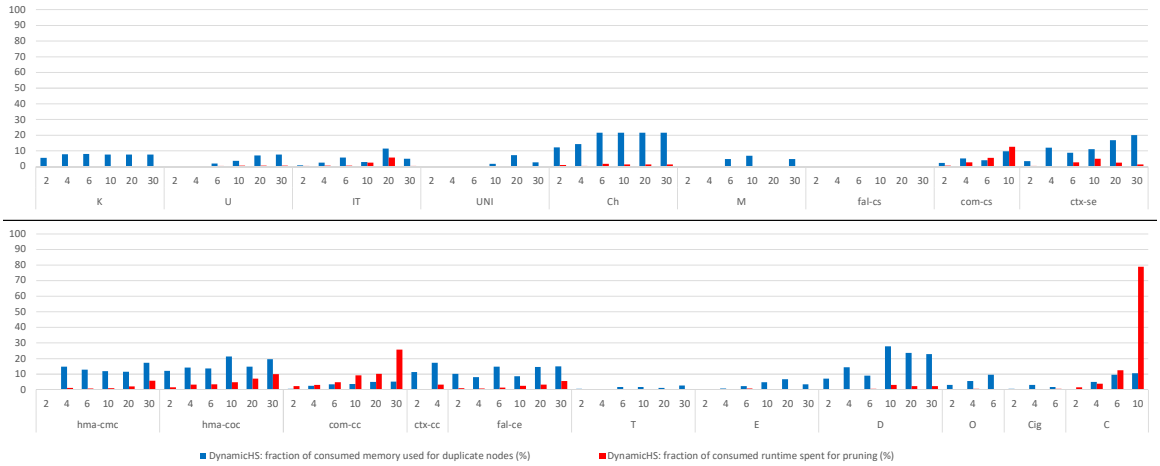


Figure 13: Analysis of DynamicHS wrt. space-efficiency of duplicate storage and time-efficiency of pruning actions: x-axis shows faulty ontologies $\mathcal{K}$ from Table 4 and number ld of leading diagnoses computed per call of hitting set algorithm (iteration of while-loop in Alg. 1). The plot shows data for the heuristic ENT. All values depicted are medians over the 20 sequential diagnosis sessions executed per diagnosis scenario.

Duplicates and Pruning: Fig. 13 illuminates aspect (IV). It displays the median fraction of the memory consumed by DynamicHS that is used for the storage of duplicate nodes per diagnosis scenario (blue bars). We can see that duplicate nodes account for less than 10 % of the used memory in 69 % of the diagnosis scenarios, for less than 20 % in 92 %, and for less than 30 % in 100 % of the scenarios. In other words, the memory overhead caused by duplicate nodes does not exceed 12 % / 25 % / 39 % in 69 % / 92 % / 100 % of the scenarios. Consequently, the additional memory consumption attributable to duplicates stays within acceptable bounds in all cases. This can be due to not too many existing duplicates (e.g., if minimal conflicts tend to be disjoint) or due to DynamicHS's memory-efficient storage of duplicates (cf. Sec. 3.3.4). The latter requires duplicates to be reconstructed on demand in the course of pruning and associated node replacement actions (cf. Sec. 3.3.4 and Example 8). The relative computational expense of these pruning actions in comparison to the overall computation time of DynamicHS is described by the red bars in Fig. 13. These tell that the relative computation time spent for pruning is negligible (less than 1 %) in 61 % of the diagnosis scenarios, marginal (less than 5 %) in 89 %, and small (less than 10 %) in 95 % of the scenarios. This overall fairly low overhead for pruning, which already includes the time for reconstructing duplicate nodes, testifies that DynamicHS's duplicate storage and reconstruction as well as its pruning techniques that completely dispense with costly reasoner calls are reasonable strategies for dealing with the statefulness of the hitting set tree.

Only in two cases, the time for pruning exceeds 20 % of DynamicHS's total computation time; in one of these cases (C, 10) it is very high, reaching almost 80 %, a possible sign that significant portions of the tree have become out-of-date through lazy updating (cf. Sec. 3.3.2). Note, however, that DynamicHS overall still saves time compared to HS-Tree in this scenario (cf. Fig. 6) since the time spent later for more intensive pruning is counterbalanced with time saved earlier by skipping pruning actions (lazy updating).

6. Conclusion

In this work we proposed DynamicHS, an optimization of Reiter's seminal HS-Tree algorithm geared towards sequential diagnosis use cases. Since, in sequential diagnosis, the addressed diagnosis problem is subject to successive change in terms of newly acquired system information in the form of observations or measurements, the main rationale behind DynamicHS is the adoption of a reuse-and-adapt principle, which maintains the already produced diagnosis search data structure and appropriately updates it whenever the diagnosis problem is modified. The goal of this stateful diagnostic search is to avoid expensive redundant operations involved in the reconstruction of discarded search data structures when using (the stateless) HS-Tree. Comprehensive experiments on a benchmark of 20 real-world diagnosis problems, from which we generated more than 6000 different sequential diagnosis problems under different settings as regards the number of diagnoses computed and the measurement selection method used, revealed that *(1)* DynamicHS leads to a (statistically significantly) better time performance than HS-Tree in almost all cases, *(2)* the time saved when using DynamicHS instead of HS-Tree is substantial in almost all cases and attains median and maximal values of 52 % and 89 %, respectively, *(3)* the memory overhead (due to the statefulness) of DynamicHS was reasonable in the majority of cases and DynamicHS was successfully applicable whenever HS-Tree was, and *(4)* the performance of DynamicHS compared to HS-Tree was particularly favorable in the hardest cases where the use of DynamicHS rather than HS-Tree could avoid time overheads of up to more than 800 %. Notably, DynamicHS achieves these performance improvements while preserving all desirable properties of HS-Tree. That is, like HS-Tree, DynamicHS is sound and complete, computes diagnoses in best-first order, and is generally applicable in that it is independent of the diagnosis domain, of the diagnosis problem structure, of the (monotonic) logic used to describe the diagnosed system, and of the adopted (sound and complete) logical inference engine.

Acknowledgments

I thank Manuel Herold for the excellent Java implementation of the DynamicHS algorithm and of the experiments. I am also grateful to Wolfgang Schmid for coordinating the execution of our experiments on our servers. This work was supported by the Austrian Science Fund (FWF), contract P-32445-N38.

References

Abreu, R., & van Gemund, A. J. (2009). A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis. In *Symp. on Abstraction, Reformulation, and Approximation*.

Abreu, R., Zoeteweij, P., & van Gemund, A. J. (2007). On the accuracy of spectrum-based fault localization. In *Testing: Academic and Industrial Conf. Practice and Research Techniques*.

Abreu, R., Zoeteweij, P., & van Gemund, A. J. (2011). Simultaneous debugging of software faults. *Journal of Systems and Software*, *84*(4), 573–586.

Baader, F., Brandt, S., & Lutz, C. (2005). Pushing the EL envelope. In *Int'l Joint Conf. on Artificial Intelligence (IJCAI)*.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F. (Eds.). (2007). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Baader, F., Kriegel, F., Nuradiansyah, A., & Penaloza, R. (2018). Repairing Description Logic Ontologies by Weakening Axioms. arXiv:1808.00248v1

Baader, F., Lutz, C., & Suntisrivaraporn, B. (2005). Is tractable reasoning in extensions of the description logic EL useful in practice. In *Int'l Workshop on Methods for Modalities (M4M)*.

Baader, F., & Penaloza, R. (2008). Axiom pinpointing in general tableaux. *Journal of Logic and Computation*, *20*(1), 5–34.

Bylander, T., Allemang, D., Tanner, M., & Josephson, J. (1991). The computational complexity of abduction. *Artificial Intelligence*, *49*, 25–60.

Chen, M., Quan Hu, L., & Tang, H. (2015). An approach for optimal measurements selection on gas turbine engine fault diagnosis. *Journal of Engineering for Gas Turbines and Power*, *137*(7).

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Annual ACM Symp. on Theory of Computing*.

Darwiche, A. (2001). Decomposable negation normal form. *Journal of the ACM (JACM)*, *48*(4), 608–647.

de Kleer, J. (1986). An assumption-based TMS. *Artificial intelligence*, *28*(2), 127–162.

de Kleer, J. (1990). Compiling devices and processes. In *Int'l Workshop on Qualitative Physics*.

de Kleer, J. (1991). Focusing on probable diagnoses. In *AAAI Conf. on Artificial Intelligence (AAAI)*.

de Kleer, J. (2008). An improved approach for generating max-fault min-cardinality diagnoses. In *Int'l Workshop on Principles of Diagnosis (DX)*.

de Kleer, J. (2009). Mininimum cardinality candidate generation. In *Int'l Workshop on Principles of Diagnosis (DX)*.

de Kleer, J. (2011). Hitting set algorithms for model-based diagnosis. In *Int'l Workshop on Principles of Diagnosis (DX)*.

de Kleer, J., Mackworth, A. K., & Reiter, R. (1992). Characterizing diagnoses and systems. *Artificial Intelligence*, *56*.

de Kleer, J., Raiman, O., & Shirley, M. (1992). One step lookahead is pretty good. In *Readings in Model-Based Diagnosis*.

de Kleer, J., & Williams, B. C. (1987). Diagnosing multiple faults. *Artificial Intelligence*, *32*(1), 97–130.

del Vescovo, C., Parsia, B., Sattler, U., & Schneider, T. (2010). The modular structure of an ontology: An empirical study. In *Workshop on Modular Ontologies (WoMO)*.

Euzenat, J., Meilicke, C., Stuckenschmidt, H., Shvaiko, P., & Trojahn, C. (2011). Ontology alignment evaluation initiative: Six years of experience. In *Journal on Data Semantics XV*.

Euzenat, J., Mochól, M., Shvaiko, P., Stuckenschmidt, H., Sváb, O., Svátek, V., van Hage, W. R., & Yatskevich, M. (2006). Results of the ontology alignment evaluation initiative 2006. In *Int'l Workshop on Ontology Matching (OM)*.

Feldman, A., Provan, G. M., & van Gemund, A. J. C. (2010). A model-based active testing approach to sequential diagnosis. *Journal of Artificial Intelligence Research (JAIR)*, *39*, 301–334.

Feldman, A., Provan, G. M., & van Gemund, A. J. (2008). Computing minimal diagnoses by greedy stochastic search. In *AAAI Conf. on Artificial Intelligence (AAAI)*.

Felfernig, A., Friedrich, G., Jannach, D., & Stumptner, M. (2004). Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence*, *152*(2), 213–234.

Felfernig, A., Mairitsch, M., Mandl, M., Schubert, M., & Teppan, E. (2009). Utility-based repair of inconsistent requirements. In *Int'l Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE)*.

Felfernig, A., Schubert, M., & Zehentner, C. (2012). An efficient diagnosis algorithm for inconsistent constraint sets. *AI EDAM*, *26*(1), 53–62.

Felfernig, A., Teppan, E., Friedrich, G., & Isak, K. (2008). Intelligent debugging and repair of utility constraint sets in knowledge-based recommender applications. In *Int'l Conf. on Intelligent User Interfaces (IUI)*.

Friedrich, G., & Nejdl, W. (1992). Choosing observations and actions in model-based diagnosis/repair systems. *KR*, *92*, 489–498.

Friedrich, G., & Shchekotykhin, K. (2005). A General Diagnosis Method for Ontologies. In *Int'l Semantic Web Conf. (ISWC)*.

Friedrich, G., Stumptner, M., & Wotawa, F. (1999). Model-based diagnosis of hardware designs. *Artificial Intelligence*, *111*(1-2), 3–39.

Fu, X., Qi, G., Zhang, Y., & Zhou, Z. (2016). Graph-based approaches to debugging and revision of terminologies in DL-lite. *Knowledge-Based Systems*, *100*, 1–12.

Gonçalves, R. S., Parsia, B., & Sattler, U. (2012). Performance heterogeneity and approximate reasoning in description logic ontologies. In *Int'l Semantic Web Conf. (ISWC)*.

Gorinevsky, D., Dittmar, K., Mylaraswamy, D., & Nwadiogbu, E. (2002). Model-based diagnostics for an aircraft auxiliary power unit. In *Int'l Conf. on Control Applications*.

Grau, B. C., Horrocks, I., Kazakov, Y., & Sattler, U. (2008a). Modular reuse of ontologies: Theory and practice. *Journal of Artificial Intelligence Research (JAIR)*, *31*, 273–318.

Grau, B. C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., & Sattler, U. (2008b). OWL 2: The next step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, *6*(4), 309–322.

Greiner, R., Smith, B. A., & Wilkerson, R. W. (1989). A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence*, *41*(1), 79–88.

Haenni, R. (1998). Generating diagnoses from conflict sets. In *FLAIRS Conf.*

Horridge, M. (2011). *Justification based Explanation in Ontologies*. Ph.D. thesis, University of Manchester.

Horridge, M., Parsia, B., & Sattler, U. (2008). Laconic and Precise Justifications in OWL. In *Int'l Semantic Web Conf. (ISWC)*.

Horridge, M., Parsia, B., & Sattler, U. (2012). Extracting justifications from BioPortal ontologies. In *Int'l Semantic Web Conf. (ISWC)*.

Hyafil, L., & Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, *5*(1), 15–17.

Jannach, D., & Engler, U. (2010). Toward model-based debugging of spreadsheet programs. In *Joint Conf. on Knowledge-Based Software Engineering (JCKBSE)*.

Jannach, D., Schmitz, T., & Shchekotykhin, K. (2016). Parallel model-based diagnosis on multi-core computers. *Journal of Artificial Intelligence Research (JAIR)*, *55*, 835–887.

Junker, U. (2004). QuickXPlain: Preferred Explanations and Relaxations for Over-Constrained Problems. In *AAAI Conf. on Artificial Intelligence (AAAI)*.

Kalyanpur, A. (2011). *Debugging and Repair of OWL Ontologies*. Ph.D. thesis, University of Maryland, College Park.

Kang, Y.-B., Li, Y.-F., & Krishnaswamy, S. (2012). Predicting reasoning performance using ontology metrics. In *Int'l Semantic Web Conf. (ISWC)*.

Karlsson, D., Nyström, M., & Cornet, R. (2014). Does SNOMED CT post-coordination scale? *Studies in Health Technology and Informatics*, *205*, 1048–1052.

Kazakov, Y., Krötzsch, M., & Simančík, F. (2014). The incredible ELK. *Journal of Automated Reasoning*, *53*(1), 1–61.

Li, L., & Yunfei, J. (2002). Computing minimal hitting sets with genetic algorithm. In *Int'l Workshop on Principles of Diagnosis (DX)*.

Lin, L., & Jiang, Y. (2003). The computation of hitting sets: Review and new algorithms. *Information Processing Letters*, *86*(4), 177–184.

Marques-Silva, J., Heras, F., Janota, M., Previti, A., & Belov, A. (2013). On computing minimal correction subsets. In *Int'l Joint Conf. on Artificial Intelligence*.

Marques-Silva, J., Janota, M., & Belov, A. (2013). Minimal sets over monotone predicates in boolean formulae. In *Int'l Conf. on Computer Aided Verification*.

Marques-Silva, J., Janota, M., & Mencia, C. (2017). Minimal sets on propositional formulae. Problems and reductions. *Artificial Intelligence*, *252*, 22–50.

Marques-Silva, J., Janota, M., Ignatiev, A., & Morgado, A. (2015). Efficient model based diagnosis with maximum satisfiability. In *Int'l Joint Conf. on Artificial Intelligence*.

Marquis, P. (2000). Consequence finding algorithms. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems*.

Meilicke, C. (2011). *Alignment incoherence in ontology matching*. Ph.D. thesis, Universität Mannheim.

Mencia, C., & Marques-Silva, J. (2014). Efficient relaxations of over-constrained CSPs. In *Int'l Conf. on Tools with Artificial Intelligence*.

Metodi, A., Stern, R., Kalech, M., & Codish, M. (2014). A novel SAT-based approach to model based diagnosis. *Journal of Artificial Intelligence Research (JAIR)*, *51*, 377–411.

Moret, B. M. (1982). Decision trees and diagrams. *ACM Computing Surveys (CSUR)*, *14*(4), 593–623.

Mozetič, I. (1991). Hierarchical model-based diagnosis. *Int'l Journal of Man-Machine Studies*, *35*(3), 329–362.

Ng, H. T. (1990). Model-based, multiple fault diagnosis of time-varying, continuous physical devices. In *Conf. on Artificial Intelligence for Applications*.

Out, D.-J., van Rikxoort, R., & Bakker, R. (1994). On the construction of hierarchic models. *Annals of Mathematics and Artificial Intelligence*, *11*(1-4), 283–296.

Parsia, B., Sirin, E., & Kalyanpur, A. (2005). Debugging OWL ontologies. In *Int'l Conf. on World Wide Web (WWW)*.

Pattipati, K. R., & Alexandridis, M. G. (1990). Application of heuristic search and information theory to sequential fault diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics*, *20*(4), 872–887.

Penaloza, R. (2019). Making Decisions with Knowledge Base Repairs. In *Modeling Decisions for Artificial Intelligence (MDAI)*.

Pill, I., & Quaritsch, T. (2012). Optimizations for the boolean approach to computing minimal hitting sets. In *European Conf. on Artificial Intelligence (ECAI)*.

Pill, I., Quaritsch, T., & Wotawa, F. (2011). From conflicts to diagnoses: An empirical evaluation of minimal hitting set algorithms. In *Int'l Workshop on Principles of Diagnosis (DX)*.

Qi, G., & Hunter, A. (2007). Measuring incoherence in description logic-based ontologies. In *The Semantic Web*.

Rector, A. L., Brandt, S., & Schneider, T. (2011). Getting the foot out of the pelvis: Modeling problems affecting use of SNOMED CT hierarchies in practical applications. *Journal of the American Medical Informatics Association (JAMIA)*, *18*(4), 432–440.

Reiter, R. (1987). A Theory of Diagnosis from First Principles. *Artificial Intelligence*, *32*(1), 57–95.

Rodler, P., Jannach, D., Schekotihin, K., & Fleiss, P. (2019). Are query-based ontology debuggers really helping knowledge engineers? *Knowledge-Based Systems*, *179*, 92–107.

Rodler, P. (2015). *Interactive Debugging of Knowledge Bases*. Ph.D. thesis, Alpen-Adria Universität Klagenfurt. arXiv:1605.05950v1

Rodler, P. (2016). Towards better response times and higher-quality queries in interactive knowledge base debugging. Tech. rep., Alpen-Adria Universität Klagenfurt. arXiv:1609.02584v2

Rodler, P. (2018). On active learning strategies for sequential diagnosis. In *Int'l Workshop on Principles of Diagnosis (DX)*.

Rodler, P. (2020a). Reuse, Reduce and Recycle: Optimizing Reiter's HS-tree for Sequential Diagnosis. In *European Conf. on Artificial Intelligence (ECAI)*.

Rodler, P. (2020b). DynamicHS: Optimizing Reiter's HS-tree for Sequential Diagnosis. In *Int'l Workshop on Principles of Diagnosis (DX) (online, http://dx-2020.org/papers/)*.

Rodler, P. (2020c). Too good to throw away: A powerful reuse strategy for Reiter's hitting set tree. In *Symp. on Combinatorial Search (SoCS)*.

Rodler, P. (2020d). Memory-limited model-based diagnosis. arXiv:2010.04282v1

Rodler, P. (2020e). Sound, complete, linear-space, best-first diagnosis search. In *Int'l Workshop on Principles of Diagnosis (DX) (online, http://dx-2020.org/papers/)*. arXiv:2009.12190

Rodler, P. (2020f). Understanding the QuickXPlain algorithm: Simple explanation and formal proof. arXiv:2001.01835v2

Rodler, P., & Eichholzer, M. (2019). On the usefulness of different expert question types for fault localization in ontologies. In *Int'l Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE)*.

Rodler, P., & Elichanova, F. (2020). Do we really sample right in model-based diagnosis? In *Int'l Workshop on Principles of Diagnosis (DX) (online, http://dx-2020.org/papers/)*. arXiv:2009.12178

Rodler, P., & Herold, M. (2018). StaticHS: A variant of Reiter's hitting set tree for efficient sequential diagnosis. In *Symp. on Combinatorial Search (SoCS)*.

Rodler, P., & Schekotihin, K. (2018). Reducing model-based diagnosis to knowledge base debugging. In *Int'l Workshop on Principles of Diagnosis (DX)*.

Rodler, P., & Schmid, W. (2018). On the impact and proper use of heuristics in test-driven ontology debugging. In *Rules and Reasoning - Int'l Joint Conf. (RuleML+RR)*.

Rodler, P., Schmid, W., & Schekotihin, K. (2017). A generally applicable, highly scalable measurement computation and optimization approach to sequential model-based diagnosis. arXiv:1711.05508v1

Rodler, P., Schmid, W., & Schekotihin, K. (2018). Inexpensive cost-optimized measurement proposal for sequential model-based diagnosis. In *Int'l Workshop on Principles of Diagnosis (DX)*.

Rodler, P., Shchekotykhin, K., Fleiss, P., & Friedrich, G. (2013). RIO: Minimizing User Interaction in Ontology Debugging. In *Web Reasoning and Rule Systems (RR)*.

Rodler, P., & Teppan, E. (2020). The scheduling job-set optimization problem: A model-based diagnosis approach.. In *Int'l Workshop on Principles of Diagnosis (DX) (online, http://dx-2020.org/papers/)*. arXiv:1711.05508v1

Romero, A. A., Grau, B. C., & Horrocks, I. (2012). MORE: Modular combination of OWL reasoners for ontology classification. In *Int'l Semantic Web Conf. (ISWC)*.

Sachenbacher, M., Malik, A., & Struss, P. (1998). From electrics to emissions: Experiences in applying model-based diagnosis to real problems in real cars. In *Int'l Workshop on Principles of Diagnosis (DX)*.

Sattler, U., Schneider, T., & Zakharyaschev, M. (2009). Which kind of module should I extract? In *Description Logics*.

Schekotihin, K., Rodler, P., & Schmid, W. (2018a). OntoDebug: Interactive ontology debugging plug-in for Protégé. In *Foundations of Information and Knowledge Systems - Int'l Symp. (FoIKS)*.

Schekotihin, K., Rodler, P., Schmid, W., Horridge, M., & Tudorache, T. (2018b). A Protégé plug-in for test-driven ontology development. In *Int'l Conf. on Biological Ontology (ICBO)*.

Schekotihin, K., Rodler, P., Schmid, W., Horridge, M., & Tudorache, T. (2018c). Test-driven ontology development in Protégé. In *Int'l Conf. on Biological Ontology (ICBO)*.

Schlobach, S., Huang, Z., Cornet, R., & van Harmelen, F. (2007). Debugging incoherent terminologies. *Journal of Automated Reasoning*, *39*(3), 317–349.

Schulz, S., Schober, D., Tudose, I., & Stenzhorn, H. (2010). The pitfalls of thesaurus ontologization–the case of the NCI Thesaurus. In *AMIA Annual Symp.*

Shchekotykhin, K., Friedrich, G., Fleiss, P., & Rodler, P. (2012). Interactive Ontology Debugging: Two Query Strategies for Efficient Fault Localization. *Web Semantics: Science, Services and Agents on the World Wide Web*, *12-13*, 88–103.

Shchekotykhin, K., Friedrich, G., Rodler, P., & Fleiss, P. (2014). Sequential diagnosis of high cardinality faults in knowledge-bases by direct diagnosis generation. In *European Conf. on Artificial Intelligence (ECAI)*.

Shchekotykhin, K., Jannach, D., & Schmitz, T. (2015). MergeXPlain: Fast computation of multiple conflicts for diagnosis. In *Int'l Joint Conf. on Artificial Intelligence (IJCAI)*.

Shchekotykhin, K. M., Rodler, P., Fleiss, P., & Friedrich, G. (2012). Direct computation of diagnoses for ontology alignment. In *Ontology Matching (OM)*.

Shi, L., & Cai, X. (2010). An exact fast algorithm for minimum hitting set. In *Int'l Joint Conf. on Computational Science and Optimization*.

Siddiqi, S., & Huang, J. (2011). Sequential diagnosis by abstraction. *Journal of Artificial Intelligence Research (JAIR)*, *41*, 329–365.

Siddiqi, S. A., Huang, J., et al. (2007). Hierarchical diagnosis of multiple faults. In *Int'l Joint Conf. on Artificial Intelligence (IJCAI)*.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, *5*(2), 51–53.

Steinbauer, G., Wotawa, F., et al. (2005). Detecting and locating faults in the control software of autonomous mobile robots. In *Int'l Joint Conf. on Artificial Intelligence (IJCAI)*.

Stern, R. T., Kalech, M., Feldman, A., & Provan, G. (2012). Exploring the duality in conflict-directed model-based diagnosis. In *AAAI Conf. on Artificial Intelligence (AAAI)*.

Stuckenschmidt, H. (2008). Debugging OWL Ontologies - A Reality Check. In *Int'l Workshop on Evaluation of Ontology-based Tools and the Semantic Web Service Challenge (EON)*.

Torasso, P., & Torta, G. (2006). Model-based diagnosis through OBDD compilation: A complexity analysis. In *Reasoning, Action and Interaction in AI Theories and Systems*.

Vinterbo, S., & Øhrn, A. (2000). Minimal approximate hitting sets and rule templates. *Int'l Journal of Approximate Reasoning*, *25*(2), 123–143.

Williams, B. C., & Ragno, R. J. (2007). Conflict-directed A* and its role in model-based embedded systems. *Discrete Applied Mathematics*, *155*(12), 1562–1595.

Wotawa, F. (2001). A variant of Reiter's hitting-set algorithm. *Information Processing Letters*, *79*(1), 45–51.

Wotawa, F. (2010). Fault localization based on dynamic slicing and hitting-set computation. In *Int'l Conf. on Quality Software*.

Xiangfu, Z., & Dantong, O. (2006). A method of combining SE-tree to compute all minimal hitting sets. *Progress in Natural Science*, *16*(2), 169–174.

Zaman, S., Steinbauer, G., Maurer, J., Lepej, P., & Uran, S. (2013). An integrated model-based diagnosis and repair architecture for ROS-based robot systems. In *IEEE Int'l Conf. on Robotics and Automation*.

Zhao, X. (2016). Deriving minimal hitting-sets for linear conflict sets. In *Int'l Workshop on Principles of Diagnosis (DX)*.

Zhao, X., & Ouyang, D. (2013). A distributed strategy for deriving minimal hitting-sets. In *Int'l Workshop on Principles of Diagnosis (DX)*.

Zolin, E. Complexity of reasoning in description logics. http://www.cs.man.ac.uk/~ezolin/dl/. Accessed: 2020-10-02.